Sprache: [de]
[cs] [en] [es] [fr]
[ja] [ko] [pl] [zh-cn]
[zh-tw]

Weitere Dokumente
Upgrade-MiniFAQ
Ports and Packages
Port Testing Guide
Updating via AnonCVS
Stable
CVSup
Manual pages
Bug Reporting
Mail lists
PF User's Guide

PDF Dateien
FAQ in PDF form

Text Dateien
FAQ in Text form

Zurück zu OpenBSD

OpenBSD

Dokumentation und häufig gestellte Fragen (FAQs)

Häufige Probleme Neue Updates

Diese FAQ sind zusätzliche Informationen zur Dokumentation in den man pages, die sowohl auf einem installierten System, als auch online zu finden sind. Die FAQ deckt die jeweils aktuelle Version von OpenBSD ab, zur Zeit also Version 3.3. Die Entwickler-Version (-current) von OpenBSD wird *nicht* von dieser FAQ behandelt.

Die FAQ ist im PDF-Format und als reine Text-Version verfügbar, und zwar im Verzeichnis **pub/OpenBSD/doc** der FTP mirrors.

Anmerkung: Die FAQs sind teilweise **noch nicht in deutsch übersetzt oder veraltet**, und verweisen daher teilweise auf die englische Version. Betroffen sind davon zur Zeit die FAQs 8,10,13 und 14.

1 - Einführung in OpenBSD

2 - Weitere Informationsquellen zu OpenBSD

3 - Wo man OpenBSD herbekommt

4 - OpenBSD 3.3 Installationsanleitung

5 - Das System aus dem Source-Code erzeugen

10 - System Management

11 - Performance Tuning

12 - Für erfahrene User

13 - IPsec benutzen (IP Security Protokoll)

14 - Disk setup

PF User's Guide

- Konfiguration
- Listen und Makros
- Tabellen
- Optionen
- Scrub
- Queueing
- Network Address Translation
- Traffic Redirection
- Packet Filtering
- Logging
- Anchors und Named (Sub) Rule Sets
- Shortcuts zum Erzeugen von Rule Sets
- Address Pools und Load Balancing
- Performance
- Probleme mit FTP
- Beispiel: Firewall für zu Hause oder ein kleines Büro

Häufige Probleme

- Wie geht das mit dem chroot() Apache?
- Wie führe ich ein Upgrade meines Systems aus?
- Wie führe ich ein Update für mein System aus? und hier
- Packet Filter
- Wie konfiguriere ich ein Multi-Boot System?
- Probleme mit grossen Laufwerken und OpenBSD
- Wie schalte ich meinen Bildschirmschoner ein ?

Neueste Updates

- Neue PF FAQ!
- FAQ 4, Wie bekomme ich ein dmesg(8), um damit ein Problem mit der Installation zu melden ?
- FAQ 4 Anpassen des Installationsprozesses
- FAQ 4, Wie installiere ich eine Anzahl gleichartiger Systeme?
- FAQ 10, Ich mag die Standard-Shell von von root nicht!

- FAQ 10, Was kann ich noch mit ksh machen?
- FAQ 4, Bekannte Installationsprobleme

Der FAQ maintainer ist Nick Holland.
Weitere Mitarbeiter an der FAQ sind Joel Knight, Eric Jackson, Wim Vandeputte und Chris Cappuccio.

Information über die Übersetzung dieser FAQ und den Rest der OpenBSD Website finden sich auf der Translation page.

Fragen und Kommentare bezüglich der FAQ können an faq@openbsd.org gerichtet werden. Allgemeine Fragen über OpenBSD gehören auf die passende Mailingliste.

Back to OpenBSD

Originally [OpenBSD: index.html,v 1.172]
$Translation: index.html,v 1.85 2003/05/04 17:11:08 jufi Exp $
$OpenBSD: index.html,v 1.75 2003/05/04 18:00:52 jufi Exp $
"If you don't find it in the index, look very carefully through the entire catalogue."
Sears, Roebuck, and Co., Consumer's Guide, 1897

OpenBSD

1 - Einführung in OpenBSD

Inhaltsverzeichnis

1.1 - Was ist OpenBSD?

Das OpenBSD Projekt produziert ein frei erhältliches, plattformübergreifendes, auf 4.4BSD-basierendes UNIX-ähnliches Betriebssystem. Unsere Ziele legen den Schwerpunkt auf Korrektheit, Sicherheit, Standardisierung und Portabilität. OpenBSD unterstützt Binäremulation der meisten Programme von SVR4 (Solaris), FreeBSD, Linux, BSD/OS, SunOS und HP-UX.

1.2 - Auf welchen Systemen läuft OpenBSD?

OpenBSD 3.3 läuft auf den folgenden Plattformen:

- i386 - von CD bootfähig
- sparc - von CD bootfähig
- sparc64 - von CD bootfähig
- hp300 - nur FTP
- mac68k - nur FTP
- macppc - von CD bootfähig
- mvme68k - nur FTP
- alpha - nur FTP
- vax

- hppa - Nur FTP* *Neu mit 3.3* *

bootfähig bedeutet, daß OpenBSD direkt von der CD starten wird. Die CDs werden auf einigen Plattformen starten. Details, wie man OpenBSD auf CD beziehen kann, gibt es in Kapitel 3 dieser FAQ.

Vorige Versionen von OpenBSD hatten auch einen Port für:

- amiga - Entfernt nach der Version 3.2
- sun3 - Entfernt nach der Version 2.9
- arc - Entfernt nach der Version 2.3
- mvme88k
- pmax

OpenBSD unterstützt zur Zeit nicht mehr als eine CPU. In der FAQ 8, SMP findest du dazu weitere Informationen.

1.3 - Ist OpenBSD wirklich frei?

OpenBSD is ganz frei. Die Binärdateien sind frei. Die Quelltexte sind frei. Alle Teile von OpenBSD unterliegen entsprechenden Urheberrechtsbestimmungen, die die freie Weiterverbreitung erlauben. Dies beinhaltet auch die Möglichkeit, die meisten Teile von den OpenBSD Quelltexten WIEDERZUVERWENDEN, sei es für private oder kommerzielle Zwecke. OpenBSD unterliegt KEINEN weiteren Beschränkungen als durch die originale BSD Lizenz. Software, die unter strikterer Lizenz verfasst wurde, kann nicht in der regulären OpenBSD Distribution eingebunden werden. Dies dient zur Sicherstellung der weiteren freien Nutzbarkeit von OpenBSD. Zum Beispiel kann OpenBSD frei für den privaten und akademischen Gebrauch, von Regierungsinstitutionen, von non-profit Organisationen und von Unternehmen eingesetzt werden.

Für weitere Informationen über populäre Lizenzen siehe: http://www.openbsd.org/policy.html.

Die Leiter von OpenBSD unterstützen das Projekt hauptsächlich aus ihren eigenen Taschen. Dies beinhaltet die Zeit für Programmieren, die eingesetzte Hardware für die verschiedenen Plattformen, die Netzwerkressourcen, um OpenBSD zu dir zu bringen, und die Zeit, um Fragen zu beantworten und die Fehlerberichte der Benutzer zu untersuchen. Die OpenBSD Entwickler sind nicht gerade reich und auch ein kleiner Beitrag an Zeit, Hardware oder Resourcen macht einen großen Unterschied.

1.4 - Warum sollte ich OpenBSD benutzen?

Neue Benutzer wollen häufig wissen, ob OpenBSD anderen UNIX-ähnlichen Betriebssytemen überlegen ist. Diese Frage ist eigentlich unbeantwortbar und das Thema von zahllosen (und nutzlosen) Debatten. Stelle diese Frage bitte unter keinen Umständen auf einer OpenBSD Mailingliste.

Anbei stehen ein paar Gründe, warum wir glauben, daß OpenBSD ein nützliches Betriebssystem ist. Ob OpenBSD das richtige System für dich ist, kannst nur du entscheiden.

- OpenBSD läuft auf vielen verschiedenen Hardware Plattformen.
- OpenBSD wird von vielen Sicherheitsexperten als das sicherste UNIX-ähnliche Betriebssystem angesehen, was das Resultat eines 1,5 Jahre langen ausführlichen Quelltextsicherheitsaudits

durch ein 10-Mann-Team ist.

- OpenBSD ist ein voll ausgestattetes UNIX-ähnliches Betriebssystem, das im Quelltext ohne Kosten verfügbar ist.
- OpenBSD integriert neueste Sicherheitstechnologien, die geeignet sind, Firewalls und Private Netzwerkdienste in verteilten Umgebungen zu errichten.
- OpenBSD profitiert von der raschen Entwicklung in vielen Bereichen, was die Möglichkeit zur Zusammenarbeit von neu entwickelten Technologien mit der internationalen Gemeinschaft der Programmierer und Endbenutzer betrifft.
- OpenBSD bietet die Möglichkeit für gewöhnliche Menschen, aktiv an der Entwicklung und Erprobung des Produktes teilzuhaben und teilzunehmen.

1.5 - Wie kann ich OpenBSD unterstützen?

Wir sind allen Menschen und Organisationen zu Dank verpflichtet, die zum OpenBSD Projekt beigetragen haben. Sie werden namentlich auf der Spendenseite aufgeführt.

OpenBSD benötigt andauernd bestimmte Arten von Unterstützung von der Benutzergemeinschaft. Wenn du OpenBSD nützlich findest, dann solltest du einen Weg finden, um etwas beizutragen. Solltest du keinen der unten angeführten Vorschläge als für dich angemessen empfinden, dann schlag etwas Besseres vor, indem du eine e-mail an donations@openbsd.org sendest.

- Kaufe ein OpenBSD CD Set. Sie beinhaltet die aktuelle Vollversion von OpenBSD und ist auf vielen Plattformen bootfähig. Der Kauf trägt Geld zur Unterstützung des OpenBSD Projekts bei und reduziert die Belastung der Netzwerkressourcen, um die Distribution via Internet zu verbreiten. Dieses kostengünstige drei-CD Set beinhaltet den vollen Quelltext. Denke daran, auch deine Freunde benötigen ihre eigenen CDs!
- Spende Geld. Das Projekt benötigt andauernd Geld, um Hardware, Netzwerkanbindungen und CD Produktionskosten zu bestreiten. Die Produktion der CDs setzt die Vorfinanzierung durch die OpenBSD Entwickler ohne garantierte Einnahmen voraus. Schick eine e-mail an donations@openbsd.org um herauszufinden, wie du etwas beisteuern kannst. Sogar kleine Spenden machen einen grossen Unterschied.
- Spende Ausrüstung und Hardware. Das Projekt sucht immer normale und spezielle Hardware. Dinge wie IDE und SCSI Festplatten, oder verschiedene Arten von RAM werden immer gerne genommen. Für andere Hardwaretypen wie Computersysteme und Motherboards solltest du den aktuellen Bedarf durch eine e-mail an donations@openbsd.org erfragen.
- Steuere deine Zeit und Fähigkeiten bei. Programmierer, die gerne an Betriebssystemen schreiben, sind natürlich immer willkommen, aber es gibt buchstäblich Dutzende von anderen Möglichkeiten, um nützlich zu sein. Verfolge die Mailinglisten und beantworte Fragen von neuen Benutzern.
- Hilf uns, die Dokumentation auf dem Laufenden zu halten, indem du neues FAQ-Material einbringst (an faq@openbsd.org). Richte eine lokale Benutzergruppe ein und überzeuge deine Freunde von OpenBSD. Zeige deinem Arbeitgeber die Fähigkeiten von BSD für die Arbeit. Wenn du ein Student bist, sprich mit deinen Professoren über OpenBSD als Lehrmittel für EDV. Es ist auch noch erwähnenswert, einen der wichtigsten Wege aufzuzeigen, um dem OpenBSD Projekt "nicht" zu helfen: Beteilige dich nicht an Beschimpfungsorgien (Flamewars) in Usenet newsgroups über andere Betriebssysteme. Dies bringt dem Projekt keine neuen

Benutzer und schadet den wichtigen Beziehungen der Entwickler zu den anderen Entwicklern.

1.6 - Wer betreut das OpenBSD Projekt?

OpenBSD wird von einem Entwicklerteam betreut, das über viele verschiedene Länder verteilt ist. Das Projekt wird von Theo de Raadt koordiniert, der in Kanada wohnhaft ist.

1.7 - Wann erscheint die nächste Version von OpenBSD?

Das OpenBSD Team veröffentlicht alle 6 Monate eine neue Version, mit den angestrebten Daten 1. Mai und 1. November. Mehr Informationen zum Entwicklungsprozess gibt es hier.

[FAQ Index] [Zu Kapitel 2 - Andere Informationsquellen]

 www@openbsd.org

Originally [OpenBSD: faq1.html,v 1.47]
$Translation: faq1.html,v 1.47 2003/05/04 17:11:08 jufi Exp $
$OpenBSD: faq1.html,v 1.40 2003/05/04 18:00:52 jufi Exp $

2 - Andere Informationsquellen über OpenBSD

Inhaltsverzeichnis

2.1 - Interessante WWW Seiten

Die offizielle WWW Seite des OpenBSD Projekts findest du unter: http://www.OpenBSD.org.

Hier kannst du viele wertvolle Informationen über alle Aspekte des OpenBSD Projekts erhalten.

Zusätzlich Hinweise für Laptop Benutzer kann man hier finden:
http://www.monkey.org/openbsd-mobile/.

2.2 - Mailinglisten

Das OpenBSD Projekt betreibt mehrere populäre Mailinglisten, die die Benutzer abonnieren und lesen sollten. Um eine Mailingliste zu abonnieren, schicke eine e-mail an majordomo@openbsd.org. Diese Adresse ist ein automatischer Abonnementservice. Im Textkörper der Nachricht sollte in einer einzigen Zeile der Befehl stehen, um sich in die gewünschte Liste einzugetragen. Zum Beispiel:

```
subscribe announce
```

Der Mailinglistendienst wird dir antworten und dich um Bestätigung bitten. Die Bestätigung schickst du zurück an den Mailinglistendienst. Du bekommst die Option eines Hypertextlink oder das Zurücksenden einer Antwort, um deine Absicht zu bestätigen. Eine Email-Antwort sollte etwas wie das folgende enthalten:

```
accept A56D-70D4-52C3
```

Nach deiner Bestätigung wirst du sofort in die Liste eingetragen und der Mailinglistendienst schickt dir eine Erfolgsbestätigung.

Um sich wieder von einer Liste abzumelden schickst du eine Nachricht an majordomo@openbsd.org. Sie könnte etwa so aussehen:

```
unsubscribe announce
```

Solltest du Schwierigkeiten mit dem Mailinglistensystem haben, dann lies zunächst die Anweisungen und Hinweise, die du durch Schicken einer e-mail an majordomo@openbsd.org mit dem Textkörper "help" erhalten kannst. Einige der beliebtesten OpenBSD Mailinglisten sind:

- **announce** - Wichtige Ankündigungen. Sehr wenige e-mails.
- **security-announce** - Bekanntmachungen von Sicherheitsemfpehlungen.
- **misc** - Benutzerfragen und Antworten. Dies ist die aktivste Liste, und sollte daher für fast alle Fragen genutzt werden.
- **tech** - Technische Themen für OpenBSD Entwickler und fortgeschrittene User. Bitte leite 'neue User' und Installations-bezogene Fragen an *misc*, und nicht an *tech*. Und bitte niemals gleichzeitig in *misc* und *tech*.
- **bugs** - Via sendbug(1) eingereichte Fehler und Diskussionen darüber.

- **source-changes** - Automatisierter Verteiler von Änderungen des CVS der Quelltexte.
- **ports** - Diskussionen über das OpenBSD Ports Systems.
- **ports-changes** - Automatische Mails der ports-spezifischen CVS source tree Änderungen.
- **advocacy** - Diskussionen, um die Nutzung von OpenBSD populärer zu machen.

Bevor du eine Frage auf **misc** oder einer anderen Liste postest, überprüfe bitte zunächst, ob deine Frage nicht schon in den Archiven auftaucht, die meisten Fragen sind schon einmal gerstellt worden. Auch wenn du das Problem zum ersten Mal siehst, ist es anderen dich oftmal schon begegnet, wurden die Mailinglisten vielleicht schon letzte Woche damit überschwemmt, und freuen sich nicht auf eine Wiederholung. Einige Archive und Mailinglisten-Richtlinien finden sich unter http://www.openbsd.org/mail.html

Eine weitere interessante Mailingliste ist openbsd-mobile@monkey.org, die sich mit der Nutzung von OpenBSD auf Notebooks und Laptops befasst.

Um diese Liste zu abonnieren:

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

Das Archiv dieser Liste findest du unter: http://www.monkey.org/openbsd-mobile/archive/

2.3 - Manualseiten

OpenBSD wird von einer ausführlichen Dokumentation in Form von Manualseiten (manual pages) und weiteren, längeren Artikeln, die sich auf spezielle Anwendungen beziehen, begleitet. Um die Manualseiten lesen zu können, müssen die man und misc tar balls installiert sein.

Hier eine Liste der nützlichsten Manualseiten für Anfänger:

- **help(1)** - Hilfe für neue User und Administratoren.
- **afterboot(8)** - Dinge, die man nach dem 1. Start beachten sollte.
- **boot(8)** - Systemstartprozeduren.
- **login.conf(5)** - Format der Passwortkonfigurationsdatei.
- **adduser(8)** - Befehle, um neue Benutzer anzulegen.
- **vipw(8)** - Editieren der Passwortdatei.
- **man(1)** - Anzeigen der on-line Manualseiten.
- **sendbug(1)** - Schicken eines Fehlerberichtes über OpenBSD an die Supportzentrale.
- **disklabel(8)** - Auslesen und Schreiben des Disklayouts.
- **ifconfig(8)** - Konfiguration der Netzwerkadapter.
- **route(8)** - Manipulation der Routingtabellen.
- **netstat(1)** - Anzeigen des Netzwerkstatus.
- **reboot, halt(8)** - Restarten und Stoppen des Systems.
- **shutdown(8)** - Runterfahren des Systems zu einer bestimmten Zeit.
- **boot_config(8)** - Änderung der Kernelkonfiguration beim Starten.

Die OpenBSD Manualseiten sind im Web über http://www.openbsd.org/cgi-bin/man.cgi zu finden, oder auch auf deinem eigenen Computer, wenn du die Datei `man33.tgz` installierst.

Im Allgemeinen kannst du die Manual Seite eines dir namentlich bekannten Befehls erreichen, indem du "man befehl" eingibst. Zum Beispiel: "man vi" bringt dir die man page des vi Editors. Solltest du den genauen Namen des Befehls nicht wissen oder "man befehl" nicht die gewünschte Manual Seite bringen, kannst du mittels "apropos irgendwas" oder "man -k irgendwas" nach dem Namen des Befehls suchen, wobei "irgendwas" möglicherweise in der von dir gesuchten Manual Seite enthalten sein sollte. Zum Beispiel:

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

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

Die Zahlen in Klammern deuten auf das Kapitel, in dem sich die Manual Seite befindet. In einigen Fällen enthalten die Manualseiten den gleichen Befehl in verschieden Kapiteln. Zum Beispiel: Du möchtest das Format der Konfigurationsdatei des cron daemons wissen. Wenn du einmal weißt, in welchem Kapitel sich die gewünschte Manual Seite befindet, kannst du mittels "man n befehl", wobei n die Kapitelnummer ist, die gewünschte Seite direkt aufrufen.

```
# man -k cron
cron (8) - clock daemon
crontab (1) - maintain crontab files for individual users (V3)
crontab (5) - tables for driving cron
# man 5 crontab
```

Zusätzlich zu den UNIX Manualseiten (enthalten in der misc Distribution) gibt es noch einen weiteren Dokumentensatz im `/usr/share/doc` Verzeichnis. Wenn du auch die text Distribution installiert hast, dann kannst du jedes Dokument mittels "make" im entsprechenden Verzeichnis formatieren. Das `psd` Unterverzeichnis beinhaltet die "Programmer's Supplementary Documents" Distribution. Im smm Unterverzeichnis liegt das "System Manager's Manual". Im `usd` Unterverzeichnis findest du die "UNIX User's Supplementary Documents" Distribution. Du kannst "make" in den drei Distributionsunterverzeichnissen starten oder ein spezielles Kapitel einer Distribution auswählen und dort ein "make" in dessen Unterverzeichnis ausführen. Einige der Unterverzeichnisse sind leer. Standardmäßig werden die Dokumente im Postscriptformat ausgegeben, das sich zum Ausdrucken eignet. Die Postscriptausgabe kann sehr groß werden -- etwa 250-300% Zuwachs an Größe. Ohne Postscriptdrucker oder -anzeige kannst du die Dokumente auch fürs Lesen auf einer Terminalanzeige formatieren. In jedem Makefile mußt du nur die Option `-Tascii` bei jedem der groff(1) Befehle setzen (oder händisch ausführen). Einige der Dokumente verwenden `ms` Formatierungsmakros, andere die `me` Makros. Das Makefile jedes Dokumentenunterverzeichnisses (`zb: /usr/share/doc/usd/04.csh/Makefile`) zeigt dir das zu verwendende Format an. Zum Beispiel:

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

Die UNIX Manualseiten sind im Allgemeinen aktueller und vertrauenswürdiger als die formatierten Dokumente, die aber manchmal kompliziertere Anwendungen in größerem Detailumfang als die Manualseiten erklären.

Für viele ist eine ausgedruckte Manual Seite nützlich. Hier folgen die Richtlinien, um eine Manualseite auszudrucken.

Wie kann ich ein "man page" Quelltext anzeigen? (d.h., eine Datei, deren Namen in einer Zahl endet, wie tcpdump.8).

Dies gilt für den gesamten Quelltextbaum. Die "man pages" befinden sich unformatiert im Verzeichnisbaum und werden automatisch durch CVS upgedated. Um sich diese Seiten anzusehen:

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

Wie bekomme ich eine reine Manual Seite ohne Formatierung oder Escapesequenzen?

Dies ist nützlich, um die Manual Seite ohne nicht druckbare Zeichen zu erhalten.
Beispiel:

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

Wie erhalte ich eine postscriptformatierte, druckreife Ausgabe einer Manual Seite?

Beachte, daß `[man_quelltext_datei]` die "man page" Quelltextdatei ist (wahrscheinlich eine Datei, die in einer Zahl endet; z.B. `tcpdump.8`). Die Postscriptversionen der Manualseiten sehen sehr gut aus. Sie können ausgedruckt oder

am Bildschirm mit einem Programm wie gv (GhostView) betrachtet werden. GhostView kannst du in unserem Ports Tree finden. Benutze die folgenden groff(1) Befehlsoptionen um eine PostScript Version von einer OpenBSD System man page zu bekommen:

```
# groff -mdoc -Tps [man_src_file] > outfile.ps
```

Die obige Befehlszeile wird nur für man pages funktionieren, die mit dem mdoc(7) macro package i formatiert wurden, das auch benutzt wird, um die BSD man pages zu formatieren. Um eine PostScript Version einer Software man page zu bekommen, die von einer weiteren Quelle erstellt wurde (entweder selbst hergestellt oder aus den ports(7) oder den packages(7)) , mache folgendes:

```
# groff -Tps -mandoc [man_src_file] > outfile.ps
```

2.4 - Fehler berichten

Bevor du einen Fehler berichtest, solltest du http://www.openbsd.org/report.html lesen.

Richtige Fehlerberichte sind eine der wichtigsten Aufgaben von Endbenutzern. Sehr detaillierte Informationen werden benötigt, um die schwersten Fehler zu diagnostizieren. Entwickler erhalten häufig Fehlerberichte via e-mail wie diese:

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

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

Hoffentlich verstehen die meisten Menschen, warum solche Fehlerberichte gelöscht werden. Jeder Fehlerbericht sollte detaillierte Informationen enthalten. Wenn ein normaler Benutzer wirklich die Untersuchung seines Fehlers wünscht, dann sollte sein Fehlerbericht in etwa so aussehen:

```
From: smartuser@example.com
To: bugs@openbsd.org
Subject: 3.3-beta panics on a SparcStation2

OpenBSD 3.2 installed from an official CDROM installed and ran fine
on this machine.

After doing a clean install of 3.3-beta from an FTP mirror, I find the
system randomly panics after a period of use, and predictably and
quickly when starting X.

This is the dmesg output:

OpenBSD 3.3-beta (GENERIC) #9: Mon Mar 17 12:37:18 MST 2003
    deraadt@sparc.openbsd.org:/usr/src/sys/arch/sparc/compile/GENERIC
real mem = 67002368
avail mem = 59125760
using 200 buffers containing 3346432 bytes of memory
bootpath: /sbus@1,f8000000/esp@0,800000/sd@1,0
mainbus0 (root): SUNW,Sun 4/75
cpu0 at mainbus0: CY7C601 @ 40 MHz, TMS390C602A FPU; cache chip bug
- trap page uncached
cpu0: 64K byte write-through, 32 bytes/line, hw flush cache enabled
memreg0 at mainbus0 ioaddr 0xf4000000
clock0 at mainbus0 ioaddr 0xf2000000: mk48t02 (eeprom)
```

```
timer0 at mainbus0 ioaddr 0xf3000000 delay constant 17
auxreg0 at mainbus0 ioaddr 0xf7400003
zs0 at mainbus0 ioaddr 0xf1000000 pri 12, softpri 6
zstty0 at zs0 channel 0 (console i/o)
zstty1 at zs0 channel 1
zs1 at mainbus0 ioaddr 0xf0000000 pri 12, softpri 6
zskbd0 at zs1 channel 0: reset timeout
zskbd0: no keyboard
zstty2 at zs1 channel 1: mouse
audioamd0 at mainbus0 ioaddr 0xf7201000 pri 13, softpri 4
audio0 at audioamd0
sbus0 at mainbus0 ioaddr 0xf8000000: clock = 20 MHz
dma0 at sbus0 slot 0 offset 0x400000: rev 1+
esp0 at sbus0 slot 0 offset 0x800000 pri 3: ESP100A, 25MHz, SCSI ID 7
scsibus0 at esp0: 8 targets
sd0 at scsibus0 targ 1 lun 0: <SEAGATE, ST1480 SUN0424, 8628> SCSI2 0/direct fixed
sd0: 411MB, 1476 cyl, 9 head, 63 sec, 512 bytes/sec, 843284 sec total
sd1 at scsibus0 targ 3 lun 0: <COMPAQPC, DCAS-32160, S65A> SCSI2 0/direct fixed
sd1: 2006MB, 8188 cyl, 3 head, 167 sec, 512 bytes/sec, 4110000 sec total
le0 at sbus0 slot 0 offset 0xc00000 pri 5: address 08:00:20:13:10:b9
le0: 16 receive buffers, 4 transmit buffers
cgsix0 at sbus0 slot 1 offset 0x0: SUNW,501-2325, 1152x900, rev 11
wsdisplay0 at cgsix0
wsdisplay0: screen 0 added (std, sun emulation)
fdc0 at mainbus0 ioaddr 0xf7200000 pri 11, softpri 4: chip 82072
fd0 at fdc0 drive 0: 1.44MB 80 cyl, 2 head, 18 sec
root on sd0a
rootdev=0x700 rrootdev=0x1100 rawdev=0x1102

This is the panic I got when attempting to start X:

panic: pool_get(mclpl): free list modified: magic=78746572; page 0xfaa93000;
 item addr 0xfaa93000
Stopped at      Debugger+0x4:   jmpl            [%o7 + 0x8], %g0
RUN AT LEAST 'trace' AND 'ps' AND INCLUDE OUTPUT WHEN REPORTING THIS PANIC!
DO NOT EVEN BOTHER REPORTING THIS WITHOUT INCLUDING THAT INFORMATION!
ddb> trace
pool_get(0xfaa93000, 0x22, 0x0, 0x1000, 0x102, 0x0) at pool_get+0x2c0
sosend(0x16, 0xf828d800, 0x0, 0xf83b0900, 0x0, 0x0) at sosend+0x608
soo_write(0xfac0bf50, 0xfac0bf70, 0xfac9be28, 0xfab93190, 0xf8078f24, 0x0)
at soo_write+0x18
dofilewritev(0x0, 0xc, 0xfac0bf50, 0xf7fff198, 0x1, 0xfac0bf70) at
dofilewritev+0x12c
sys_writev(0xfac87508, 0xfac9bf28, 0xfac9bf20, 0xf80765c8, 0x1000, 0xfac0bf70)
at sys_writev+0x50
syscall(0x79, 0xfac9bfb0, 0x0, 0x154, 0xfcffffff, 0xf829dea0) at syscall+0x220
slowtrap(0xc, 0xf7fff198, 0x1, 0x154, 0x1, 0xfac87508) at slowtrap+0x1d8
ddb> ps

   PID   PPID   PGRP    UID  S       FLAGS  WAIT       COMMAND
 27765   8819  29550      0  3         0x86  netio      xconsole
  1668  29550  29550      0  3      0x4086  poll       fvwm
 15447  29550  29550      0  3     0x44186  poll       xterm
  8819  29550  29550     35  3      0x4186  poll       xconsole
  1238  29550  29550      0  3      0x4086  poll       xclock
 29550  25616  29550      0  3      0x4086  pause      sh
  1024  25523  25523      0  3     0x40184  netio      XFree86
*25523  25616  25523     35  2     0x44104             XFree86
 25616  30876  30876      0  3      0x4086  wait       xinit
```

```
 30876  16977  30876      0  3      0x4086  pause      sh
 16977      1  16977      0  3      0x4086  ttyin      csh
  5360      1   5360      0  3        0x84  select     cron
 14701      1  14701      0  3     0x40184  select     sendmail
 12617      1  12617      0  3        0x84  select     sshd
 27515      1  27515      0  3       0x184  select     inetd
  1904      1   1904      0  2        0x84             syslogd
  9125      1   9125      0  3        0x84  poll       dhclient
     7      0      0      0  3    0x100204  crypto_wa  crypto
     6      0      0      0  3    0x100204  aiodoned   aiodoned
     5      0      0      0  3    0x100204  syncer     update
     4      0      0      0  3    0x100204  cleaner    cleaner
     3      0      0      0  3    0x100204  reaper     reaper
     2      0      0      0  3    0x100204  pgdaemon   pagedaemon
     1      0      1      0  3      0x4084  wait       init
     0     -1      0      0  3     0x80204  scheduler  swapper

Thank you!
```

In report.html finden sich weitere Informationen über das Erzeugen und Einsenden von Fehlerberichten (bug reports). Detaillierte Informationen über deine Hardware sind absolut notwendig, wenn du glaubst, der Bug könnte *irgendwie* mit deiner Hardware oder deiner Hardware-Konfiguration zusammenhängen. Normalerweise ist die Ausgabe von dmesg(8) in dieser Hinsicht ausreichend. Eine detaillierte Beschreibung deines Problems ist notwendig. dmesg beschreibt deine Hardware, der Text erklärt warum Samrt User denkt, das System sei defekt, (3.2 lief noch bestens), wie dieser Crash erzeugt wurde (indem er X gestartet hat), und die Ausgabe der Debugger-Befehle "`ps`" und "`trace`". In diesem Fall hat Smart User die Informationen bereitgestellt, die er via Serieller Konsole bekommen hat, wenn du das nicht tun kannst, bist du gezwungen, mittels Stift und Papier die Informationen zum Crash aufzuzeichnen. (Das Beispiel oben war im übrigen ein echtes Problem, und die oben bereitgestellte Information führte dazu, dass dieses Problem, das Sun4c Systeme betraf, behoben werden konnte.)

Wenn ein normaler Benutzer ein funktionierendes OpenBSD System, von dem er einen Fehlerbericht abschicken will, hat, dann sollte er sendbug(1) verwenden, um den Fehlerbericht zu verfassen und zum GNATS Problemerfassungssystem zu senden. Wenn sein System nicht startet, dann sollte er mit sendbug(1) seinen Fehlerbericht an das GNATS Problemerfassungssystem melden. Aber du solltest es, wenn möglich, immer verwenden. Du wirst noch zusätzliche Informationen, wie z. B. was geschah, deine genaue Konfiguration inkludieren müssen, und wie man das Problem reproduzieren kann. Der sendbug(1) Befehl benötigt eine Verbindung zum Internet und einen funktionsfähigen MTA, um den Fehlerbericht per Email versenden zu können.

Nach dem Einsenden eines Bug-Report via sendbug(1) wirst du per Email über den Status informiert. Du kannst auch von Entwicklern kontaktiert werden, die dich nach weiteren Informationen fragen, oder dich bitten, Patches zu testen. Desweiteren kannst du dir auch die Archive der `bugs@openbsd.org` Mailingliste ansehen, Details dazu gibt es auf der Mailinglisten-Seite oder auch den Status in der Bug Report online abfragen, und zwar im Bug Tracking System.

[FAQ Index] [Zu Kapitel 1 - Einführung zu OpenBSD] [Zu Kapitel 3 - OpenBSD beziehen]


www@openbsd.org

Originally [OpenBSD: faq2.html,v 1.59]
$Translation: faq2.html,v 1.40 2003/05/01 12:27:20 jufi Exp $
$OpenBSD: faq2.html,v 1.35 2003/05/01 12:48:46 jufi Exp $

3 - OpenBSD beziehen

Inhaltsverzeichnis

3.1 - OpenBSD CD kaufen

Eine OpenBSD CD zu kaufen ist der allgemein beste Weg, mit deiner Unterstützung zu beginnen. Hier ist die Bestellseite: http://www.openbsd.org/de/orders.html.

Es gibt mehrere gute Gründe, eine OpenBSD CD zu besitzen:

- CD Verkäufe unterstützten die Weiterentwicklung von OpenBSD.
- Die Entwicklung eines plattformübergreifenden Betriebssystems benötigt andauernd Investitionen in Hardware.
- Deine Unterstützung in Form eines CD Kaufes hat einen realen Einfluß auf zukünftige Entwicklungen.
- Die CD beinhaltet Binärdateien (und Quelltexte) für alle unterstützten Plattformen.
- Die CD ist bootfähig auf mehreren Plattformen und man kann mittels CD das Betriebssystem direkt installieren.
- Mit der CD kann man auch die Installation starten, um einen Snapshot zu installieren.
- Die Installation von CD ist schneller! Und Netzwerkressourcen werden eingespart.
- Die OpenBSD CDs beinhalten immer sehr hübsche Aufkleber. Dein System ist nicht komplett, solange du sie nicht hast. Diese Aufkleber kannst du nur durch den Kauf einer CD oder Spenden von Hardware erwerben.

Wenn du eine offizielle Version von OpenBSD installierst, dann solltest du eine CD benutzen.

3.2 - OpenBSD T-Shirts erwerben

Ja, OpenBSD hat T-Shirts zu deinem Tragevergnügen. Hier kannst du sie sehen: http://www.OpenBSD.org/de/tshirts.html. Nett, oder? :)

3.3 - Gibt es ein OpenBSD ISO image?

Einige andere OpenSource Betriebssysteme werden üblicherweise als CD-ROM Images verbreitet. Das ist bei OpenBSD *nicht* der Fall.

Das OpenBSD Projekt stellt die offiziellen ISO Images nicht zur Verfügung, die als Master für die offiziellen CDs benutzt werden. Der Grund dafür ist einfach, daß wir gerne die CDs verkaufen möchten, und zwar um die weitere Entwicklung von OpenBSD sicherzustellen. Das offizielle OpenBSD CD-ROM Layout unterliegt dem Copyright von Theo de Raadt. Theo erlaubt es niemandem, die Images der offiziellen CD weiter zu verbreiten. Als weiterer Anreiz zum Kauf der CDs gibts es zusätzlich noch 'artwork' und Aufkleber.

Denk aber daran, dass nur das CD Layout unter Copyright liegt, OpenBSD selbst ist frei. Nichts hindert irgendjemanden daran, sich OpenBSD zu besorgen und seine eigene CD zu machen. Wenn du aus irgendeinem Grund ein CD Image herunterladen willst, sieh im Archiv der Mailinglisten nach, dort gibt es Quellen. Natürlich gibt es dabei nur zwei Möglichkeiten: Entweder verletzen die im Internet erhältlichen Images das Copyright von Theo de Raadt oder sind einfach keine offiziellen Images. Die Quelle von inoffiziellen Images sind entweder vertrauenswürdig oder eben nicht, die Entscheidung liegt bei dir. Wir schlagen einfach allen Leuten vor, die OpenBSD umsonst haben wollen, die FTP Installationsoption zu wählen. Diejenigen, die eine bootbare CD für ihr System benötigen, können bootdisk ISO images (namens `cd33.iso`) für eine Reihe von Plattformen herunterladen, die es dann erlauben, den Rest des Systems via FTP zu installieren. Diese ISO-Images haben nur eine Grösse von wenigen MB, und enthalten nichts weiter als die Installationstools, aber nicht die wirklichen Dateien.

3.4 - Downloaden via FTP oder AFS

Es gibt zahlreiche internationale FTP Server, die OpenBSD Versionen und Snapshots führen. AFS Zugang ist auch möglich. Du solltest immer den dir nächsten Server wählen. Bevor du mit dem Runterladen einer Version oder eines Snapshots beginnst, solltest du mittels ping(8) und traceroute(8) feststellen, welcher Server für dich am schnellsten ist. Klarerweise ist das offizielle OpenBSD CD Set immer näher als jeder Server. Zugangsinformationen findest du hier:

http://www.openbsd.org/de/ftp.html.

3.5 - Aktueller Quelltext (CVS)

Die Quelltexte von OpenBSD dürfen frei verbreitet werden und sind frei erhältlich. Im Allgemeinen ist der beste Weg, den Verzeichnisbaum mit den aktuellen Quelltexten aufzubauen, die Quelltexte von der letzten CD zu installieren und sie dann mittels AnonCVS regelmäßig zu erneuern. Informationen über AnonCVS findest du hier:

http://www.openbsd.org/de/anoncvs.html.

sowie auch in der FAQ 8, CVS.

Sollte deine Netzwerkanbindung für AnonCVS unzureichend sein oder dein Internetzugang besteht via UUCP, kannst du die Quelltexte mittels CTM auf dem laufenden halten. Solltest du dich in dieser Situation befinden, dann ist das Beginnen mit der CD umso wichtiger. Informationen über CTM findest du hier:

http://www.openbsd.org/de/ctm.html.

Eine weitere Möglichkeit an den Quelltext zu kommen bietet dir das Webinterface "cvsweb" an:
http://www.openbsd.org/cgi-bin/cvsweb/.

[FAQ Index] [Zu Kapitel 2 - Andere Informationsquellen] [Zu Kapitel 4 - Installationsleitfaden]


www@openbsd.org

Originally [OpenBSD: faq3.html,v 1.39]
$Translation: faq3.html,v 1.34 2003/05/04 13:17:17 jufi Exp $
$OpenBSD: faq3.html,v 1.30 2003/05/04 13:44:33 jufi Exp $

OpenBSD

4 - OpenBSD 3.3 Installation Guide

Table of Contents

4.1 - Overview of the OpenBSD installation procedure

OpenBSD has a very robust and adaptable text-based install procedure. In addition to its robustness, the install procedure can be done using 1 floppy disk. Most architectures have a similar installation procedure; however there are some differences in details. In all cases, you are urged to read the platform-specific INSTALL document in the platform directory on the CDROM or FTP sites (for example, `INSTALL.i386`, `INSTALL.mac68k` or `INSTALL.sparc`).

On most architectures, you have several installation options, including FTP, CDROM, and local disk files. One option **not** supported is downloading an ISO image to make your own official CDROM. CDROMs are available for purchase, however. Many platforms now have a bootable CD-ROM image (`cd33.iso`) allowing creation of a bootable CD-ROM for systems which do not support booting from floppy drives. This is strictly a boot media, however, install files must still be retrieved via FTP or other source.

4.1.1 - Supported OpenBSD Architectures

OpenBSD 3.3 supports a number of architectures listed below in alphabetical order. Please refer to each architecture's page for specific information on what each architecture supports.

- alpha - DEC Alpha-based machines.
- hp300 - Hewlett-Packard HP300/HP400 machines.
- hppa - HP PA-RISK based systems **new for 3.3**
- i386 - Intel i386 compatibles.
- mac68k - Most MC680x0-based Apple Macintosh models.
- mvme68k - Motorola MVME147/16x/17x 68K VME cards.
- macppc - Support for Apple based PowerPC systems.
- sparc - SPARC Platform by Sun Microsystems.
- sparc64 - Sun's UltraSPARC systems.
- vax - DEC's VAX computers.

4.1.2 - Supported Installation Media

OpenBSD can be installed from multiple media types. The most common and architecture independent options are laid out below. These options can be used after booting from an OpenBSD CD-ROM, floppy disk or ramdisk kernel.

CD-ROM	To do a CD-ROM install, you must have either purchased an Official OpenBSD CD-ROM or created your own OpenBSD CD. This is usually the easiest way to install an OpenBSD system. **NOTE:** Official OpenBSD CDs are bootable if your BIOS supports it.
FTP	This installation option allows you to install OpenBSD by downloading the installation packages in realtime over the network.
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 i386 install disk.

4.1.3 - Creating bootable OpenBSD install media.

To create an installation floppy image you must either download the correct boot floppy image from one of the OpenBSD distribution sites or copy the image from an Official OpenBSD CD-ROM. You can find a list of FTP servers at the OpenBSD FTP Distribution page. Most architectures have one or more boot floppy images to choose from; different images are for different hardware variations. The differences between the i386 platform installation floppies will be outlined below. For the other architectures with multiple boot floppies, see the INSTALL document in the respective FTP directory. For those with only one, just download the respective `floppy33.fs` image.

The i386 platform has five separate installation disk images to choose from:

- **`floppy33.fs`** (Desktop PC) supports many PCI and ISA NICs, IDE and simple SCSI adapters and some PCMCIA support.
- **`floppyB33.fs`** (Servers) supports many RAID controllers, and some of the less common SCSI adapters. However,

support for many standard SCSI adapters and many EISA and ISA NICS has been removed.

- **`floppyC33.fs`** (Laptops) supports the Cardbus and PCMCIA devices found in many laptops.
- **`cdrom33.fs`** is, in effect a combination of all three boot disks. It can be used to make a bootable 2.88M floppy, or more commonly, as a boot image for a custom recordable CD.
- **`cd33.iso`** is an ISO9660 image that can be used to create a bootable CD with most popular CDR creation software on most platforms. This is `cdrom33.fs` in a "ready to record" format.

Most i386 users will just use the `floppy33.fs` installation floppy.

Yes, there may be situations where one install disk is required to support your SCSI adapter and another disk is required to support your network adapter. Fortunately, this is a rare event, and can usually be worked around.

Once you have the correct floppy image, you need to get a clean floppy disk. If there are ANY bad sectors on the floppy disk, the installation will most likely fail.

4.1.3.1 - Creating floppies on Unix.

To create a formatted floppy, use the fdformat(1) command to both format and check for bad sectors.

```
# fdformat /dev/fd0a
Format 1440K floppy `/dev/fd0a'? (y/n): y
processing VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV done.
```

If your output is like the above example, then your disk is OK. However, if you do not see ALL "V"'s then your disk is most likely bad, and you should try a new one.

Once you have a clean, formatted floppy it is time to write the installation image to floppy. For this, you can use the dd(1) utility. An example usage of `dd(1)` is below:

```
# dd if=floppy33.fs of=/dev/rfd0c bs=32k
```

Once the image is written, check to make sure that the copied image is the same as the original with the cmp(1) command. If the diskette is identical to the image, you will just see another prompt.

```
# cmp /dev/rfd0c floppy33.fs
```

4.1.3.2 - Creating floppies on DOS/Windows.

If you are creating this image on the Windows/DOS platform you can get tools mentioned below from the tools directory on any of the ftp mirrors, or in `3.3/tools` directory on CD1 of the OpenBSD CD set. For users of DOS/Windows 9x, *rawrite* can be used to write your boot floppy.

To prepare a floppy in MS-DOS or Windows, simply use the native formatting tools.

To write the installation image to the prepared floppy you can use *rawrite*, *fdimage*, or *ntrw*. *rawrite* will not work on Windows NT, 2000 or XP.

Note that `FDIMAGE.EXE` and `RAWRITE.EXE` are both MS-DOS applications, and thus, are limited to MS-DOS's "8.3" file naming convention. As `floppy33B.fs` and `floppy33C.fs` have longer file names, you will have to find out how your system stored the file in "8.3 format" before using `FDIMAGE.EXE` or `RAWRITE.EXE` to make your boot floppies.

Example usage of *rawrite*:

```
C:\> rawrite
RaWrite 1.2 - Write disk file to raw floppy diskette

Enter source file name: floppy33.fs
Enter destination drive: a
Please insert a formatted diskette into drive A: and press -ENTER- : Enter
```

Example Usage of *fdimage*:

```
C:\> fdimage -q floppy33.fs a:
```

Example Usage of *ntrw*:

```
C:\> ntrw floppy33.fs a:
3.5", 1.44MB, 512 bytes/sector
bufsize is 9216
1474560 bytes written
```

4.1.4 - Booting OpenBSD Installation Images.

This section is initially broken down into architecture dependent sections for popular architectures that OpenBSD supports. This is so we can properly instruct each user on what to do on their respective platform.

Booting i386

Booting an install image on the i386 architecture is nothing new to most people. If you are using the floppy disk, simply stick the floppy into your floppy drive and boot your system. Your install image will automatically load (assuming floppy boot is enabled in your BIOS). If you are planning on booting from CD, you must go into your systems BIOS and set the boot options to allow booting from CD. Some older BIOSs do not have this option, and you must use a floppy for booting your installation image. Don't worry though; even if you boot from floppy you can still install from the CD.

Booting sparc

NOTE: On the sparc64, only the SBus machines (Ultra 1, Ultra 2) are bootable from floppy.

To boot from floppy, place your floppy disk with the OpenBSD installation image on it into your floppy drive. Then use the following command to boot from your floppy:

```
ok boot floppy
```

To boot from CD-ROM, place your OpenBSD CD-ROM disk into your drive. If your Sun only has one CD-ROM drive, then just go to the boot prompt, where you can '`boot cdrom`':

```
ok boot cdrom
```

Of course, this will only work in new command mode. If you are at the old command mode prompt (a right arrow), type 'n' for the new command mode. (If you are using an old sparc that is pre-sun4c, you probably don't have a new command mode. In this case, you need to experiment.) If you have multiple CD-ROM devices, you need to boot from the correct one. Try `probe-scsi` from the new command mode.

```
ok probe-scsi

Target 0
  Unit 0   Disk     QUANTUM LIGHTNING 365S
Target 1
  Unit 0   Removable Disk     QUANTUM EMPIRE_1080S
Target 3
  Unit 0   Removable Disk     Joe's CD ROMs
```

Figure out which disk is the CD ROM you want to boot from. Note the target number.

```
ok boot /sbus/esp/sd@X,0
```

4.2 - Preinstallation Checklist

Before you start your install, you should have some idea what you want to end up with. You will want to know the following items, at least:

- Machine Name
- Hardware installed and available
  - Verify compatibility with your platform's hardware compatibility page
  - If ISA, you also need to know hardware settings, and confirm they are as OpenBSD requires.
- Install method to be used (CDROM, FTP, etc.)
- Desired disk layout
  - Does existing data need to be saved elsewhere?
  - Will OpenBSD co-exist on this system with another OS? If so, how both will be booted? Will you need to install a "boot manager"?
  - Will the entire disk be used for OpenBSD, or will an existing (or future) partition have to be worked around?
  - How do you wish to sub-partition the OpenBSD part of your disk?
- Network settings, if not using DHCP:
  - Domain Name
  - Domain Name Server(s) (DNS servers)
  - IP addresses and subnet masks for each NIC
  - Gateway address
- Will you be running X on this system?

4.3 - Starting the Install

Whatever your means of booting is, it is now time to use it. During the boot process, the kernel and all of the programs used to install OpenBSD are loaded into memory. The most common problem when booting is a bad floppy disk or a drive alignment problem. The boot floppy is quite tightly packed -- any bad spot will cause problems.

At almost any point during the OpenBSD install process, you can terminate the current install attempt by hitting CTRL-C and can restart it without rebooting by running `install` at the shell prompt.

When your boot is successful, you will see a lot of text messages scroll by. This text, on many architectures in white on blue, is the dmesg, the kernel telling you what devices have been found, and where. Don't worry about remembering this text, as a copy is saved as `/var/run/dmesg.boot`. On some architectures, SHIFT+PGUP will let you examine text that has scrolled off the screen.

Then, you will see the following:

```
rootdev=0x1100 rrootdev=0x2f00 rawdev=0x2f02
erase ^?, werase ^W, kill ^U, intr ^C, status ^T
(I)nstall, (U)pgrade or (S)hell? i
```

And with that, we reach our first question. Most of the time, you have the three options shown:

- **Install:** load OpenBSD onto the system, overwriting whatever may have been there. Note that it is possible to leave some partitions untouched in this process, such as a `/home`, but otherwise, assume everything else is overwritten.
- **Upgrade:** Install a new set of install files on this machine, but do not overwrite any configuration information, user data, or additional programs. No disk formatting is done, nor are the /etc or /var directories overwritten. You will not be given the option of installing the `etc33.tgz` file. After the install, you will have to manually merge the changes of `etc33.tgz` into your system before you can expect it to be fully functional. This is an important step which must be done, as otherwise certain key services (such as pf(4)) may not start.
  NOTE: The Upgrade process is not designed to skip releases! While this will often work, it is not supported. For OpenBSD 3.3, upgrading 3.2 to 3.3 is the only supported upgrade. If you have to upgrade from an older version, a complete reinstall is recommended.
- **Shell:** Sometimes, you need to perform repairs or maintenance to a system which will not (or should not) boot to a

normal kernel. This option will allow you to do maintenance to the system.

On occasion, you will not see the "Upgrade" option listed. After a flag day event, it is not possible to directly upgrade; one must rebuild the system from scratch. An example of this was the OpenBSD/sparc platform, which switched executable formats from a.out to ELF between 3.1 and 3.2, so a user upgrading to 3.2 or 3.3 on their SPARC must completely reload their system.

In this example, we will do an install, but the upgrade process is similar.

```
Welcome to the OpenBSD/i386 3.3 install program.

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

Specify terminal type: [vt220]
Do you wish to select a keyboard encoding table? [n] y
```

In most cases, the default terminal type is appropriate; however if you are using a serial console for install, don't just take the default, respond appropriately.

If you do not select a keyboard encoding table, a US keyboard layout will be assumed.

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

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

Proceed with install? [n] y
```

If you take the default here, the install process will terminate and drop you to a shell prompt.

4.3.1 - Setting up disks during installation.

Setting up disks in OpenBSD varies a bit between platforms. For i386 and macppc, disk setup is done in two stages. First, the OpenBSD slice of the hard disk is defined using fdisk(8), then that slice is subdivided into OpenBSD partitions using disklabel(8).

Some users may be a little confused by the terminology used here. It will appear we are using the word "partition" in two different ways. This observation is correct. There are two layers of partitioning in several OpenBSD platforms, the first, one could consider the Operating System partitioning, which is how multiple OSs on one computer mark out their own space on the disk, and the second one is how the OpenBSD partition is sub-partitioned into individual filesystems. The first layer is visible as a disk partition to DOS, Windows, and any other OS that can coexist with other Operating Systems on the IBM AT descended machines. The second layer of partitioning is visible only to OpenBSD and those OSs which can directly read an OpenBSD filesystem.

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

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

Available disks are: wd0.
Which one is the root disk? (or done) [wd0] Enter
```

The root disk is the disk the system will boot from, and normally where swap space resides. Usually, this will be the default -- if it isn't, you will need to know how to force your computer to boot from a non-standard disk. IDE disks will show up as

`wd0`, `wd1`, etc., SCSI disks and RAID devices will show up as `sd0`, `sd1`, and so on. All the disks OpenBSD can find are listed here -- if you have drives which are not showing up, you have unsupported or improperly configured hardware.

```
Do you want to use *all* of wd0 for OpenBSD? [no] Enter
```

If you say "yes" to this question, the entire disk will be allocated to OpenBSD. This will result in a simple Master Boot Record and partition table being written out to disk -- one partition, the size of the entire hard disk, set to the OpenBSD partition type, and flagged as the bootable partition. This will be a common option for most production uses of OpenBSD; however, there are some systems this should not be done on. Many Compaq systems, some Dell and other systems use a "maintenance" partition, which should be kept intact. If your system has any other partitions of any type you do not wish to erase, do not select "yes" to the above question.

For the sake of this example, we will assume the disk is to be split between OpenBSD and a pre-existing Windows 2000 partition, so we take the default of "no", which will take us into the fdisk(8) program. You can also get more information on fdisk(8) here.

Important Note: Users with a large hard disk (larger than 8G on a newer i386, though on older machines and different platforms, often much smaller) will want to see this section before going any further.

```
You will now create a single MBR partition to contain your OpenBSD data. This
partition must have an id of 'A6'; must *NOT* overlap other partitions; and
must be marked as the only active partition.

The 'manual' command describes all the fdisk commands in detail.

Disk: wd0       geometry: 2586/240/63 [39100320 Sectors]
Offset: 0       Signature: 0xAA55
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
------------------------------------------------------------------------
*0: 06    0   1  1 -  202 239 63 [          63:     3069297 ] DOS > 32MB
 1: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
 2: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
 3: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
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)
        setpid          Set the identifier of a given table entry
        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>
```

A few commands are worthy of elaboration:

- **`r`** or **`reinit:`** Clears existing partition table, makes one big OpenBSD partition, flags it active. Equivalent to saying "yes" to the "`use *all* of ...`" question.
- **`p`** or **`print:`** Displays the current partition table in sectors. "`p m`" will show the partition table in megabytes, "`p g`" will show it in gigabytes.
- **`e`** or **`edit:`** edit or alter a table entry.
- **`f`** or **`flag:`** Marks a partition as the active partition, the one that will be booted from

- **`exit`** and **`quit:`** Careful on these, as some users are used to "exit" and "quit" having opposite meanings.

It is worth pointing out once again, a screwup here will result in significant data loss. If you are going to do this on a drive with important data, it might be worth practicing on a "disposable" drive, in addition to having a good backup.

Our drive here has a 1.5G partition for Windows 2000 (using the FAT filesystem). Looking at the info from the above display, we can see that the Windows partition occupies through cylinder 202 on the drive. So, we are going to allocate the rest of the disk to OpenBSD, starting at cylinder 203. You could also calculate OpenBSD's starting sector of 3069360 by adding the existing partition's starting sector (63) and its size (3069297).

You can edit the drive layout in either Cylinder/Heads/Sectors form or just raw sectors. Which is easier depends upon what you are doing; in this case, working around an existing partition, using CHS format will probably be easier.

```
fdisk: 1> e 1
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
-------------------------------------------------------------------------
 1: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
Partition id ('0' to disable)  [0 - FF]: [0] (? for help) a6
Do you wish to edit in CHS mode? [n] y
BIOS Starting cylinder [0 - 2585]: [0] 203
BIOS Starting head [0 - 239]: [0] Enter
BIOS Starting sector [1 - 63]: [0] 1
BIOS Ending cylinder [0 - 2585]: [0] 2585
BIOS Ending head [0 - 239]: [0] 239
BIOS Ending sector [1 - 63]: [0] 63
fdisk:*1> p
Disk: wd0       geometry: 2586/240/63 [39100320 Sectors]
Offset: 0       Signature: 0xAA55
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
-------------------------------------------------------------------------
*0: 06    0   1  1 -  202 239 63 [          63:     3069297 ] DOS > 32MB
 1: A6  203   0  1 - 2585 239 63 [     3069360:    36030960 ] OpenBSD
 2: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
 3: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
fdisk:*1> p m
Disk: wd0       geometry: 2586/240/63 [19092 Megabytes]
Offset: 0       Signature: 0xAA55
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
-------------------------------------------------------------------------
*0: 06    0   1  1 -  202 239 63 [          63:        1499M] DOS > 32MB
 1: A6  203   0  1 - 2585 239 63 [     3069360:       17593M] OpenBSD
 2: 00    0   0  0 -    0   0  0 [           0:           0M] unused
 3: 00    0   0  0 -    0   0  0 [           0:           0M] unused
fdisk:*1>
```

Note that the prompt changed to include an asterisk ('`*`') to indicate you have unsaved changes. As we can see from the output of `p m` we have not altered our Windows partition, we have successfully allocated the rest of the drive for OpenBSD, and the partitions do not overlap. We are in business. Almost.

What we haven't done is flagged the partition as active so the machine will boot OpenBSD on the next reboot:

```
fdisk:*1> f 1
Partition 1 marked active.
fdisk:*1> p
Disk: wd0       geometry: 2586/240/63 [39100320 Sectors]
Offset: 0       Signature: 0xAA55
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
------------------------------------------------------------------------
 0: 06    0   1  1 -  202 239 63 [          63:     3069297 ] DOS > 32MB
*1: A6  203   0  1 - 2585 239 63 [     3069360:    36030960 ] OpenBSD
 2: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
 3: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
fdisk:*1>
```

And now, we are ready to save our changes:

```
fdisk:*1> w
Writing MBR at offset 0.
wd0: no disk label
fdisk: 1> q
```

Creating a disklabel

The next step is to use disklabel(8) to slice up the OpenBSD partition. More details on using disklabel(8) can be found in FAQ 14, disklabel.

```
Here is the partition information you chose:

Disk: wd0       geometry: 2586/240/63 [39100320 Sectors]
Offset: 0       Signature: 0xAA55
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
------------------------------------------------------------------------
*0: 06    0   1  1 -  202 239 63 [          63:     3069297 ] DOS > 32MB
 1: A6  203   0  1 - 2585 239 63 [     3069360:    36030960 ] OpenBSD
 2: 00    0   0  0 -    0   0  0 [           0:           0 ] unused
 3: 00    0   0  0 -    0   0  0 [           0:           0 ] unused

You will now create an OpenBSD disklabel inside the OpenBSD MBR
partition. The disklabel defines how OpenBSD splits up the MBR partition
into OpenBSD partitions in which filesystems and swap space are created.

The offsets used in the disklabel are ABSOLUTE, i.e. relative to the
start of the disk, NOT the start of the OpenBSD MBR partition.

disklabel: no disk label
WARNING: Disk wd0 has no label. You will be creating a new one.

# using MBR partition 1: type A6 off 3069360 (0x2ed5b0) size 36030960 (0x225c9f0)

Treating sectors 3069360-39100320 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)
> ?
Available commands:
        p [unit]  - print label.
        M         - show entire OpenBSD man page for disklabel.
```

```
        e         - edit drive parameters.
        a [part]  - add new partition.
        b         - set OpenBSD disk boundaries.
        c [part]  - change partition size.
        d [part]  - delete partition.
        D         - set label to default.
        g [d|b]   - Use [d]isk or [b]ios geometry.
        m [part]  - modify existing partition.
        n [part]  - set the mount point for a partition.
        r         - recalculate free space.
        u         - undo last change.
        s [path]  - save label to file.
        w         - write label to disk.
        q         - quit and save changes.
        x         - exit without saving changes.
        X         - toggle expert mode.
        z         - zero out partition table.
        ? [cmnd]  - this message or command specific help.
Numeric parameters may use suffixes to indicate units:
        'b' for bytes, 'c' for cylinders, 'k' for kilobytes, 'm' for megabytes,
        'g' for gigabytes or no suffix for sectors (usually 512 bytes).
        Non-sector units will be rounded to the nearest cylinder.
Entering '?' at most prompts will give you (simple) context sensitive help.
>
```

Again, a few of these commands could use a little elaboration:

- **`p`** - displays (prints) the current disklabel to the screen, and you can use the modifiers `k`, `m` or `g` for kilobytes, megabytes or gigabytes.
- **`D`** - Clears any existing disklabel, creates a new default disklabel which covers just the current OpenBSD partition. This can be useful if the disk previously had a disklabel on it, and the OpenBSD partition was recreated to a different size -- the old disk label may not get deleted, and may cause confusion.
- **`m`** - Modifies an existing entry in a disklabel. Do not over estimate what this will do for you. While it may alter the size of a disklabel partition, it will NOT alter the filesystem on the drive. Using this option and expecting it to resize existing partitions is a good way of losing large amounts of data.

Slicing up your disk properly is important. The answer to the question, "How should I partition my system?" is "Exactly how you need it". This will vary from application to application. There is no universal answer. If you are unsure of how you want to partition your system, see this discussion.

In this system, we have over 17G available for OpenBSD. That's a lot of space, and it isn't likely we will need most of it. So, we will deliberately not use absolute minimum sizes. We would rather have a few hundred megabytes of unused space than a kilobyte too little.

On the root disk, the two partitions 'a' and 'b' **must** be created. The installation process will not proceed until these two partitions are available. 'a' will be used for the root filesystem (/) and 'b' will be used as swap space.

After a little thought, we decide to create just enough partitions to allow the creation of the recommended separate filesystems (`/, /tmp, /var, /usr, /home`) along with a swap partition:

- ***wd0a:*** / (root) - 150M. Should be more than enough.
- ***wd0b:*** (swap) - 300M.
- ***wd0d: /tmp*** - 120M. `/tmp` is used for building some software, 120M will probably be enough for most things.
- ***wd0e: /var*** - 80M. If this were to be a web or mail server, we'd have made this partition much larger, but, that's not what we are doing.
- ***wd0g: /usr*** - 2G. We want this partition to be large enough to load quite a few user applications, plus be able to update and rebuild the system if desired or needed. The Ports tree will be here as well, which will take almost 100M of this space before ports are built.
- ***wd0h: /home*** - 4G. This will allow plenty of user file space.

Now, if you add those up, you will see over 10G of space is unused! Unused space won't hurt anything, and it gives us flexibility to enlarge things in the future if need be. Need more */tmp*? No problem, create a new one in the unused space,

change */etc/fstab* and problem solved.

```
> p m
device: /dev/rwd0c
type: ESDI
disk: ESDI/IDE disk
label: ST320011A
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 16383
total sectors: 39102336
free sectors: 36030960
rpm: 3600

16 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a: 17593.2M  1498.7M    unused        0     0
  c: 19092.9M     0.0M    unused        0     0
  i:  1498.7M     0.0M     MSDOS
> d a
> a a
offset: [3069360] Enter
size: [36030960] 150M
Rounding to nearest cylinder: 307440
FS type: [4.2BSD] Enter
mount point: [none] /
> a b
offset: [3376800] Enter
size: [35723520] 300M
Rounding to nearest cylinder: 614880
FS type: [swap] Enter
> a d
offset: [3991680] Enter
size: [35108640] 120m
Rounding to nearest cylinder: 245952
FS type: [4.2BSD] Enter
mount point: [none] /tmp
> a e
offset: [4237632] Enter
size: [34862688] 80m
Rounding to nearest cylinder: 164304
FS type: [4.2BSD] Enter
mount point: [none] /var
> a g
offset: [4401936] Enter
size: [34698384] 2g
Rounding to nearest cylinder: 4194288
FS type: [4.2BSD] Enter
mount point: [none] /usr
> a h
offset: [8596224] Enter
size: [30504096] 4g
Rounding to nearest cylinder: 8388576
FS type: [4.2BSD] Enter
mount point: [none] /home
> p m
device: /dev/rwd0c
```

```
type: ESDI
disk: ESDI/IDE disk
label: ST320011A
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 16383
total sectors: 39102336
free sectors: 22115520
rpm: 3600

16 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a:   150.1M  1498.7M    4.2BSD     1024  8192    16  # /
  b:   300.2M  1648.8M      swap
  c: 19092.9M     0.0M    unused        0     0
  d:   120.1M  1949.1M    4.2BSD     1024  8192    16  # /tmp
  e:    80.2M  2069.2M    4.2BSD     1024  8192    16  # /var
  g:  2048.0M  2149.4M    4.2BSD     1024  8192    16  # /usr
  h:  4096.0M  4197.4M    4.2BSD     1024  8192    16  # /home
  i:  1498.7M     0.0M     MSDOS
> q
Write new label?: [y] Enter
```

You will note there is a *c* partition we seem to have ignored. This partition is your entire hard disk; don't attempt to alter it. You will also note the *i* partition wasn't defined by us; this is the pre-existing Windows 2000 partition. Partitions are not assigned any particular letters -- with the exception of *a* (root), *b* (swap) and *c* (entire disk), the rest of the partitions (through letter *p*) are available for use as you desire.

If you look closely at the output of the disklabel, you will note that your drive RPM rating is probably wrong. This is historical; the drive speed is not used in any way by the system. Do not worry about it.

Configuring your mount points and formatting your filesystems

Now comes the final configuration of your mount points. If you configured the mount points through disklabel(8), this step consists of just verifying your selections; otherwise, you can specify them now.

```
The root filesystem will be mounted on wd0a.
wd0b will be used for swap space.
Mount point for wd0d (size=122976k), none or done? [/tmp] Enter
Mount point for wd0e (size=82152k), none or done? [/var] Enter
Mount point for wd0g (size=2097144k), none or done? [/usr] Enter
Mount point for wd0h (size=4194288k), none or done? [/home] Enter
Mount point for wd0d (size=122976k), none or done? [/tmp] done
Done - no available disks found.

You have configured the following partitions and mount points:

wd0a /
wd0d /tmp
wd0e /var
wd0g /usr
wd0h /home

The next step creates a filesystem on each partition, ERASING existing data.
Are you really sure that you're ready to proceed? [n] y
/dev/rwd0a:     307440 sectors in 305 cylinders of 16 tracks, 63 sectors
        150.1MB in 20 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/rwd0d:     245952 sectors in 244 cylinders of 16 tracks, 63 sectors
```

```
        120.1MB in 16 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/rwd0e:     164304 sectors in 163 cylinders of 16 tracks, 63 sectors
        80.2MB in 11 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/rwd0g:     4194288 sectors in 4161 cylinders of 16 tracks, 63 sectors
        2048.0MB in 261 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/rwd0h:     8388576 sectors in 8322 cylinders of 16 tracks, 63 sectors
        4096.0MB in 521 cyl groups (16 c/g, 7.88MB/g, 1920 i/g)
/dev/wd0a on /mnt type ffs (rw, asynchronous, local, ctime=Thu Oct 10 21:
50:36 2 002)
/dev/wd0h on /mnt/home type ffs (rw, asynchronous, local, nodev, nosuid,
 ctime=Thu Oct 10 21:50:36 2002)
/dev/wd0d on /mnt/tmp type ffs (rw, asynchronous, local, nodev, nosuid,
ctime=Thu Oct 10 21:50:36 2002)
/dev/wd0g on /mnt/usr type ffs (rw, asynchronous, local, nodev, ctime=Th
u Oct 10 21:50:36 2002)
/dev/wd0e on /mnt/var type ffs (rw, asynchronous, local, nodev, nosuid,
ctime=Th u Oct 10 21:50:36 2002)
```

You may wonder why the installer again asks for mount points. This allows you to recover from any errors or omissions in the mount points specified during the creation of the disklabel. For instance, the installation process will automatically delete any duplicate mount points you enter during the configuration of the disklabel. The disklabel program will allow you to enter such duplicates, and thus they must be checked for after the disklabel program exits. The deleted duplicate mount points will result in partitions without mount points, that you must assign new mount points for if you wish to use the space.

Notice the "Are you really sure that you are ready to proceed?" question defaults to no, so you will have to deliberately tell it to proceed and format your partitions. If you chose no, you would simply be dropped into a shell and could start the install again by typing install, or just by rebooting again with your boot disk.

At this point all filesystems will formatted for you. This could take some time depending on the size of the partitions and the speed of the disk.

4.3.2 - Setting the System Hostname

Now you must set the system hostname. This value will be saved in the file `/etc/myname`, which is used during normal boots to set the hostname of the system. As of OpenBSD 3.2 the name stored in `/etc/myname` includes the fully qualified domain name of the system (FQDN). If you do not set the FQDN of the system, the default value of 'my.domain' will be used.

It is important to set this name now, because it will be used when the cryptographic keys for the system are generated during the first boot after installation. This generation takes place whether the network is configured or not.

```
Enter system hostname (short form, e.g. 'foo'): puffy
```

4.3.3 - Configuring your Network

Now it is time to configure your network. The network must be configured if you are planning on doing a ftp or nfs based install, considering it will be based upon the information you are about to enter. Here is a walk through of the network configuration section of the install process.

```
Configure the network? [y] Enter
Available interfaces are: fxp0.
Which one do you wish to initialize? (or 'done') [fxp0] Enter
Symbolic (host) name for fxp0? [puffy] Enter
The default media for fxp0 is
        media: Ethernet autoselect (100baseTX full-duplex)
Do you want to change the default media? [n] Enter
IP address for fxp0? (or 'dhcp') 199.185.137.55
Netmask? [255.255.255.0] Enter
Done - no available interfaces found.
DNS domain name? (e.g. 'bar.com') [my.domain] example.com
```

```
DNS nameserver? (IP address or 'none') [none] 199.185.137.1
Use the nameserver now? [y] Enter
Default route? (IP address, 'dhcp' or 'none') 199.185.137.128
add net default: gateway 199.185.137.128
Edit hosts with ed? [n] Enter
Do you want to do any manual network configuration? [n] Enter
```

In the above example, we use a static IP address. As indicated, you can use "`dhcp`" instead on most platforms (not Alpha), assuming your environment supports it. In the case of DHCP, most of the information will be grabbed from the remote DHCP server; you will be given a chance to confirm it. Here is a sample of the network configuration part of the install, this time done with DHCP:

```
Configure the network? [y] Enter
Available interfaces are: fxp0.
Which one do you wish to initialize? (or 'done') [fxp0] Enter
Symbolic (host) name for fxp0? [puffy] Enter
The default media for fxp0 is
        media: Ethernet autoselect (100baseTX full-duplex)
Do you want to change the default media? [n] Enter
IP address for fxp0? (or 'dhcp') dhcp
Issuing hostname-associated DHCP request for fxp0.
Internet Software Consortium DHCP Client 2.0pl5-OpenBSD
Listening on BPF/fxp0/00:08:c7:77:b4:6b
Sending on   BPF/fxp0/00:08:c7:77:b4:6b
Sending on   Socket/fallback/fallback-net
DHCPDISCOVER on fxp0 to 255.255.255.255 port 67 interval 1
DHCPOFFER from 199.185.137.128
DHCPREQUEST on fxp0 to 255.255.255.255 port 67
DHCPACK from 199.185.137.128
New Network Number: 199.185.137.0
New Broadcast Address: 199.185.137.255
bound to 199.185.137.55 -- renewal in 43200 seconds.
Done - no available interfaces found.
DNS domain name? (e.g. 'bar.com') [example.org] Enter
DNS nameserver? (IP address or 'none') [199.185.137.1] Enter
Use the nameserver now? [y] Enter
Default route? (IP address, 'dhcp' or 'none') [199.185.137.128] Enter
add net default: gateway 199.185.137.128
Edit hosts with ed? [n] Enter
Do you want to do any manual network configuration? [n] Enter
```

NOTE: Only **one** interface can easily be configured using DHCP during an install. If you attempt to configure more than one interface using DHCP you will encounter errors. You have to manually configure the additional interfaces after the installation.

Now, we set the password for the root account:

```
Password for root account? (will not echo) pAssWOrd
Password for root account? (again) pAssWOrd
```

Use a secure password for the root account. You will creat other user accounts after the system is booted. From passwd(1):

> The new password should be at least six characters long and not purely alphabetic. Its total length must be less than _PASSWORD_LEN (currently 128 characters). A mixture of both lower and uppercase letters, numbers, and meta-characters is encouraged.

4.3.4 - Choosing Installation Media

After your network is set up, the install script will give you a chance to make manual adjustments to the configuration. Then the filesystems you created will be mounted and a root password set. This will get your local disks ready for the OpenBSD packages to be installed upon them.

Next, you will get a chance to choose your installation media. The options are listed below.

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

Sets can be located on a (m)ounted filesystem; a (c)drom, (d)isk or (t)ape
device; or a (f)tp, (n)fs or (h)ttp server.
Where are the install sets? c
Available CD-ROMs are: cd0.
```

In this example we are installing from CD-ROM. This will bring up a list of devices on your computer identified as a CD-ROM. Most people will only have one. If you need to, make sure you pick the device which you will use to install OpenBSD from.

NOTE: All possible sources for install sets are listed, but not all may be available on your system. e.g. (n)fs is shown but not all architectures allow NFS installations. If you choose a source that is not available, you will get an error message and be given the chance to choose another source for your installation sets.

```
Available CD-ROMs are: cd0.
Which one contains the install media? (or 'done') [cd0] Enter

Pathname to the sets?  (or 'done') [3.3/i386] Enter
```

Here, you are prompted for which directory the installation files are, which is `3.3/i386/` on the official CDROM.

4.3.5 - Choosing Filesets.

Now it's time to choose which packages you will be installing. You can get a description of these files in the next section. The files that the install program finds will be shown to you on the screen. Your job is just to specify which files you want. By default all the non-X packages are selected; however, some people may wish to limit this to the bare minimum required to run OpenBSD, which would be `base33.tgz`, `etc33.tgz` and `bsd`. Others will wish to install all packages. The example below is that of a full install.

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

        [X] bsd
        [ ] bsd.rd
        [X] base33.tgz
        [X] etc33.tgz
        [X] misc33.tgz
        [X] comp33.tgz
        [X] man33.tgz
        [X] game33.tgz
        [ ] xbase33.tgz
        [ ] xshare33.tgz
        [ ] xfont33.tgz
        [ ] xserv33.tgz

File Name? (or 'done') [bsd.rd] all
```

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

        [X] bsd
        [X] bsd.rd
        [X] base33.tgz
        [X] etc33.tgz
        [X] misc33.tgz
        [X] comp33.tgz
        [X] man33.tgz
        [X] game33.tgz
        [X] xbase33.tgz
        [X] xshare33.tgz
        [X] xfont33.tgz
        [X] xserv33.tgz
```

You can do all kinds of nifty things here -- `-x*` would remove all X components, if you changed your mind. In this case, we are going to load all the sets. While the system will run with fewer sets, the starting default is recommended. More details on selecting sets here.

Once you have successfully picked which packages you want, you will be prompted to make sure you want to extract these packages and they will then be installed. A progress bar will be shown that will keep you informed on how much time it will take. The times range greatly depending on what system it is you are installing OpenBSD on, the packages installed, and the speed of the source media. This part may from a few minutes to several hours.

```
File Name? (or 'done') [done] Enter
Ready to install sets? [y] Enter
Getting bsd ...
100% |*************************************************|  4472 KB    00:03
Getting bsd.rd ...
100% |*************************************************|  4190 KB    00:02
Getting base33.tgz ...
100% |*************************************************| 30255 KB    00:21
Getting etc33.tgz ...
100% |*************************************************|  1469 KB    00:01
Getting misc33.tgz ...
100% |*************************************************|  1828 KB    00:01
Getting comp33.tgz ...
100% |*************************************************| 16207 KB    00:13
Getting man33.tgz ...
100% |*************************************************|  5921 KB    00:04
Getting game33.tgz ...
100% |*************************************************|  2545 KB    00:01
Getting xbase33.tgz ...
100% |*************************************************|  9073 KB    00:06
Getting xshare33.tgz ...
100% |*************************************************|  1574 KB    00:02
Getting xfont33.tgz ...
100% |*************************************************| 30666 KB    00:21
Getting xserv33.tgz ...
100% |*************************************************| 14948 KB    00:11

Sets can be located on a (m)ounted filesystem; a (c)drom, (d)isk or (t)ape
device; or a (f)tp, (n)fs or (h)ttp server.
Where are the install sets? (or 'done')
```

At this point, you can pull additional files from other sources if desired, or hit '`done`' if you have installed all the file sets you need.

If your system has a small amount of RAM (less than 20M on i386), do not answer that prompt quite yet!

4.3.6 - Special steps for machines with little RAM

As OpenBSD has grown, the minimum RAM requirement has grown, and is likely to continue to increase. The next step in the install process will require more than 16M RAM, and will fail on older machines with less than 20M RAM (again, these are i386 numbers -- some other platforms will require far less RAM, though this trick can be used with them as well, if they are running near the lower-limits).

During the install process, there is normally no swap; the real RAM is all you have. The 'MAKEDEV' step that follows will require more than the rest of the install required. As a system with small amounts of RAM can still be very usable for many applications, working around this limitation is a quite useful trick.

The solution is to activate swap now. The swap partition has been created, the files are installed to the hard disk, the only trick here is to manually invoke it. So, do not just hit '`done`' at the above prompt, but rather, hit '!', which will bring up a shell, and launch swapon(8) from the mounted hard drive:

```
Where are the install sets? (or 'done') !
Type 'exit' to return to install.
# /mnt/sbin/swapon /dev/wd0b
# exit
Where are the install sets? (or 'done') done
```

You can now resume the normal installation.

4.3.7 - Finishing up

You will now be asked if you plan to run X on this system. If you answer 'Y', `/etc/sysctl.conf` will be modified to include the line `machdep.allowaperture=1` or `machdep.allowaperture=2`, depending on your platform.

Your last task is to enter the time zone. Depending on where your machine lives, there are may be several equally valid answers for the question. In the example that follows, we used `US/Eastern`, but could also have used `EST5EDT` or `US/Michigan` and had the same result. Hitting `?` at the prompts will guide you through your choices.

```
Extract more sets? [n] Enter
Do you expect to run the X Window System? [y] y
Saving configuration files......done.
Generating initial host.random file ......done.
What timezone are you in? ('?' for list) [US/Pacific] ?
Africa/      Chile/       GB-Eire      Israel       NZ-CHAT      Turkey
America/     Cuba         GMT          Jamaica      Navajo       UCT
Antarctica/  EET          GMT+0        Japan        PRC          US/
Arctic/      EST          GMT-0        Kwajalein    PST8PDT      UTC
Asia/        EST5EDT      GMT0         Libya        Pacific/     Universal
Atlantic/    Egypt        Greenwich    MET          Poland       W-SU
Australia/   Eire         HST          MST          Portugal     WET
Brazil/      Etc/         Hongkong     MST7MDT      ROC          Zulu
CET          Europe/      Iceland      Mexico/      ROK          posix/
CST6CDT      Factory      Indian/      Mideast/     Singapore    posixrules
Canada/      GB           Iran         NZ           SystemV/     right/
What timezone are you in? ('?' for list) [US/Pacific] US
What sub-timezone of 'US' are you in? ('?' for list) ?
Alaska          Central         Hawaii          Mountain        Samoa
Aleutian        East-Indiana    Indiana-Starke  Pacific
Arizona         Eastern         Michigan        Pacific-New
Select a sub-timezone of 'US' ('?' for list): Eastern
Setting local timezone to 'US/Eastern'...done.
```

If you are concerned about very precise time, you may wish to read this.

The last steps are for the system to create the `/dev` directory (which may take a while on some systems, especially if you had to use the Small RAM trick above), and install the boot blocks.

```
Making all device nodes...done.
Installing boot block...
boot: /mnt/boot
proto: /usr/mdec/biosboot
device: /dev/rwd0c
/usr/mdec/biosboot: entry point 0
proto bootblock size 512
room for 12 filesystem blocks at 0x16f
Will load 7 blocks of size 8192 each.
Using disk geometry of 63 sectors and 240 heads.
 0:  9 @(203 150 55) (3078864-3078872)
 1: 63 @(203 151 1) (3078873-3078935)
 2: 24 @(203 152 1) (3078936-3078959)
 3: 16 @(203 8 47) (3069910-3069925)
/mnt/boot: 4 entries total
using MBR partition 1: type 166 (0xa6) offset 3069360 (0x2ed5b0)
...done.

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

The operating system has halted.
Please press any key to reboot.
```

OpenBSD is now installed on your system and ready for its first boot, but before you do...

Before you reboot

At this point, your system is installed and ready to be rebooted and configured for service. Before doing this, however, it would be wise to check out the Errata page to see if there are any bugs that would immediately impact you.

After you reboot

One of your first things to read after you install your system is afterboot(8).

You may also find the following links useful:

- Adding users in OpenBSD
- Initial Network Setup
- Man Pages of popular/useful commands
- OpenBSD man pages on the Web
- The OpenBSD Ports and Packages system for installing software, as well as here and here

One last thing...

The OpenBSD developers ask you to Send in a copy of your dmesg. This is really appreciated by the developers, and ultimately, all users.

4.3.8 - Other Information Resources

Additional documents already exist for those of you who might have special needs or interests. You can retrieve these from any of the mirror ftp sites.

- INSTALL.i386 - Comprehensive installation document. Similar documents exist for all platforms.
- INSTALL.linux - Installing OpenBSD along with Linux.
- INSTALL.mbr - Explaining the Master Boot Record.

- INSTALL.pt - Explaining Partition Tables.
- INSTALL.dbr - DOS Floppy Disk Boot Sector.
- INSTALL.chs - Explaining CHS Translation.
- INSTALL.ata - ATA/ATA-1/ATA-2/IDE/EIDE/etc FAQ
- INSTALL.os2br - The os2 Boot Sector.

4.4 - What files are needed for Installation?

The complete OpenBSD installation is broken up into a number of separate *file sets*. Not every application requires every file set. Here is an overview of each:

- ***bsd*** - This is the Kernel. **Required**
- ***bsd.rd*** - RAM disk kernel
- ***base33.tgz*** - Contains the base OpenBSD system **Required**
- ***etc33.tgz*** - Contains all the files in /etc **Required**
- ***comp33.tgz*** - Contains the compiler and its tools, libs. **Recommended**
- ***man33.tgz*** - Contains man pages **Recommended**
- ***misc33.tgz*** - Contains misc info, setup documentation
- ***game33.tgz*** - Contains the games for OpenBSD
- ***xbase33.tgz*** - Contains the base install for X11
- ***xfont33.tgz*** - Contains X11's font server and fonts
- ***xserv33.tgz*** - Contains X11's X servers
- ***xshare33.tgz*** - Contains manpages, locale settings, includes, etc for X

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

The following are minimum suggested filesystem 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.

- These are minimum 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, or put them on separate partitions.
- For a multiuser system which may generate lots of logs, you will still want to make your `/var` partition significantly larger (`/var/log`).
- If you plan to rebuild the kernel or system from source, you will want to make the `/usr` partition significantly larger, **at least** 800M-1G larger than indicated below.

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	80M	250M	25M	140M
hp300	80M	250M	25M	140M
hppa	100M	200M	25M	120M
i386	60M	250M	25M	140M
mac68k	80M	250M	25M	100M
macppc	80M	250M	25M	140M
mvme68k	80M	250M	25M	100M
sparc	80M	250M	25M	120M
sparc64	80M	250M	25M	100M

```
vax             100M     200M     25M      120M
```

In addition, it is recommended that a `/tmp` partition be used. The `/tmp` partition is used in the compiling of ports, among other things, so how big you make it depends on what you do with it. 50M may be plenty for most people, but some large applications may require 100M or more of `/tmp` space.

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.

Swap space is used to store system core dumps on in the event of a crash(8). If this is a consideration for you, your swap space should be slightly larger than the amount of main memory you are likely to ever have in the system. Note that upon reboot, savecore(8) will attempt to save the contents of the swap partition to a file in `/var/crash` so again, if this is a priority for you, your `/var` partition must have enough *free space* to hold these dump files.

There are five 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. This is now done by the install process, in fact, if you use the above described partitions.
- **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.
- **Size**: Many platforms have limits on the area of a disk where the boot ROM can load the kernel from. In some cases, this limit may be very small (504M for an older 486), in other cases, a much larger limit (8G on new i386 systems). As the kernel can end up anywhere in the root partition, the entire root partition should be within this area. For more details, see this section. A good guideline might be to keep your / partition completely below 2G, unless you know your platform (and particular machine!) can handle more (or less!) than that.

Some additional thoughts on partitioning:

- For your first attempt at an experimentation system, one big `/` partition and swap may be easiest until you know how much space you need. By doing this you will be sacrificing some of the default security features of OpenBSD that require separate filesystems for `/, /tmp, /var, /usr` and `/home`.
- A system exposed to the Internet or other hostile forces should have a separate `/var` (and maybe even a separate `/var/log`) for logging.
- A `/home` partition can be nice. New version of the OS? Wipe and reload everything else, leave your `/home` partition untouched. Remember to save a copy of your configuration files, though!
- A separate partition for anything which may accumulate a large quantity of files that may need to be deleted can be faster to reformat and recreate than to delete. See the upgrade-minifaq for an example (`/usr/obj`).
- If you wish to rebuild your system from source for any reason, the source will be in `/usr/src`. If you don't make a separate partition for `/usr/src`, make sure `/usr` has sufficient space.
- A commonly forgotten fact: you do **not** have to allocate all space on a drive when you set the system up! Since you will now find it a challenge to buy a new drive smaller than 20G, it can make sense to leave a chunk of your drive unallocated. If you outgrow a partition, you can allocate a new partition from your unused space, duplicate your existing partition to the new partition, change /etc/fstab to point to the new partition, remount, you now have more space.
- If you make your partitions too close to the minimum size required, you will probably regret it later, when it is time to upgrade your system.
- If you permit users to write to `/var/www` (i.e., personal web pages), you might wish to put it on a separate partition, so you can use quotas to restrict the space they use, and if they fill the partition, no other parts of your system will be impacted.

4.6 - Multibooting OpenBSD (i386)

Multibooting is having several operating systems on one computer, and some means of selecting the which OS is to boot. It is *not* a trivial task! If you don't understand what you are doing, you may end up deleting large amounts of data from your computer. New OpenBSD users are *highly* encouraged to start with a blank hard drive on a dedicated machine, and then practice your desired configuration on a non-production system before attempting a multiboot configuration on a production machine. FAQ 14 has more information about the OpenBSD boot process.

When multibooting, the requirements of all operating systems must be met by your configuration. People often ask if there is a way around the 8G boot limit of OpenBSD. While there are some programs that claim to get around various limits of various operating systems, none of them are known to do this with current versions of OpenBSD.

Here are several options to multibooting:

Setting active partitions

This is probably the most overlooked, and yet, sometimes the best solution for multibooting. Simply set the active partition in whatever OS you are currently using to be the one you want to boot by default when you next boot. Virtually every OS offers a program to do this; OpenBSD's is fdisk(8), similar named programs are in Windows 9x and DOS, and many other operating systems. This can be highly desirable for OSs or systems which take a long time to shut down and reboot -- you can set it and start the reboot process, then walk away, grab a cup of coffee, and come back to the system booted the way you want it -- no waiting for the Magic Moment to select the next OS.

Boot floppy

If you have a system that is used to boot OpenBSD infrequently (or don't wish other users of the computer to note anything has changed), consider using a boot floppy. Simply use one of the standard OpenBSD install floppies, and create a `/etc/boot.conf` file (yes, you will also have to create an `/etc` directory on the floppy) with the contents:

```
boot hd0a:/bsd
```

to cause the system to boot from hard drive 0, OpenBSD partition 'a', kernel file `/bsd`. Note you can also boot from other drives with a line like: "`boot hd2a:/bsd`" to boot off the third hard drive on your system. To boot from OpenBSD, slip your floppy in, reboot. To boot from the other OS, eject the floppy, reboot.

In this case, the boot(8) program is loaded from the floppy, looks for and reads `/etc/boot.conf`. The "`boot hd0a:/bsd`" line instructs boot(8) where to load the kernel from -- in this case, the first HD the BIOS sees. Keep in mind, only a small file (`/boot`) is loaded from the floppy -- the system loads the entire kernel off the hard disk, so this only adds about five seconds to the boot process.

Windows NT/2000/XP NTLDR

To multiboot OpenBSD and Windows NT/2000/XP, you can use NTLDR, the boot loader that NT uses. To multi-boot with NT, you need a copy of your OpenBSD Partition Boot Record (PBR). After running installboot, you can copy it to a file using dd(1):

```
# dd if=/dev/rsd0a 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 NTLDR at the NTLDR Hacking Guide.

On Windows XP you can also edit the boot information using the GUI; see the XP Boot.ini HOWTO.

Note: The Windows NT boot loader is only capable of booting OSs from the primary hard drive. You can not use it to load OpenBSD from the second drive on a system.

Other boot loaders

Some other bootloaders OpenBSD users have used successfully include GAG, OSBS, The Ranish Partition Manager and GRUB.

OpenBSD and Linux (i386)

Please refer to INSTALL.linux, which gives in depth instructions on getting OpenBSD working with Linux.

4.7 - 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 configured!) 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.
```

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 dmesgs. Custom kernels that have device drivers removed are not helpful.

4.8 - Adding a file set after install

"Oh no! I forgot to add a file set when I did the install!"

Sometimes, you realize you really DID need `comp33.tgz` (or any other system component) after all, but you didn't realize this at the time you installed your system. Good news: There are two easy ways to add file sets after the initial install:

Using the Upgrade process

Simply boot your install media (CD-ROM or Floppy), and choose Upgrade (rather than Install). When you get to the lists of file sets to install, choose the sets you neglected to install first time around, select your source, and let it install them for you.

Using tar(1)

The install file sets are simply compressed tar files, and you can expand them manually from the root of the filesystem:

```
# cd /
# tar xzvpf comp33.tgz
```

Do NOT forget the `'p'` option in the above command in order to restore the file permissions properly!

One common mistake is to think you can use pkg_add(1) to add a missing file sets. This does not work. pgk_add(1) is for package files, not generic tar files like the install sets.

4.9 - What is 'bsd.rd'?

`bsd.rd` is a "RAM Disk" kernel. This file can be very useful; many developers are careful to keep it on the root of their system at all times.

Calling it a "RAM Disk kernel" describes the root filesystem of the kernel -- rather than being a physical drive, the utilities available after the boot of `bsd.rd` are stored in the kernel, and are run from a RAM-based filesystem. `bsd.rd` also includes a healthy set of utilities to allow you to do system maintenance and installation.

On some platforms, `bsd.rd` is actually the preferred installation technique -- you place this kernel on an existing filesystem, boot it, and run the install from it. On most platforms, if you have a running older version of OpenBSD, you can FTP a new version of `bsd.rd`, reboot from it, and install a new version of OpenBSD without using any removable media at all.

Here is an example of booting `bsd.rd` on an i386 system:

```
  Using Drive: 0 Partition: 3
  reading boot.....
  probing pc0 com0 com1 apm mem[639k 255M a20=on]
  disk fd0 hd0
  >> OpenBSD/i386 BOOT 1.29
  boot> boot hd0a:/bsd.rd
. . . normal boot to install . . .
```

As indicated, you will be brought to the install program, but you can also drop to the shell to do maintenance on your system.

The general rule on booting `bsd.rd` is to change your boot kernel from `/bsd` to `bsd.rd` through whatever means used on your platform.

4.10 - Common Installation Problems

4.10.1 - Install hangs during MAKEDEV

Usually, this is caused by a lack of physical RAM in the system. See here.

4.10.2 - My Compaq only recognizes 16M RAM

Some Compaq systems have an issue where the full system RAM is not detected by the OpenBSD second stage boot loader properly, and only 16M may be detected and used by OpenBSD. This can be corrected either by creating/editing `/etc/boot.conf` file, or by entering commands at the "`boot>`" prompt before OpenBSD loads. If you had a machine with 64M RAM, but OpenBSD was only detecting the first 16M, the command you would use would be:

```
machine mem +0x3000000@0x1000000
```

to add 48M (`0x3000000`) after the first 16M (`0x1000000`). Typically, if you had a machine with this problem, you would enter the above command first at the install floppy/CDROM's `boot>` prompt, load the system, reboot, and create a `/etc/boot.conf` file with the above line in it so all future bootings will recognize all available RAM.

It has also been reported that a ROM update will fix this on *some* systems.

4.10.3 - My i386 won't boot after install

Your install seemed to go fine, but on first boot, you see no sign of OpenBSD attempting to boot. There are a few common reasons for this problem:

- **No partition was flagged active in fdisk(8).** To fix this, reboot the machine using the boot floppy or media, and "flag" a partition as "active" (bootable). See here and here
- **No valid boot loader was ever put on the disk.** If you answer "Y" to the "Use entire disk for OpenBSD?" question during the install, or use the "`reinit`" option of fdisk(8), the OpenBSD boot record is installed on the Master Boot Record of the disk; otherwise, the existing master boot code is untouched. This will be a problem if no other boot record existed. The solution is to boot the install media again, drop to the shell and invoke fdisk(8) and use the "`update`" option.
- **In some rare occasions, something may go wrong with the second stage boot loader install.** Reinstalling the second stage boot loader is discussed here.

4.10.4 - My (older, slower) machine booted, but hung at the ssh-keygen steps

It is very likely your machine is running fine, just taking a while to do the ssh key generation process. A SparcStation2 or a Macintosh Quadra may take 45 minutes or more to complete the three ssh-keygen(1) steps, some machines will take even longer. Just let it finish; it is only done once per install.

4.10.5 - I got the message "ftplist: No such file" when doing an install

When doing an FTP install of a snapshot during the *-beta* stage of the OpenBSD development cycle, you may see this:

```
Do you want to see a list of potential FTP servers?  [y] ENTER
grep: /tmp/ftplist: No such file or directory
Server IP address or hostname?
```

This is normal and expected behavior during this pre-release part of the cycle. The install program looks for the FTP list on the primary FTP server in a directory that won't be available until the release date, so you get the above message.

Simply use the FTP mirror list to find your favorite FTP mirror, and manually enter its name when prompted.

Note: You should not see this if you are installing *-release* or from CDROM.

4.11 - Customizing the Install Process

`siteXX.tgz` file

The OpenBSD install/upgrade scripts allow the selection of a user-created set called "`siteXX.tgz`", where XX is the release version (e.g. 33). The `siteXX.tgz` file set is, like the other file sets, a gzip(1) compressed tar(1) archive rooted in '/' and is un-tarred like the other sets with the options `xzpf`. This set will be installed last, after all other file sets.

This file set allows the user to add to and/or override the files installed in the 'normal' sets and thus customize the installation or upgrade.

Some example uses of a `siteXX.tgz` file:

- Create a siteXX.tgz file that contains all the file changes you made since first installing OpenBSD. Then, if you have to re-create the system you simply select siteXX.tgz during the re-install and all of your changes are replicated on the new system.
- Create a series of machine specific directories that each contain a siteXX.tgz file that contains files specific to those machine types. Installation of machines (e.g. boxes with different graphics cards) of a particular category can be completed by selecting the appropriate siteXX.tgz file.
- Put the files you routinely customize in a same or similar way in a siteXX.tgz file -- `/etc/skel` files, `/etc/pf.conf`, `/var/www/conf/httpd.conf`, `/etc/rc.conf`, etc.

`install.site/upgrade.site` scripts

As the last step in the install/upgrade process, the scripts look in the root directory of the newly installed/upgraded system for `install.site` or `upgrade.site`, as appropriate to the current process, and runs this script in an environment chrooted to the installed/upgraded system's root. Remember, the upgrade is done from a booted file system, so your target file system is actually mounted on `/mnt`. However, your script can be written as if it it is running in the "normal" root of your file system. Since this script is run after all the files are installed, you have almost full functionality of your system (though, in single user mode) when your script runs.

Note that the `install.site` script would have to be in a `siteXX.tgz file`, while the `upgrade.site` script could be put in the root directory before the upgrade, or could be put in a `siteXX.tgz` file.

The scripts can be used to do anything possible in a script.

- Remove files that are installed/upgraded that you don't want present on the system.
- Remove/upgrade/install the packages you want on the installed system.
- Do an immediate backup/archive of the new system before you expose it to the rest of the world.

The combination of `siteXX.tgz` and `install.site/upgrade.site` files is intended to give the user broad customization capabilities without having to build their own custom install sets.

4.12 - How can I load a number of similar systems?

Here are some tools you can use when you have to deploy a number of similar OpenBSD systems.

`siteXX.tgz` and `install/upgrade.site` files

See the above article.

Restore from Dump

On most platforms, the boot media includes the restore(8) program, which can be used to restore a backup made by dump(8). Thus, you could boot from a floppy, CD, or bsd.rd file, then fdisk, disklabel, and restore the desired configuration from tape or other media, and install the boot blocks. More details here.

Disk Imaging

Unfortunately, there are no known disk imaging packages which are FFS-aware and can make an image containing only the active file space. Most of the major disk imaging solutions will treat an OpenBSD partition as a "generic" partition, and can make an image of the whole disk. This often accomplishes your goal, but usually with huge amounts of wasted space -- an empty, 10G `/home` partition will require 10G of space in the image, even if there isn't a single file in it. While you can typically install a drive image to a larger drive, you would not be able to directly use the extra space, and you would not be able to install an image to a smaller drive.

If this is an acceptable situation, you may find the dd command will do what you need, allowing you to copy one disk to another, sector-for-sector. This would provide the same functionality as commercial programs without the cost.

4.13 - How can I get a dmesg(8) to report an install problem?

When reporting a problem, it is critical to include the complete system dmesg(8). However, often when you need to do this, it is because the system is working improperly or won't install so you may not have disk, network, or other resources you need to get the dmesg to the appropriate mail list. There are other ways, however:

- **Floppy disk:** The boot disks and CDROM has enough tools to let you record your dmesg to an MSDOS floppy disk for reading on another machine. Place an MSDOS formatted floppy in your disk drive and execute the following commands:

```
mount -t msdos /dev/fd0a /mnt
dmesg >/mnt/dmesg.txt
umount /mnt
```

If you have another OpenBSD system, you can also write it to an OpenBSD compatible floppy -- often, the boot floppy has enough room on it to hold the dmesg. In that case, leave off the "`-t msdos`" above.

- **Serial Console:** See this article on setting up the serial console, then capture the output to a file.
- **FTP:** Under some circumstances, you may be able to use the ftp(1) client on the boot disk or CDROM to send the dmesg to a local FTP server, where you can retrieve it later.

[FAQ Index] [To Section 3 - Obtaining OpenBSD] [To Section 5 - Building the System from Source]


www@openbsd.org

$OpenBSD: faq4.html,v 1.138 2003/05/01 01:47:41 nick Exp $

5 - Neuerzeugen des Systems aus den Quelltexten

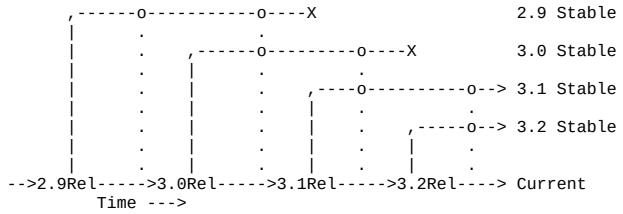
Inhaltsverzeichnis

5.1 - OpenBSD's Flavors

Es gibt drei "flavors" von OpenBSD:

- **-release:** Die Version von OpenBSD, die alle 6 Monate auf CD ausgeliefert wird.
- **-stable:** Release und zusätzliche Patches, die für Sicherheit und Zuverlässigkeit als notwendig erachtet werden.
- **-current:** Die jetzt-im-Moment-aktuelle Version von OpenBSD, die sich in die nächste release-Version verwandelt.

Grafisch sieht die Entwicklung dieser flavors ungefähr so aus:



Der code tree wird alle sechs Monate in *-current*, *-release* und *-stable* aufgeteilt -- *-release* wird zu einem festen Punkt (einem "Tag") in der Geschichte des source tree - der niemals geändert, und genau der ist auch auf den CDs und FTP Servern.

-Current dagegen ist die Baustelle, an der die aktive Arbeit gemacht wird, und die dann der nächste *-release* von OpenBSD wird.

Der *-stable* branch (auch bekannt als der "Patch branch") basiert auf *-release*, und ist, wie man sehen kann, ein "branch" aus dem Entwicklungspfad von OpenBSD. Wenn wichtige Korrekturen und Fixes an *-current* vorgenommen werden, werden diese in die *-stable* branches "zurückportiert". In der obigen Illustration sind die vertikalen gepunkteten Linien Bug Fixes, die in die *-stable* branches eingefügt werden. Du wirst auch erkennen, dass der Lebenszyklus des *2.9-stable* branch mit Erscheinen des *3.1-release* beendet war und der *3.0-stable* branch mit Erscheinen von *3.2-release* beendet war -- alte Versionen werden typischerweise noch zwei "releases" lang weitergepflegt. Es braucht nunmal Ressourcen und Zeit, ältere Versionen zu pflegen, und obwohl wir das gerne tun würden, konzentrieren wir uns doch lieber auf neue Features. Der *-stable* branch kann vom Design her sehr leicht aus dem *-release* branch der selben Version erzeugt werden (z.B. von *3.2-release* zu *3.2-stable*).

Der *-stable* branch ist *-release* plus patches, die man auf der errata Seite findet, und ein paar einfache fixes, die keinen errata-Entrag verdienen. Normalerweise ist die Bedienung und der Betrieb von *-stable* genauso wie der des *-release*, auf dem er basiert. Wenn die man pages geändert werden müssen, wird es wahrscheinlich nicht in *-stable* eingefügt werden. Mit anderen Worten, Unterstützung für neue Hardware wird NICHT in *-stable* eingefügt, und neue Features werden nur dann eingefügt, wenn das als sehr wichtig erachtet wird.

Warnung: *-current* ist ein sich bewegendes Ziel. Es ändert sich fast minütlich, und kann sich genauso gut mehrere Male in der Zeit ändern, die man braucht, um dem Source Code zu holen. Wie vorher schon gesagt, gibt es keinerlei Garantie, dass das System kompiliert oder gar funktioniert (natürlich hoffen wir das immer). Es ist absolut möglich und nicht ungewöhnlich, dass man sich den *-current* Source holt, und er nicht kompiliert werden kann, während es fünf Minuten später wieder klappt. **Wenn du damit nicht umgehen kannst, lass deine Finger von *-current*.**

Die meisten User sollten also einfach *-stable* oder *-release* benutzen. Wo das jetzt klar ist, sollte man aber auch wissen, dass viele Leute *-current* auf Produktionssystemen fahren, und das ist insofern wichtig, weil es hilft Bugs zu finden und neue Features zu testen. Wenn du aber nicht weisst, wie man sauber Probleme beschreibt, diagnostiziert oder damit umgeht, rede dir (oder jemand anderem) besser nicht ein, du würdest "dem Projekt helfen", indem du *-current* benutzt. "Es funktioniert nicht!" ist kein nützlicher bug report. "The recent changes to the pciide driver broke compatibility with my Slugchip-based IDE interface, dmesg of working and broken systems follow..." könnte dagegen durchaus ein sinnvoller und nützlicher Bericht sein.

Es mag Zeiten geben, wo auch ein "normaler" User gerne "auf des Messers Schneide" leben will und *-current* benutzt. Der häufigste Grund dafür ist, dass er ein Gerät hat, was nicht von *-release* unterstützt wird (und daher natürlich auch nicht von *-stable*), oder er möchte ein neues Feature aus *-current* benutzen. In diesem Fall hat er die Wahl zwischen *-current* oder dem Nichtbenutzen seines Gerätes, und dann ist eben das Benutzen von *-current* oftmals weniger schlimm. Man sollte dann aber auf keinen Fall ein Händchenhalten der Entwickler erwarten, falls es zu Problemen kommt.

Snapshots

Zwischen den formalen Veröffentlichungen neuer Versionen von OpenBSD werden *snapshots* über die FTP Sites verfügbar gemacht. Wie der Name schon sagt, sind das Versionen von Code, der gerade zu eben diesem Zeitpunkt aktuell war, als der Übersetzer des snapshot sich eine Kopie des Source Code für eben diese Plattform gemacht hat. Denke bitte daran, dass das auf einigen Plattformen auch einige TAGE sein können, bis der Snapshot dann auch kompiliert und zur Verfügung gestellt wird. Es gibt auch keinerlei Garantien, dass die Snapshots funktionsfähig sind, oder auch nur installationsfähig. Oftmals werden Snapshots dann erzeugt, wenn es eine Änderung gibt, die getestet werden muss. Bei einigen Plattformen werden Snapshots auf fast täglicher Basis erstellt, andere sind weniger regelmässig. Wenn du unbedingt *-current* benutzen willst, ist ein aktueller Snapshot oft das einzige, was du dazu brauchst, und das Upgraden auf den aktuellen Snapshot ist auch ein wunderbarer Ausgangspunkt, bevor du versuchst *-current* zu erzeugen.

Manchmal wird gefragt, ob es einen Weg gibt, genau den Source-Code zu bekommen, der zur Erzeugung eines Snapshot eingesetzt wurde. Die Antwort ist "Nein". Erstens ergibt sich daraus kein besonderer Vorteil. Zweitens werden Snapshots nach Bedarf erzeugt, wenn die Zeit es erlaubt oder es genügend Ressourcen gibt. Auf schnellen Plattformen könnte man z.B. mehrere Snapshots an einem Tag erzeugen. Auf den langsameren kann das schon mal eine ganze Woche dauern. Und das Platzieren eines "Tag" für jeden Snapshot in den Source Code ist nun wirklich extrem unpraktisch.

Die Dinge in Balance halten

Es ist wichtig zu verstehen, dass OpenBSD ein Betriebssystem ist, und als ganzes gesehen werden muss, nicht ein Kernel mit einem Schwanz an Applikationen, die drangehängt sind. Du musst sicherstellen, dass dein Kernel, dein "Userland" (die unterstützenden Werkzeuge und Dateien) und dein `ports` tree alle synchronisiert sind, oder es werden unangenehme Dinge passieren. Oder anders ausgedrückt (da es immer wieder Leute gibt, die das versuchen), kannst du keine brandneuen `ports` auf einem System laufen lassen, das einen Monat alt ist, oder einen Kernel aus *-current* neu erzeugen und dann erwarten, dass er mit einem *release*-Userland zusammenarbeitet. Ja, das heisst, dass du dein System auf aktuellem Stand halten musst, wenn du ein neues Programm laufen lassen willst, das z.B. erst heute in den ports tree eingefügt wurde. Tut uns erneut leid, aber OpenBSD hat nunmal nur sehr begrenzte Ressourcen, so dass wir sowas nicht ändern können.

Man sollte verstehen, dass der Update-Prozess, wenn man ein Upgrade mittels Source durchführt, **nur in eine Richtung unterstützt wird: Von älter zu neuer**, und von *-stable* zu *-current*. Du kannst kein *3.2-current* (oder einen Snapshot) laufen lassen, und dich dann entscheiden, dass dir das zu gefährlich ist, und einfach nach *3.2-stable* zurückgehen. Du bist auf dich allein gestellt, wenn du einen anderen Weg wählst als den normalen und unterstützen, nämlich dein System von Grund auf neu zu installieren, erwarte bitte keinerlei Hilfeleistung vom OpenBSD Entwicklerteam.

Ja, das soll wirklich heissen, dass du lang und breit darüber nachdenken sollst, bevor du dich an *-current* wagst.

5.2 - Warum brauche ich einen selber kompilierten Kernel?

Es gibt dafür mehrere Gründe, obwohl das Kompilieren eines eigenen Kernels eher für erfahrenere Benutzer geeignet ist, die bereits ein gutes Allgemeinverständnis vom System besitzen.

- Dein Computer hat sehr wenig Hauptspeicher, den du so gut wie möglich ausnutzen willst, indem du nicht benötigte Gerätetreiber entfernst.
- Du möchtest Standardoptionen entfernen oder Optionen hinzufügen, die nicht standardmäßig aktiviert sind.
- Du möchtest experimentelle Optionen hinzufügen.

In den meisten Fällen wirst du keinen eigenen Kernel kompilieren müssen. Ein GENERIC Kernel wird normalerweise alles beinhalten, das du brauchst. Es gibt sogar mehrere Gründe, warum du keinen eigenen Kernel kreieren willst. Der Hauptgrund ist, daß man zwar logisch aussehende Änderungen an der Kernelkonfiguration auf einfache Weise durchführen kann, aber dein Kernel funktioniert dann nicht. Dies soll ein Warnsignal sein. Wenn etwas nicht zu funktionieren scheint, dann probiere bitte vorher einen GENERIC Kernel zu starten, bevor du einen Fehlerbericht einschickst. Die Entwickler werden im Normalfall jeden Fehlerbericht ignorieren, der nicht mit einem GENRIC Kernel erzeugt und getestet wurde, es sei denn das Problem kann ebenfalls mit einem GENERIC Kernel reproduziert werden. Nimm das als Warnung.

5.3 - Kernelkonfigurationsoptionen

Kernelkonfigurationsoptionen bieten dir die Möglichkeit, zusätzliche Unterstützung zu deinem Kernel hinzuzufügen. Dies erlaubt dir, nur die von dir gewünschten Geräte zu unterstützen. Es gibt eine Vielzahl von Möglichkeiten, deinen Kernel auf deine Wünsche abzustimmen. Hier werden wir nur auf einige der am häufigsten benutzten Möglichkeiten eingehen. Siehe options(4) für eine komplette Liste der Optionen. Es stehen auch

Beispielkonfigurationsdateien für deine Architektur bereit.

Nicht alle Kerneloptionen sind auf Kompatibilität mit allen anderen Optionen getestet worden. Daher solltest du keine Option in deinem Kernel setzen, wenn du nicht einen triftigen Grund dafür hast! Am meisten wird der GENERIC Kernel getestet. Dies ist normalerweise die Kombination der Optionen in /usr/src/sys/arch/<deine Arch>/conf/GENERIC und /usr/src/sys/conf/GENERIC.

- Alpha Kernel Konfgurationsdateien
- i386 Kernel Konfigurationsdateien
- macppc Kernel Konfigurationsdateien
- sparc Kernel Konfigurationsdateien
- sparc64 Kernel Konfigurationsdateien
- vax Kernel Konfigurationsdateien
- Andere Architekturen

Wenn du diese Dateien genauer betrachtest, dann wird dir eine Zeile wie diese auffallen:
include "../../../conf/GENERIC"
Dies bedeutet, daß ein Verweis auf eine andere Konfigurationsdatei vorhanden ist. Diese Datei beinhaltet architekturunabhängige Optionen. Wenn du also deinen eigenen Kernel konfigurieren willst, dann mußt du auch /sys/conf/GENERIC beachten. Dort **sind** Optionen enthalten, die **benötigt** werden.

Alle nachfolgend aufgelisteten Optionen sollten in deiner Kernelkonfigurationsdatei im Format von:
option OPTION
angeführt sein. Um die Option debug im Kernel zu plazieren, füge eine Zeile wie diese ein:
option DEBUG
Die Optionen im OpenBSD Kernel werden in Compilerpräprozessoroptionen übersetzt, daher würde eine Option wie DEBUG den Quelltext mit der Option -DDEBUG kompilieren, was einem #define DEBUG im Kernel entspricht.

OpenBSD hat viele Kompatibilitätsmöglichkeiten, die dir erlauben, Binärdateien von anderen Betriebsystemen auszuführen. Nicht alle Optionen sind auf jeder Architektur vorhanden, daher lies die entsprechende Manualseite, ob die Architektur unterstützt wird.

- COMPAT_SVR4(8) - Kompatibilität mit SVR4 Binärdateien.
- COMPAT_BSDOS(8) - Kompatibilität mit BSD/OS Binärdateien.
- COMPAT_LINUX(8) - Kompatibilität mit Linux Binärdateien.
- COMPAT_SUNOS(8) - Kompatibilität mit SunOS Binärdateien.
- COMPAT_ULTRIX(8) - Kompatibilität mit Ultrix Binärdateien.
- COMPAT_FREEBSD(8) - Kompatibilität mit FreeBSD Binärdateien.
- COMPAT_HPUX(8) - Kompatibilität mit HP-UX Binärdateien. Nur auf einigen m68k Architekturen verfügbar.
- COMPAT_IBCS2(8) - Kompatibilität mit ibcs2 Binärdateien.
- COMPAT_OSF1(8) - Digital Unix Binärdateien laufen lassen. Nur auf der Alpha Plattform verfügbar.
- COMPAT_43 - Kompatibilität mit 4.3BSD. Von dessen Gebrauch wird zwar abgeraten, aber wird für einige Applikationen benötigt.
- COMPAT_11 - Kompatibilität mit NetBSD 1.1.
- COMPAT_NOMID - Kompatibilität mit a.out executables ohne machine id.

Es ist immer hilfreich, Probleme mit dem Kernel debuggen zu können. Aber viele wollen diese Optionen nicht in ihren Kernel setzen, weil diese Optionen den Kernel sehr vergrößern. Sie sind aber extrem hilfreich im Falle eines Fehlers. Ein Entwickler wird die Quelle deiner Probleme viel schneller finden können. Hier eine Liste der debug Optionen:

- DDB(4) - Kernelinterner Debugger. Nicht auf allen Plattformen. Daher konsultiere die Manualseite.
- KGDB(7) - Kompiliert einen `remote kernel debugger` mittels `remote target` Option von gdb.
- makeoptions DEBUG="-g" - Erzeugt bsd.gdb gemeinsam mit bsd. Dies ist nützlich, um crash dumps mit gdb zu debuggen.
- DEBUG - Diverse debugging Optionen im Kernel, die im Quelltext definiert wurden.
- KTRACE - Fügt "hooks" für "system call tracing facility" ein. Dies erlaubt den Benutzern: ktrace(1).
- DIAGNOSTIC - Fügt Code im Kernel für interne Konsistenzüberprüfungen ein.
- GPROF - Fügt Code im Kernel für Kernelprofilierung mit kgmon(8) ein.
- makeoptions PROF="-pg" - Die -pg Option dient zur Unterstützung von "profiling" im Kernel. Die Option GPROF wird dafür benötigt.

Dateisystemoptionen.

- FFS - Berkeley Fast Filesystem. **ANMERKUNG:** Diese Option wird immer benötigt.
- EXT2FS - Second Extended File System, benötigt für das Lesen von Linux Partitionen.
- MFS - Memory File System, speichern von Dateien in swap - Speicher.
- NFS - Network File System, benötigt für NFS.
- CD9660 - Das iso9660 + rockridge Dateisystem; zum Lesen von CDs.
- MSDOSFS - Benötigt, um MS-DOS FAT Dateisysteme zu lesen; bietet Unterstützung für Windows 95 (lange Dateinamen + Groß-/Kleinschreibungerweiterungen).
- FDESC - Beinhaltet Code für ein Dateisystem, das auf /dev/fd gemountet werden kann.
- KERNFS - Beinhaltet Code für das Mounten eines speziellen Dateisystems (normalerweise auf /kern), in dem sich Dateien befinden, die spezielle Kernelvariablen und -parameter repräsentieren.
- NULLFS - Code für ein loopback Dateisystem. In mount_null(8) hat dazu weitere Informationen
- PROCFS - Beinhaltet Code für ein spezielles Dateisystem (normalerweise auf /proc)
- PORTAL - Beinhaltet das (experimentelle) portal Dateisystem. Dies erlaubt interessante Tricks wie TCP sockets durch das Öffnen von Dateien im Dateisystem zu öffnen.
- UMAPFS - Beinhaltet ein loopback Dateisystem, in dem Benutzer- und Gruppenids umgelegt werden dürfen -- nützlich, wenn man fremde Dateisysteme mit anderen uids und gids als das lokale System mounten will (zB.: NFS).
- UNION - Beinhaltet Code für das union Dateisystem, welches das Mounten von Verzeichnissen übereinander erlaubt, so daß beide Dateisysteme sichtbar bleiben. Dieser Code ist noch nicht sehr stabil.
- XFS - Fügt "hooks" für ein Dateisystem ein, das kompatibel mit dem AFS Dateisystem ist. Derzeit vom Arla/AFS Code benutzt.
- FFS_SOFTUPDATES - Erlaubt den Gebrauch von softupdates. Hier mehr dazu: Softupdates FAQ.
- NFSSERVER - Serverseitiger NFS Code im Kernel.
- NFSCLIENT - Klientseitiger NFS Code im Kernel.
- FIFO - Unterstützung für FIFOs. **Empfohlen**.
- NVNODE=integer - Wobei integer eine Ganzzahl ist, die der Größe des Caches bei name-to-inode Übersetzungsroutinen entspricht (dh. der namei() Cache, obwohl er auch von vielen anderen Funktionen im Kernelquelltext aufgerufen wird).
- EXT2FS_SYSTEM_FLAG - Diese Option ändert das Verhalten der APPEND- und der IMMUTABLE-option für Dateien in einem EXT2FS-Dateisystem. Siehe options(4) für weitere Details.
- QUOTA - Unterstützung für Quoten im Dateisystem. Hier mehr dazu: FAQ 10, Quotas.

Diverse Optionen

- PCIVERBOSE - Mehr Details während des Startprozesses von PCI Geräten.
- EISAVERBOSE - Mehr Details während des Startprozesses von EISA Geräten.
- PCMCIAVERBOSE - Mehr Details während des Startprozesses von PCMCIA Geräten.
- APERTURE - Liefert kernelinterne Unterstützung für VGA Bildzwischenspeicher durch Umlegen von Benutzerprozessen. Benötigt für X.
- LKM - Unterstützung für Loadable Kernel Modules (ladbare Kernelmodule). Nicht auf allen Architekturen. Siehe lkm(4) für weitere Informationen.
- INSECURE - Setzt das Kernelsicherheitsniveau auf -1. Siehe init(8) für weitere Informationen bezüglich der Kernelsicherheitseinstellungen.
- RAM_DISK_HOOKS - Erlaubt den Aufruf von rechnerabhängigen Funktionen, wenn der Ramdisktreiber konfiguriert ist.
- RAM_DISK_IS_ROOT - Zwingt die Ramdisk, root zu sein.
- CCDNBUF=integer - Setzt die Anzahl des "component buffers" von CCD(4). Standard ist 8. Für mehr über CCD siehe CCD(4) oder das Performancetuning FAQ Kapitel.
- KMEMSTATS - Damit unterhält malloc(9) (kernel memory allocator) Statistiken über seinen Gebrauch. Wenn die Option DEBUG benutzt wird, wird diese Option automatisch von config aktiviert.
- BOOT_CONFIG - Fügt die Unterstützung für die -c boot Option ein.

Netzwerkoptionen
Siehe auch das Netzwerk FAQ oder das Netzworking Performancetuning FAQ.

- GATEWAY - Ermöglich IPFORWARDING und erhöht (auf den meisten Plattformen) die Größe der NMBCLUSTERS.
- NMBCLUSTERS=integer - Steuert die Größe der mbuf cluster map.
- IPFORWARDING - Ermöglich IP routing Verhalten. Wenn diese Option aktiviert ist, wird der Rechner IP Datagramme zwischen seinen Netzwerkkarten weiterleiten, die für andere Rechner bestimmt sind.
- MROUTING - Beinhaltet Unterstützung für IP multicast Router.
- INET - Beinhaltet Unterstützung für den TCP/IP protocol stack. Diese Option wird BENÖTIGT.
- MCLSHIFT=Wert - Diese Option ist der Logarithmus zur Basis 2 der Größe des mbuf clusters. Siehe options(4) für mehr Information über diese Option.
- NS - Inkludiere Unterstützung für den Xerox XNS protocol stack. Siehe ns(4).

- ISO,TPIP - Inkludiere Unterstützung für den allgegenwärtigen OSI protocolstack. Siehe iso(4) für mehr Information.
- EON - Inkludiere Unterstützung für OSI tunneling über IP.
- CCITT,LLC,HDLC - Inkludiere Unterstützung für den X.25 protocol stack.
- IPX, IPXIP - Inkludiere Unterstützung für: Internetwork Packet Exchange protocol, das von Novell NetWare verwendet wird.
- NETATALK - Inkludiere Kernelunterstützung für die AppleTalk Protokolfamilie.
- TCP_COMPAT_42 - Vom Gebrauch dieser Option wird sehr abgeraten, daher sollte sie auch nicht aktiviert werden: TCP Fehlerkompatibilität mit 4.2BSD. In 4.2BSD waren die TCP Sequenzzahlen 32-bit signierte Werte. Moderne Implementationen des TCP verwenden unsignierte Werte.
- TCP_NEWRENO - Aktiviert "NewReno fast recovery phase", welche die Wiederherstellung eines verlorenen Segmentes pro Rücklaufzeit erlaubt.
- TCP_SACK - Aktiviert selektive Bestätigungen.
- TCP_FACK - Aktiviert weiterleitende Bestätigungen, die präzisere Schätzungen über noch ausstehende Daten während der "fast recovery phase" mittels SACK Informationen erlauben. Diese Option kann gemeinsam mit TCP_SACK benutzt werden.
- PPP_FILTER - Diese Option aktiviert auf pcap(3) basierende Filterung für PPP Verbindungen.
- PPP_BSDCOMP - PPP BSD Kompression.
- PPP_DEFLATE - Wird gemeinsam mit PPP_BSDCOMP verwendet.
- IPSEC - Diese Option ermöglicht die Unterstützung des IP security protocol. Siehe ipsec(4) für mehr Details. Dies impliziert nun die Option KEY, welche Unterstützung für PFKEYv2 bietet.
- ENCDEBUG - Diese Option ermöglicht, daß Debuginformationen aufgezeichnet werden, wenn IPSEC Fehler bemerkt.

SCSI Subsystemoptionen

- SCSITERSE - Knappere SCSI Fehlermeldungen. Dies läßt die Tabelle, um ASC/ASCQ Informationen zu dekodieren, aus, was etwa 8 Bytes einspart.
- SCSIDEBUG - Druckt zusätzliche Debuginformationen für das SCSI Subsystem auf die Konsole.

5.4 - Einen eigenen Kernel kompilieren

Vollständige Instruktionen zum Kreieren deines eigenen Kernels findest du hier in afterboot(8).

Um deinen eigenen Kernel von CDROM zu kompilieren, mußt du zunächst den Quelltext haben. Der Source ist sowohl auf der offiziellen CD (Disk 3) als auch auf den FTP Sites zu finden. Das folgende Beispiel nimmt an, dass die CD3 unter /mnt gemountet ist:

```
# cd /usr/src
# tar xvzf /mnt/src.tar.gz
```

Hinweis: Wenn du den source vom FTP Server herunterlädst, wirst du ZWEI Dateien finden, *src.tar.gz* und *srcsys.tar.gz*. Das erste ist das "userland" -- alles bis auf den Kernel, das zweite ist der Kernel Source. Lade beide runter und entpacke sie wie oben beschrieben, in den meisten Fällen willst du sowieso beide haben. Auf der CDROM sind die beiden Dateien zu einer zusammengefasst.

Am einfachsten ist es, mit einem GENERIC Kernel zu beginnen. Dieser befindet sich unter `/usr/src/sys/arch/${arch}/conf/GENERIC`, wobei ${arch} deine Architektur ist. In diesem Verzeichnis befinden sich auch weitere Beispielskonfigurationen. Hier zwei Beispiele fürs Kompilieren deines Kernels. Das erste Beispiel behandelt das Kompilieren von einem Quelltextbaum, der nur Leseberechtigung hat. Das zweite Beispiel zeigt, wie es auf einem Quelltextbaum mit Schreibberechtigung funktioniert.

```
# 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
```

entweder gefolgt von:

```
    # make depend
        - ODER -
    # make clean
# make
```

Du musst '`make depend`' laufen lassen, sobald du irgendwelche Änderungen an deinem source tree (einschliesslich Updates und Patches) gemacht hast (in anderen Worten nahezu immer, es sei denn du musst '`make clean`' benutzen).

Wenn du Änderungen an deinen Kernel-Konfigurations-Optionen gemacht hast und/oder Änderungen an deinem source tree, solltest du '`make clean`' anstelle des obigen '`made depend`' benutzen. Es ist immer besser 'make clean' zu benutzen, obwohl das in längeren Compilerläufen resultieren kann, da eben mehr erzeugt wird.

Um einen Kernel von einem Quelltextbaum mit Schreibberechtigung zu bauen, tue folgendes:

```
# cd sys/arch/$ARCH/conf
# vi IRGENDEINEDATEI (um deine gewünschten Änderung durchzuführen)
# config IRGENDEINEDATEI (hier mehr darüber :
config(8))
# cd ../compile/IRGENDEINEDATEI
# make
```

Wobei $ARCH für die von dir benutzte Architektur ist (z.B. i386). Du kannst auch ein **make depend** durchführen, um die Abhängigkeiten fürs nächste Kernelkompilieren zu schaffen.

Jetzt muß der Kernel noch an den richtigen Platz.

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

Um deinen alten Kernel zu starten, mußt du nur

```
boot> bsd.old
```

beim Starten eingeben, und dein alter Kernel wird anstelle von /bsd geladen.

Manchmal wirst du auch neue Bootblöcke installieren müssen, wenn du einen Kernel gebaut hast. Um dies zu tun, siehe FAQ 14.8, Installieren von Bootblocks. Dies wird dir eine Übersicht vom Gebrauch des OpenBSD Bootloaders geben.

5.5 - Boot-Time Konfiguration

Manchmal findet der Kernel beim Booten dein Gerät, aber eventuell den falschen IRQ. Und vielleicht brauchst du dieses Gerät sofort. Nun, ohne den Kernel neu zu bauen, kannst du mit der OpenBSD eigenen Boot-Time Konfiguration dieses Problem lösen. Dies wird aber dein Problem nur einmal lösen. Wenn du rebootest, dann mußt du diese Prozedur wiederholen. Daher ist dies nur als vorübergehende Lösung gedacht, und du solltest das Problem durch das Neukompilieren deines eigenen Kernels lösen. Dein Kernel wird die **option BOOT_CONFIG** benötigen, die GENERIC bereits beinhaltet.

Den Großteil dieses Dokumentes kannst du in der Manualseite boot_config(8) finden. Um in die Benutzerkernelkonfiguration (User Kernel Config) oder UKC zu gelangen, mußt du beim Starten die -c Option verwenden.

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

Oder welchen Kernel du auch immer laden willst. So kommst du in die UKC. Hier kannst du Befehle ausführen, die kernelspezifische Geräte ändern oder deaktivieren.

Hier eine Liste der gängigen Befehle in der UKC.

- `add` **`device`** - Füge ein Gerät durch Kopieren eines anderen hinzu
- `change` **`devno | device`** - Ändere ein oder mehrere Geräte
- `disable` **`devno | device`** - Deaktiviere ein oder mehrere Geräte
- `enable` **`devno | device`** - Aktiviere ein oder mehrere Geräte
- `find` **`devno | device`** - Suche ein oder mehrere Geräte
- `help` - Kurze Zusammenfassung dieser Befehle
- `list` - Liste ALLE bekannten Geräte au
- `exit/quit` - Setze Starten fort
- `show` **`[attr [val]]`** - Zeige alle Geräte mit Eigenschaft (attribute) und optional mit einem spezifierten Wert (value)

Wenn du einmal dein Gerät konfiguriert hast, dann steige mit quit oder exit aus UKC aus und setze das Starten fort. Nun solltest du deine Kernelkonfiguration korrigieren und einen eigenen Kernel kompilieren. Siehe Einen eigenen Kernel kompilieren für weitere Hilfe.

5.6 - Mehr Lognachrichten während des Startens

Mehr Lognachrichten während des Startens zu bekommen kann sehr hilfreich sein, wenn man Bootprobleme lösen will. Wenn du ein Problem hast und du zu wenig Informationen beim Starten erhältst, dann gib beim Starten bei "boot>" wieder "-c" ein, um zur UKC zu gelangen, dann:

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

Nun wirst du extrem ausführliche Meldungen beim Booten erhalten.

5.7 - Mittels config(8) deine Kernelbinärdatei verändern

Die **-e** und **-u** Options von config(8). können sehr hilfreich sein, und Zeit sparen, die du mit dem Neukompilierewn von Kerneln verschwenden würdest. Die **-e** Option erlaubt dir die UKC auf einem laufenden System zu benutzen. Die Änderungen werden dann beim nächsten Reboot wirksam. Die **-u** Option testet, ob irgendwelche Änderungen am laufenden Kernel während des Bootens gemacht wurden. D.h., ob du mittels **boot -c** die UKC beim Starten benutzt hast.

Das folgende Beispiel zeigt das Deaktivieren des ep* Gerätes im Kernel. Zur Sicherheit mußt du die **-o** Option benutzen, die die Änderung in die angegebene Datei schreibt. Z.B.: **config -e -o bsd.new /bsd** wird die Änderungen in bsd.new schreiben. Das folgende Beispiel verwendet die **-o** Option nicht, daher werden die Änderungen einfach ignoriert und nicht in eine Kernelbinärdatei geschrieben. Für weitere Informationen über Fehler- und Warnmeldungen siehe die config(8) Manualseite.

```
$ sudo config -e /bsd
OpenBSD 3.2 (GENERIC) #25: Thu Oct  3 19:51:53 MDT 2002
    deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
warning: no output file specified
Enter 'help' für information
ukc> ?
        help                            Command help list
        add         dev                 Add a device
        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, mitout saving changes
        quit                            Quit, saving current changes
        timezone    [mins [dst]]        Show/change timezone
        nmbclust    [number]            Show/change NMBCLUSTERS
        cachepct    [number]            Show/change BUFCACHEPERCENT
        nkmempg     [number]            Show/change NKMEMPAGES
        shmseg      [number]            Show/change SHMSEG
        shmmaxpgs   [number]            Show/change SHMMAXPGS
ukc> list
  0 audio* at
sb0|sb*|gus0|pas0|sp0|ess*|wss0|wss*|ym*|eap*|eso*|sv*|neo*|cmpci*|clcs*|clct*|auich*|autri*|auvia*|fms*|uaudio*|maestro*|esa*|yds*|emu*
flags 0x0
  1 midi* at sb0|sb*|opl*|opl*|opl*|opl*|ym*|mpu*|autri* flags 0x0
  2 nsphy* at
aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|ep*|ep*
phy -1 flags 0x0
  3 nsphyter* at
aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|ep*|ep*
phy -1 flags 0x0
  4 qsphy* at
aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|ep*|ep*
phy -1 flags 0x0
  5 inphy* at
aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|ep*|ep*
phy -1 flags 0x0
  6 iophy* at
aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|ep*|ep*
phy -1 flags 0x0
  7 eephy* at
aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|ep*|ep*
phy -1 flags 0x0
  8 exphy* at
aue*|xe*|ef*|gx*|stge*|bge*|nge*|sk*|ste*|sis*|sf*|wb*|tx*|tl*|vr*|ne0|ne1|ne2|ne*|ne*|ne*|dc*|dc*|rl*|fxp*|fxp*|xl*|xl*|ep0|ep0|ep0|ep*|ep*|ep*|ep*|ep*
phy -1 flags 0x0
[...snip...]
--- more ---
[snip]
ukc> disable ep
 65 ep0 disabled
 66 ep* disabled
 67 ep* disabled
149 ep0 disabled
150 ep0 disabled
151 ep* disabled
152 ep* disabled
202 ep* disabled
ukc> quit
not forced
```

Im obigen Beispiel werden alle ep* Geräte im Kernel deaktiviert und werden auch nicht abgefragt. In einigen Situationen, in denen du die UKC beim Booten mittels **boot -c** benutzt hast, willst du diese Änderungen dauerhaft niederschreiben. Dafür brauchst du die **-u** Option. Im folgenden Beispiel wurde die UKC gestartet und das wi(4) Gerät deaktiviert. Da diese Änderung mit boot -c NICHT dauerhaft sind, müssen die Änderungen erst geschrieben werden. Dieses Beispiel schreibt die Änderung in boot -c in eine neue Kernelbinärdatei namens bsd.new.

```
$ sudo config -e -u -o bsd.new /bsd
OpenBSD 3.2 (GENERIC) #25: Thu Oct  3 19:51:53 MDT 2002
    deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/GENERIC
Processing history...
105 wi* disabled
106 wi* disabled
Enter 'help' for information
ukc> quit
```

[FAQ Index] [Zu Kapitel 4 - Installationsleitfaden] [Zu Kapitel 6 - Netzwerk Einstellungen]

www@openbsd.org
Originally [OpenBSD: faq5.html,v 1.80]
$Translation: faq5.html,v 1.49 2003/04/09 16:41:45 jufi Exp $
$OpenBSD: faq5.html,v 1.46 2003/04/09 17:03:01 jufi Exp $

6 - Netzwerk

Inhaltsverzeichnis

6.0.1 - Bevor wir weiter gehen

Für den Rest dieses Dokumentes sei gesagt, daß es hilfreich ist, das Kapitel des FAQ Kernelkonfiguration und Einstellungen gelesen und zumindest teilweise verstanden zu haben, weiterhin helfen die Manual Seiten ifconfig(8) und netstat(1).

Wenn du ein Netzwerkadministrator bist und Routingprotokolle aufsetzt und dein OpenBSD Rechner dein Router wird, dann solltest du dein Wissen über IP Netzwerke mit Understanding IP addressing vertiefen. Dies ist wirklich ein exzellentes Dokument. "Understanding IP addressing" beinhaltet grundlegendes Wissen, auf dem man beim IP Netzwerken aufbauen kann, insbesondere wenn man mit mehreren Netzwerken arbeitet oder für sie verantwortlich ist.

Wenn du mit Anwendungen wie Web-, FTP- oder Mailserver arbeitest, dann könntest du viel vom Lesen der entsprechenden RFCs profitieren. Natürlich kannst du nicht alle lesen. Aber dennoch, lies jene, die dich interessieren oder die du bei deiner Arbeit brauchen könntest. Lies nach, wie alles funktionieren sollte. Die RFCs definieren mehrere (tausend) Standards für Protokolle im Internet und wie sie arbeiten sollten.

6.1 - Erste Netzwerkeinstellungen

6.1.1 - Identifzieren und Einstellen deiner Netzwerkkarten

Um beginnen zu können, mußt du zunächst deine Netzwerkkarte identifizieren können. Bei OpenBSD werden Netzwerkkarten nach ihrem Typ, nicht nach Verbindungsart benannt. Du kannst sehen, ob deine Netzwerkkarte initialisiert wurde, entweder schon beim Booten oder auch später mittels des Befehls dmesg(8). Weiterhin kannst du mit dem Befehl ifconfig(8) deine Karte überprüfen. Als Beispiel hier die Ausgabe in dmesg für eine Intel Fast Ethernet Netzwerk-Karte, die als Gerätenamen fxp hat.

```
fxp0 at pci0 dev 10 function 0 "Intel 82557" rev 0x0c: irq 5, address
00:02:b3:2b:10:f7
inphy0 at fxp0 phy 1: i82555 10/100 media interface, rev. 4
```

Wenn du deinen Geräte-Namen nicht weisst, sieh bitte in der Liste der unterstützten Hardware für deine Plattform nach. Du wirst eine Liste vieler bekannte Karten und ihre OpenBSD Geräte-Namen finden (wie etwa fxp), zusammen mit einer Nummer, die vom Kernel zugewiesen wird, und du hast den sogenannten "interface Name" (wie z.B. fxp0).

Du kannst herausfinden, ob deine Netzwerkkarte(n) erkannt wurde(n), indem du das ifconfig(8) Kommando benutzt. Das folgende Kommando zeigt uns alle Netzwerk-Interfaces im System. Diese Beispielausgabe zeigt ein physikalisches Ethernet Interface, eine fxp(4).

```
$ ifconfig -a
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33224
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x4
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000
lo1: flags=8008<LOOPBACK,MULTICAST> mtu 33224
fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        media: Ethernet autoselect (100baseTX full-duplex)
```

```
        status: active
        inet 10.0.0.38 netmask 0xffffff00 broadcast 10.0.0.255
        inet6 fe80::202:b3ff:fe2b:10f7%fxp0 prefixlen 64 scopeid 0x1
pflog0: flags=0<> mtu 33224
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
sl1: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
ppp1: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
tun0: flags=10<POINTOPOINT> mtu 3000
tun1: flags=10<POINTOPOINT> mtu 3000
enc0: flags=0<> mtu 1536
bridge0: flags=0<> mtu 1500
bridge1: flags=0<> mtu 1500
vlan0: flags=0<> mtu 1500
vlan1: flags=0<> mtu 1500
gre0: flags=8010<POINTOPOINT,MULTICAST> mtu 1450
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
gif1: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
gif2: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
gif3: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
```

ifconfig(8) gibt uns eine Menge mehr Informationen, als wir zu diesem Zeitpunkt benötigen. Natürlich sehen wir trotzdem unser Interface. Im obigen Beispiel ist die Netzwerkkarte bereits konfiguriert. Das ist offensichtlich, da auf fxp0 bereits ein IP Netzwerk konfiguriert ist, sprich die Werte "inet 10.0.0.38 netmask 0xffffff00 broadcast 10.0.0.255". Ausserdem sind die **UP** und **RUNNING** Flags gesetzt.

Schlussendlich fällt auf, das standardmässig eine Menge mehr Interfaces aktiviert sind. Dies sind virtuelle Interfaces, die verschiedene Funktionen haben. Informationen dazu findest du auf den folgenden man pages:

- lo - Loopback Interface
- pflog - Packet Filter Logging Interface
- sl - SLIP Network Interface
- ppp - Point to Point Protokoll
- tun - Tunnel Network Interface
- enc - Encapsulating Interface
- bridge - Ethernet Bridge Interface
- vlan - IEEE 802.1Q Encapsulation Interface
- gre - GRE/MobileIP Encapsulation Interface
- gif - Generic IPv4/IPv6 Tunnel Interface

Der 1. Schritt zur Konfiguration deiner Netzwerkkarte ist das Erstellen der **/etc/hostname.xxx** Datei, wobei der Name deiner Karte den Platz von xxx einnehmen sollte. Aus der Information der obigen Beispiele würde der Name **/etc/hostname.fxp0** lauten. Das Layout dieser Datei sollte einfach so aussehen:

```
address_family address netmask broadcast [weitere Optionen]
```

(Viel mehr Details zu dieser Datei findest du in der hostname.if(5) man page.) Eine typische Interface-Konfigurationsdatei für eine IPv4 Addresse würde so aussehen:

```
$ cat /etc/hostname.fxp0
inet 10.0.0.38 255.255.255.0 NONE
```

Du solltest auch den media type für Ethernet angeben, wenn du z.B. den 100baseTX full-duplex Modus erzwingen willst.

```
inet 10.0.0.38 255.255.255.0 NONE media 100baseTX mediaopt full-duplex
```

(Auf keinen Fall solltest du das tun, wenn nicht beide Seiten der Verbindungen auf Voll-Duplex gestellt sind ! Wenn du keine besonderen Anforderungen hast, kannst du diese media settings einfach ignorieren.)

Oder vielleicht willst du auch spezielle flags für ein einzelnes Interface benutzen. Das Format der Datei ändert sich dabei nicht besonders!

```
$ cat /etc/hostname.vlan0
inet 172.21.0.0 255.255.255.0 NONE vlan 2 vlandev fxp1
```

Der nächste Schritt ist das Einstellen deines Standard-Gateways (default gateway). Dazu trag einfach die IP deines Gateways in die Datei **/etc/mygate** ein. Dies erlaubt das Aktivieren deines Gateways beim Starten. Jetzt solltest du deine Nameserver eintragen und die Datei **/etc/hosts** einrichten. Für die Nameserver benötigst du eine Datei namens **/etc/resolv.conf**. Mehr über das Format dieser Datei findest du in der resolv.conf(5) Manual Seite. Für den Normalgebrauch ist hier ein Beispiel, in dem deine Nameserver 125.2.3.4 und 125.2.3.5 sind. Du gehörst zur Domain "deinedomaene.com".

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

Jetzt kannst du entweder rebooten oder das **/etc/netstart** Script ausführen, indem du (als root) folgendes eingibst:

```
# 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
```

Dabei werden ein paar Fehlermeldungen ausgegeben. Indem du dieses Skript ausführst, versuchst du ein paar Sachen zu konfigurieren, die bereits konfiguriert sind. Daher existieren bereits bereits einige der Routen in der kernel routing table. Von hier ab sollte dein System laufen und online sein. Du kannst hier erneut mit ifconfig(8) prüfen, ob deine Interfaces richtig konfiguriert wurden. Deine Routen kannst via netstat(1) oder route(8) überprüfen. Wenn du Probleme mit dem Routing hast, möchtest du vielleicht das -n Flag für route(8) benutzen, dass die IP-Adressen ausgibt, statt einen DNS Lookup zu machen, und den Hostnamen anzuzeigen. Hier ist ein Beispiel mit beiden Kommandos:

```
$ netstat -rn
Routing tables

Internet:
Destination        Gateway            Flags     Refs     Use    Mtu  Interface
default            10.0.0.1           UGS          0      86      -  fxp0
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      -  fxp0
10.0.0.1           aa:0:4:0:81:d      UHL          1       0      -  fxp0
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 - Einrichten deines OpenBSD Rechners als Gateway

Dies sind nur die grundlegende Informationen, um deinen OpenBSD Rechner als Gateway (auch Router genannt) einzurichten. Wenn du OpenBSD als Router im Internet verwenden willst, solltest du auch die unten folgenden Packet Filter Instruktionen beachten, um potentiell schädliche IP Daten zu blockieren. Auch solltest du wegen der Knappheit an IPv4 Adressen die Informationen bezüglich Network Address Translation beachten, um deinen IP Adressbereich zu schonen.

Der GENERIC Kernel hat bereits die Fähigkeit für IP Forwarding, aber dies muß erst eingeschaltet werden. Du solltest dies mit sysctl(8) tun. Um diese Änderung permanent einzutragen, mußt du die Datei /etc/sysctl.conf editieren. Füge einfach folgende Zeile in diese Konfigurationsdatei ein.

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

Ohne Reboot kannst du dies auch direkt mit sysctl(8) durchführen. Beachte aber, daß diese Änderung nach einem Reboot weg ist und dass der folgende Befehl als root ausgeführt werden muß.

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

Nun modifiziere die Routen der anderen Hosts. Es gibt viele verschiedene Möglichkeiten, OpenBSD als Router einzusetzen, z. B. mittels Software wie routed(8), gated, mrtd, und zebra. OpenBSD hat Unterstützung in der ports Kollektion sowohl für gated, zebra als auch mrtd.

OpenBSD unterstützt mehrere T1, HSSI, ATM, FDDI, Ethernet und serielle (PPP/SLIP) Schnittstellen.

6.1.3 - Einrichten von Aliases auf deiner Netzwerkkarte

OpenBSD hat einen einfachen Mechanismus, um IP Aliase für deine Netzwerk-Karten zu setzen. Dazu musst du einfach die Datei */etc/hostname.<if>* editieren. Sie wird beim Booten vom /etc/rc(8) Skript gelesen, das ein Teil der rc startup Hierarchie ist. Für dieses Beispiel nehmen wir an, der User hat ein Interface **dc0** und befindet sich im Netzwerk 192.168.0.0. Weitere wichtige Informationen:

- IP für dc0 ist 192.168.0.2
- NETMASK ist 255.255.255.0

Ein paar Bemerkungen zu Aliasen: Bei OpenBSD verwendet man nur den Adapternamen. Es gibt keine Unterschiede zwischen dem ersten und dem zweiten Alias. Daher muß man sie nicht - wie in einigen anderen Betriebssystemen - als dc0:0, dc0:1 bezeichnen. Wenn du dich auf einen speziellen IP Alias beziehst oder einen hinzufügst, dann nimm "`ifconfig int alias`" anstelle nur "`ifconfig int`" auf der Befehlszeile. Du kannst Aliase mit "`ifconfig int delete`" löschen.

Angenommen du verwendest mehrere IP Adressen im selben IP Subnetz mit Aliases, dann ist die Netzmaskeneinstellung für jeden Alias 255.255.255.255. Sie müssen nicht der Netzmaske der ersten IP der Netzwerkkarte folgen. In diesem Beispiel */etc/hostname.dc0* werden zwei Aliase zur Netzwerkkarte dc0 hinzugefügt, die als 192.168.0.2 mit Netzmaske 255.255.255.0 konfiguriert wurde.

```
# cat /etc/hostname.dc0
inet 192.168.0.2 255.255.255.0 media 100baseTX
inet alias 192.168.0.3 255.255.255.255
inet alias 192.168.0.4 255.255.255.255
```

Wenn du einmal diese Datei erstellt hast, benötigst du einen Reboot, um die Änderung automatisch durchführen. Du kannst aber auch die Aliase manuell mit ifconfig(8) hochbringen. Für den ersten Alias geht das so:

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

Um die Aliases zu sehen:

```
$ ifconfig -A
dc0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST>
        media: Ethernet manual
        inet 192.168.0.2  netmask 0xffffff00 broadcast 192.168.0.255
        inet 192.168.0.3 netmask 0xffffffff broadcast 192.168.0.3
```

6.2 - Packet Filter (PF)

Hinweis: Ab OpenBSD 3.2 sind die Funtionen von */etc/nat.conf* in */etc/pf.conf* eingebaut worden, die beiden Dateien somit verschmolzen.

Die neue */etc/pf.conf* Datei hat vier Teile:

- **Options:** Verschiedene Optionen, um das Verhalten von PF zu kontrollieren.
- **Scrub:** Überarbeitung von Paketen zur Normalisierung und Defragmentierung.
- **NAT and Redirection Rules:** NAT erlaubt den Zugang von vielen Maschinen durch eine IP-Adresse. Redirection erlaubt hereinkommende Anfragen an eine bestimmte Maschine hinter dem NAT weiterzuleiten.
- **Filter Rules:** Erlaubt das selektive Filtern oder Blocken von Paketen, die über eines der Interfaces laufen.

Keine dieser Sektionen muss existieren, aber wenn, dann müssen sie in der oben genannten Reihenfolge sein.

Das Packet Filter Subsystem wurde erstellt, um zwei Dinge zu tun: die Rechte von Forwarding auf Paketlevel zu setzen und zu überprüfen (Paketfilter) und das mapping von Hosts/Subnetzen zu einer Reihe von externen Adressen zu steuern (NAT). Die Konfigurationsdateien für diese zwei Dienste sind */etc/pf.conf(5)* und */etc/nat.conf(5)*.

Um diese Dienste auf deinem System zu starten, musst du die Datei */etc/rc.conf(8)* editieren und die Zeile wie folgt setzen:

```
pf=YES
```

Wenn du NAT benutzt, musst du höchstwahrscheinlich auch den sysctl(8) Wert `net.inet.ip.forwarding` auf 1 setzen. Das kannst du machen, indem die relevanten Zeilen in der Datei */etc/sysctl.conf(5)* änderst und deinen Computer rebootest.

Wenn du Packet Filter in deinen Kernel einkompiliert hast, aber es nicht in deiner */etc/rc.conf(8)* Datei aktiviert hast, kannst du das mit dem pfctl(8) Befehl nachholen.

```
# pfctl -R /etc/pf.conf
# pfctl -N /etc/nat.conf
```

```
# pfctl -e
```

Die erste Zeile setzt den Filter mittels */etc/pf.conf* und die zweite aktivert das NAT mit den Regeln aus */etc/nat.conf* (mehr zu NAT später in Sektion 6.3, NAT), und zuletzt aktiviert die letzte Zeile PF. Das kann man auch in einer einzigen Zeile kombinieren.

```
# pfctl -R /etc/pf.conf -N /etc/nat.conf -e
```

Wenn du Änderungen an */etc/pf.conf* vornimmst, nachdem du PF gestartet hast, kannst du deine Regeln neu laden, indem du die passende Datei neu lädst:

```
# pfctl -R /etc/pf.conf
```

Dieses Dokument wird einige grundlegende pf.conf(5) und nat.conf(5) Konfigurationen weiter unten behandeln. Du kannst dir auch das resultierende ruleset ansehen, mit all den Anpassungen, die unten im Details erklärt sind. Weitere Packet Filter Informationen finden sich auf der Packet Filter web site und im Packet Filter HOWTO.

Packet Filter

Um den Packet Filter beim Booten zu aktivieren, musst du */etc/rc.conf* so anpassen, dass dann `pf=YES` enthalten ist. Packet Filter (pf) wird von */etc/pf.conf* kontrolliert, das beim Booten gelesen wird. Eine detailliertere Erklärung bekommt man hier: pf.conf(5). In den folgende Beispielen wird `fxp0` das externe Interface zum Internet hin darstellen. Abhängig von deinen Netzwerkkarten in deinem Computer wird das bei dir natürlich anders sein. Diese Regeln gehen von einer full-time Internetverbindung aus, wie man sie z.B. bei einem Webserver hat.

Packet Filter Regeln werden sequentiell vom Anfang bis zum Schluss abgearbeitet; das hilft jedes Paket beim Passieren jeder Regel zu beobachten, bevor es dann seinen Zielort erreicht.

Z. B. erlauben die Standardregeln, daß alle Pakete rein und raus dürfen:

```
pass out all
pass in all
```

Das ist die Kurzform von:

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

was man auch auffassen kann als "lasse alle hereinkommenden Pakete von jeder Quelle zu jedem Ziel passieren", mit einem eingebauten "auf jedem Interface (was immer angenommen wenn kein Interface sepzifiziert wird) für jede Adress-Familie , inet (v4) oder inet6 (v6)".

Das ist natürlich noch kein wirklicher Filter. Nützlichere Filter basieren auf der Address-Familie (IPv4 oder IPv6), Protokoll(en) und Port(s), die von den Diensten genutzt werden, die gefiltert werden sollen. Jedes der in */etc/protocols(5)* aufgeführten Protokolle kann entweder mit Namen oder Nummer angegeben werden, wir werden uns allerdings nur mit tcp(4), udp(4) und icmp(4) befassen.

Nun nehmen wir mal an, wir wollten keinerlei eingehende Verbindung zum TCP Port 3306 (MySQL) zulassen, weil die Datenbank nur von localhost aus erreichbar sein soll. Unser "ruleset" würde dann wie folgt aussehen:

pass out all pass in all block in on fxp0 inet proto tcp from any to any port 3306

Das bedeutet "blocke alle hereinkommenden IPv4 (inet) Pakete von jeder Quelle zu jeglichem Ziel mit dem Zielport 3306." Es ist notwendig, dass du bei jedem port-basiertem Filter ein Protokoll angibst und die Adress-Familie spezifizierst. Dienste, die in der Datei /etc/services(5) definiert sind, kannst du auch einfach bei ihrem Namen nennen, wie z.B. *www* oder *mysql*. Ein Paket, das in Richtung des TCP Port 3306 auf Interface `fxp0` hereinkommt, wird die erste "pass in" Regel noch passieren, und dann von der "block in port 3306" Regel geblockt werden. Wenn du die Reihenfolge deiner "incoming" Regeln umgedreht hast (denke immer daran, die Reihenfolge ist wichtig):

```
pass out all
block in on fxp0 inet proto tcp from any to any port 3306
pass in all
```

Pakete, die an den TCP Port 3306 adressiert sind, würden dann passieren, da die letzte Regel es allen Paketen erlaubt, den Filter zu passieren. Es ist wichtig, dass im Auge zu behalten, wenn man Filter-Regeln schreibt: **Die letzte passende Regel gibt den Ausschlag**.

Natürlich gibt es jeder Regel eine Ausnahme, so auch hier. Die *quick* Option z.B. lässt das Paket schickt das Paket gemäss der ersten dafür passenden Regel weiter. Sehen wir uns mal unser obiges, fehlerhaftes Beispiel an, wenn wir *quick* zur "block in" Regel hinzufügen:

```
pass out all
block in quick on fxp0 inet proto tcp from any to any port 3306
pass in all
```

Ein Paket, dass an unseren Host und den TCP Port 3306 adressiert ist, trifft zuerst auf die "block in quick" Regel und wird sofort verworfen. Alle Pakete, die an andere Ports oder Protokolle adressiert sind, finden keine passende Regel, bis sie unsere "pass in" Regel erreichen, die allen Paketen das Passieren gestattet.

Standardmäßiges Ablehnen

Die sicherste Paketfilter "policy" ist ein standardmässiges Ablehnen. Jeglicher Traffic, der nicht explizit erlaubt ist, wird abgelehnt. Diese

Policy ist bedeutend sicherer als jeden einzelnen zu schützenden Dienst abzusichern, erlaubt kleinere Regelwerke, und schützt auch vor einem möglicherweise falsch konfiguriertem Dienst, der fälschlicherweise vergessen wurde.

Sehen wir uns nun ein weiteres Beispiel-Regelwerk (ruleset) Zeile für Zeile an. Hier ein Beispiel für einen Webserver mit einer Standard-Ablehnungs-Policy, die SSH Verbindungen zulässt (zum Administrieren) und Verbindungen auf http (port 80) und https (port 443).

```
block in on fxp0 from all
pass  in on fxp0 inet proto tcp from any to any port 22
pass  in on fxp0 inet proto tcp from any to any port 80
pass  in on fxp0 inet proto tcp from any to any port 443
pass out on fxp0 from all
```

Das wir eingehende Verbindungen von überall zu den Ports 22(ssh), 80(http) und 443(https) erlauben. Alle anderen Verbindungsversuche wurden fallengelassen, und alle ausgehenden Verbindungen erlaubt. Das ist schon ein recht strenges Regelwerk. Aber was, wenn nur interne Hosts aus deinem 1.1.1.0 Address-Block Zugriff auf SSH haben sollen, aber HTTP und HTTPS von überall aus erreichbar sein sollen ?

```
block in on fxp0 from all
pass  in on fxp0 inet proto tcp from 1.1.1.0/24 to any port 22
pass  in on fxp0 inet proto tcp from any to any port 80
pass  in on fxp0 inet proto tcp from any to any port 443
pass out on fxp0 from all
```

Recht nett, aber wenn nur eine einzige Machine (1.1.1.1) den Webserver administrieren darf? In diesem Fall ändern wir dies:

```
pass  in on fxp0 inet proto tcp from 1.1.1.0/24 to any port 22
```

in:

```
pass  in on fxp0 inet proto tcp from 1.1.1.1/32 to any port 22
```

Beispielregeln

Hier sind einige gute Regeln, die fast jeder gebrauchen kann (Mit der Annahme, dass fxp0 das externe Interface mit der Anbindung ins Internet ist). Zuerst schaffen einen einfachen Schutz gegen "address spoofing". Diese Adressen sollten normalerweise nicht im Internet herumfliegen, und wenn doch, ist das nicht gut, also blocken wir sie:

```
block out quick on fxp0 inet from any to { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 } to any
block out quick on fxp0 inet from any to { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 }
```

Unser Regelwerk sieht schon recht gut aus; wenn wir alles zusammentun, sieht das ganze wie folgt aus:

```
# adress spoofing wird geblockt
block in quick on fxp0 inet from { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 } to any
block out quick on fxp0 inet from any to { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 }

# nur unsere Administrationsmaschine darf sich auf ssh verbinden
pass in quick on fxp0 inet proto tcp from 1.1.1.1/32 to any port 22

# andere duerfen http und https erreichen
pass in quick on fxp0 inet proto tcp from any to any port 80
pass in quick on fxp0 inet proto tcp from any to any port 443

# den Rest mit einem pauschalen block ausschliessen
block in quick on fxp0 from any to any

# und ausgehenden Traffic rauslassen
pass out on fxp0 from any to any
```

Packet Logging

Nun, dies ist zwar schon recht gut, aber es könnte besser sein. Was, wenn wir z.B. alle Verbindungsversuche zu Port 22(ssh) loggen wollten, die von unserer Firewall blockiert werden? Einfach - dafür hat Packet Filter das *log* Stichwort:

```
pass in quick on fxp0 inet proto tcp from 1.1.1.1/32 to any port 22
block in log quick on fxp0 inet proto tcp from any to any port 22
```

Diese Regel wird erlauben, daß sich unser Administrationsrechner auf Port 22 verbinden darf, aber alle anderen Verbindungsversuche auf Port 22 ablehnt und aufzeichnet. Pakete, die geloggt wurden (also die mitprotokolliert worden sind) werden zum pflog0 Interface gesendet, das wiederum vom pflogd(8), überwacht wird, der normalerweise die Pakete im tcpdump(8) Binärformat in die Datei */var/log/pflog* schreibt. Der

pflogd(8) wird standarmässig in */etc/rc(8)* gestartet, wenn pf in */etc/rc.conf(8)*. eingeschaltet ist. Lesen kannst du diese Logfiles wie folgt:

```
# tcpdump -n -e -ttt -r /var/log/pflog
```

Man sollte wissen, dass die Benutzung von tcpdump KEINE real-time Anzeige der pflog Datei ermöglicht. Die kann man aber wie folgt bekommen:

```
# tcpdump -i pflog0
```

Man könnte tcpdump auch benutzen, um den Focus beim Debugging zu schärfen:

```
# tcpdump -e -i pflog0 port 80
```

Wenn man das eingibt, hat das KEINEN Einfluss auf die Daten, die in die Datei */var/log/pflog* geschrieben werden.

Wenn man seine Logdateien untersucht, sollte man besondere Aufmerksamkeit auf das "verbose protocol decoding" von tcpdump richten (wird mittels der -v Kommandzeilen Option aktiviert). Tcpdump's protocol decoder haben keine perfekte Sicherheitshistorie. Zumindest in der Theorie ist ein verzögerter Angriff über die "partial packet payloads" denkbar, der vom Logging-Device aufgezeichnet wird.

Auch wer überhaupt Zugriff auf die Logs hat, sollte man genau prüfen. Pflogd greift sich 96 Bytes des Paketes und loggt es. Zugriff auf die Logs kann teilweisen Zugriff auf sensible Teile von Paketen bedeuten (wie z.B. bei telnet(1) oder ftp(1) Logins).

Packet Logging mittels syslog

In vielen Situationen ist es wünschenswert, die Firewall-Logs im ASCII Format zu haben und/oder sie an einen remote logging Server zu senden. All das kann man mittels 2 kleinen Shell-Skripten und geringfügigen Änderungen an den OpenBSD-Konfigurationsdateien erreichen.

Syslogd(8) ist der Standard-Daemon für das logging, er loggt in ASCII und ist auch in der Lage, die Logs an einen anderen, entfernten Log-Server zu senden.

Zuerst müssen wir einen User namens *pflogger* mit einer .nologin. Shell erzeugen. Der einfachste Weg dazu ist adduser(8).

Nach dem Erzeugen des Users *pflogger* erzeuge die folgenden zwei Skripte:

/etc/pflogrotate

```
FILE=/home/pflogger/pflog5min.$(date "+%Y%m%d%H%M")
kill -ALRM $(cat /var/run/pflogd.pid)
if [ $(ls -l /var/log/pflog | cut -d " " -f 8) -gt 24 ]; then
        mv /var/log/pflog $FILE
        chown pflogger $FILE
        kill -HUP $(cat /var/run/pflogd.pid)
fi
```

/home/pflogger/pfl2sysl

```
#!/bin/sh
# feed rotated pflog file(s) to syslog
for logfile in /home/pflogger/pflog5min* ; do
        tcpdump -n -e -ttt -r $logfile | logger -t pf -p local0.info
        rm $logfile
done
```

Editiere den cron job für den User *root*

```
# crontab -u root -e
```

und füge die folgende zwei Zeilen ein:

```
# rotate pf log file every 5 minutes
0-59/5 *       *       *       *       /bin/sh /etc/pflogrotate
```

Erzeuge einen cron job für den User *pflogger*

```
# crontab -u pflogger -e
```

und füge die folgende zwei Zeilen ein:

```
# feed rotated pflog file(s) to syslog
0-59/5 *       *       *       *       /bin/sh /home/pflogger/pfl2sysl
```

Füge die folgende Zeile in */etc/syslog.conf* ein:

```
local0.info    /var/log/pflog.txt
```

Wenn du zu einem entfernten Log-Server loggen willst, musst du ausserdem folgende Zeile einfügen:

```
local0.info    @syslogger
```

und stelle sicher, dass der Host *syslogger* in der Datei /etc/hosts(5) definiert wurde.

Alle geloggten Pakete werden nun nach */var/log/pflog.txt* gesendet. Wenn man die zweite Zeile hinzufügt, werden sie alle zum entfernten Logging Host *syslogger* geschickt.

/etc/pflogrotate verarbeitet nun und löscht dann */var/log/pflog*, so dass eine Rotation von *pflog* mittels newsyslogd(8) nicht mehr notwendig und daher abgeschaltet werden sollte. Trotzdem ersetzt */var/log/pflog.txt /var/log/pflog* und seine Rotation sollte daher aktiviert werden. Ändere */etc/newsyslog.conf* wie folgt:

```
#/var/log/pflog         600     3       250     *       ZB      /var/run/pflogd.pid
/var/log/pflog.txt      600     7       *       24
```

Pf wird nun im ASCII-Format nach /var/log/pflog.txt loggen. Wenn es so in */etc/syslog.conf* konfiguriert ist, wird es das Logfile auch zu einem anderen Log-Server schicken. Das Logging geht nicht sofort los, sondern es dauert etwa 5-6 Minuten (dem cron-Job-Intervall) bis die geloggten Pakete in der Datei erscheinen.

Multiple Protokolle

Was aber, wenn wir Verbindungen zu einem Dienst erlauben wollen, der über mehrere Protokolle läuft, wie z.B. BIND, der TCP und UDP benutzt ? Mit Packet Filter kannst du mehrere Regeln zu einer zusammenfassen (mehr dazu später):

```
# Passierenlassen von DNS traffic für BIND
pass in quick on fxp0 inet proto { tcp, udp } from any to any port 53
```

Beachte die Spaces (Leerzeichen) auf beiden Seiten der '{ }' Zeichen. Das ist viel netter, als die Variante, die du sonst benutzen müsstest:

```
pass in quick on fxp0 inet proto tcp from any to any port 53
pass in quick on fxp0 inet proto udp from any to any port 53
```

Packet Normalization

Packet Normalization bedeutet das erneute Zusammenfügen von fragmentierten Paketen und das Leeren der IP Options. Einige BSs und Applikationen haben Schwierigkeiten mit anormalen und fragmentierten Paketen, und daher ist es im allgemeinen richtig und gut Pakete von den Filter-Regeln normalisieren zu lassen. Das macht man mit Hilfe der **scrub** Direktive, die man wie folgt benutzt:

```
scrub in all
```

Das fügt dem System eine minimale zusätzliche Last hinzu, und verlangt ein wenig Speicher, um die Paket-Fragmente zu cachen. Die Vorteile von Packet Normalization wiegen diesen kleinen Nachteil jedoch fast jedesmal aus.

IP Optionen

PF blockt standardmässig Pakete, deren IP Optionen gesetzt sind. Das kann die Arbeit von "OS fingerprinting" Werkzeugen wie nmap erschweren. Wenn du eine Anwendung hast, die das Passierenlassen solcher Pakete erfordert, wie z.B. multicast oder IGMP, kannst du die **allow-opts** Direktive benutzen:

```
pass in quick on fxp0 all allow-opts
```

TCP Flags, bestehende Verbindungen und keep state

Packet Filter kann also Pakete auf Basis von TCP flags filtern und kann bereits bestehende Verbindungen verfolgen und den Status der Verbindungen ebenso. Es wird also empfohlen, dass alle User, die Paketen auf Basis von TCP flags filtern wollen, die Rolle jedes einzelnen flags auch wirklich verstehen. Wenn du z.B. alle Pakete ablehnen willst, die die FIN, URG, und PSH flags gesetzt haben (wie z.B. ein nmap OS fingerprinting Versuch) könntest du dazu ein Regel wie diese hier verwenden:

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

(Danke an Kyle Hargraves für diesen Tip) Der nächste coole Trick von Packet Filter ist seine Fähigkeit, sich den Zustand eine Verbindung zu merken (sog. stateful filtering). Den "state" (Status) zu verwalten oder zu halten, wurde zum Beispiel als "nichts sagen, bis man angesprochen wird" beschrieben, oder in anderen Worten: sobald eine Verbindung einmal hergestellt wurde, müssen Pakete nicht mehr die ganzen Regeln passieren. Das ist ein sehr mächtiges Feature, das es erlaubt viel einfachere und dennoch sicherere Regeln zu schreiben.

Als Beispiel sehen wir uns mal an, wie wir "state" in unser bisheriges Regelwerk einbauen können (na, schon verwirrt?). Zur Wiederholung: Wir erlauben administrativen Zugriff aus unserem Class C-Netz auf port 22 (ssh) und erlauben hereinkommenden Web-Traffic auf den Port 80(http) und 442 (https). Alles andere wird abgeblockt. Was aber, wenn ich z.B. ssh(1) vom Webserver wonaders hin machen will ? Was, wenn ich lynx(1) benutzen muss, um etwas in den FAQs nachzulesen ? Nunja, das kann ich zunächst nicht, weil ich ja allen einkommenden Traffic bis auf die angebenen Ports geblockt habe. Während das natürlich das sicherste ist, ist es mit Sicherheit auch ziemlich unbequem. Indem man die Worte *keep state* in unsere "pass out" Regel einfügt, erlauben wir automatisch, dass eingehende Pakete, die Antworten auf unsere ausgehenden Verbindungsaufbauten sind, passieren dürfen, wie z.B. beim Web-Browsen. Denke daran, wir müssen nicht angeben, für welches

Protokoll wir "keep state" einbauen.

```
block in  on fxp0 inet proto tcp all
pass  in  on fxp0 inet proto tcp from 1.1.1.0/24 to any port 22 keep state
pass  in  on fxp0 inet proto tcp from any to any port 80 keep state
pass  in  on fxp0 inet proto tcp from any to any port 443 keep state
pass  out on fxp0 inet proto tcp all keep state
```

Diese kleine Änderung wird die Flexibilität und Sicherheit unseres Regelwerkes erheblich erhöhen: Zum Beispiel erlauben wir im Regelwerk oben allen TCP traffic zu den Ports 80 & 442. Das können wir noch weiter einschränken. Um den Aufbau einer TCP Verbindung zu erlauben, müssen wir nur den anfänglichen Handshake zulassen; nachdem das geschehen ist, können wir den Traffic zu diesem Port blocken und es "keep state" überlassen, die Verbindung zu managen. Um den anfänglichen Handshake zuzulassen brauchen wir nur Pakete mit gesetztem SYN-Flag und nicht gesetztem ACK-Flag zuzulassen. Dadurch, das wir nur Pakete durchlassen, die nur das SYN Flag gesetzt haben, verhinden wir auch viele Formen des Portscanning. Flags S/SA bedeutet: Von den Flags S (SYN) und A (ACK) darf nur S gesetzt sein. Andere Flags werden nicht betrachtet. Die Regeln sehen jetzt wie folgt aus:

```
block in  on fxp0 inet proto tcp all
pass  in  on fxp0 inet proto tcp from 1.1.1.0/24 to any port 22 \
       flags S/SA keep state
pass  in  on fxp0 inet proto tcp from any to any port 80 \
       flags S/SA keep state
pass  in  on fxp0 inet proto tcp from any to any port 443 \
       flags S/SA keep state
block out on fxp0 inet proto tcp all
pass  out on fxp0 inet proto tcp all flags S/SA keep state
```

Führen wir jetzt alle Dinge zusammen, indem wir alle bisherigen Regeln einfach in ein einziges Regelwerk umwandeln. Dieses Regelwerk wird IPv4 unterstützen, die Standard-Regel wird zunächst alles abblocken, Management-Verbindungen aus dem internen Netzwerk (via SSH) erlauben und ebenso hereinkommenden Traffic zu den Ports 80 (http) und 443(hhtps). Es wird auch nicht routbare gespoofte IP Adressen abblocken, und alle Pakete verwerfen, die zu fragmentiert sind, um sie zu untersuchen. Ein recht gutes Setup für einen öffentlichen Webserver. Und so könnte */etc/pf.conf* aussehen:

```
# Aufräumen von fragmentierten und abnormalen Paketen
scrub in all

# Das Spoofen nicht-routbarer Addressen verhindern
block in quick on fxp0 inet from { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 } to any
block out quick on fxp0 inet from any to { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 }

# Standardmässig alles verbieten, was nicht später explizit
# erlaubt wird.
block in on fxp0 all

# Andere dürfen http und https benutzen
pass in on fxp0 inet proto tcp from any to any port 80 \
       flags S/SA keep state
pass in on fxp0 inet proto tcp from any to any port 443 \
       flags S/SA keep state

# Ausgehender Traffic darf raus und es wird ein "state" dabei erzeugt
# Alle Protokolle einschliesslich TCP, UDP und ICMP dürfen raus und
# erzeugen einen "state", so das externe DNS-Server auf unsere DNS-Requests
# (UDP) antworten können.
block out on fxp0                all
pass  out on fxp0 inet proto tcp  all flags S/SA keep state
pass  out on fxp0 inet proto udp  all            keep state
pass  out on fxp0 inet proto icmp all            keep state
```

Obwohl das schon ganz gut aussieht, erlaubt es Packet Filter, einige Dinge zu tun und zu nutzen, die deine *pf.conf* Datei besser lesbar machen und leichter zu pflegen.

Sets

Sets sind nützliche "shortcuts" zum Schreiben einfacher und sauberer Regel in PF. Was wäre, zum Beispiel, wenn du Verbindungen zu einem Dienst erlauben musst, der auf verschiedenen Protokollen läuft, wie etwa BIND, der TCP und UDP benutzt ?

```
pass in quick on fxp0 inet proto { tcp, udp } from any to any port 53
```

Beachte die Spaces auf beiden Seiten der '{ }' Klammern.

Gruppen von IPs, die irgendwie miteinander in Beziehung stehen, können ebenfalls in sets zusammengefasst werden, die überall benutzt werden können, wo man auch eine einzelne IP benutzen kann. Nehmen wir als Beispiel unsere anti-spoofing Regeln von oben:

```
# Erlaube niemandem nicht-routebare Addressen zu spoofen
block in quick on fxp0 inet from { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 } to any
block out quick on fxp0 inet from any to { 127.0.0.0/8, 192.168.0.0/16, \
172.16.0.0/12, 10.0.0.0/8 }
```

Variablen-Auswertung

Ein Problem der obigen Beispiel-*pf.conf*Datei ist, dass du eine Menge Zeilen auswechseln musst, sobald du deine Netzwerkkarte oder IP-Adresse ändern musst. Das kann man durch Variablen-Asuwertung vermindern:

```
NoRouteIPs="{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }"
ExtIF="fxp0"
block in quick on $ExtIF from $NoRouteIPs to any
block out quick on $ExtIF from any to $NoRouteIPs
```

Alles zusammengenommen

Fügen wir nun alles zusammen und betrachten die Eleganz der Datei:

```
# Sinnvolle Variablen definieren
ExtIF="fxp0"              # External Interface
IntNet="1.1.1.0/24"       # Our internal network
NoRouteIPs="{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }"
Services="{ www, https }"

# Aufräumen bei fragmentierten und unnormalen Paketen
scrub in all

# Keine nicht-routbaren Addressen erlauben
block in  quick on $ExtIF from $NoRouteIPs to any
block out quick on $ExtIF from any to $NoRouteIPs

# standardmässig alle einkommenden Pakete blocken, mit Ausnahmen jener,
# die explizit erlaubt sind
block in on $ExtIF all

# Andere dürfen HTTP und HTTPS benutzen
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

pass  in on $ExtIF inet proto tcp from any to any port $Services \
       flags S/SA keep state

# und ausgehender Traffic darf raus,
# alle Protokolle, einschliesslich TCP, UDP und ICMP, es wird ein "state",
# erzeugt, so dass externe DNS Server auch auf unsere DNS Anfragen antworten können
(UDP).
block out on $ExtIF                 all
pass  out on $ExtIF inet proto tcp  all flags S/SA keep state
pass  out on $ExtIF inet proto udp  all            keep state
pass  out on $ExtIF inet proto icmp all            keep state
```

Wenn du Probleme bekommst, wirst du vermutlich gerne das Logging bei bestimmten Regeln aktivieren, um effektiv deine Probleme lösen zu können, wie z.B.:

```
pass in log quick on fxp0 proto tcp from 1.1.1.0/24 to any port 22
```

pflogd(8) schreibt Logeinträge in */var/log/pflog*. Denke daran, dass */var/log/pflog* eine Binär-Datei ist, und von tcpdump(8) gelesen werden soll, NICHT direkt von Menschen.

Wenn du die Konfigurationsdatei veränderst, so dass jetzt in das Logfile geschrieben wird, denke daran, dass du **pfctl -R /etc/pf.conf** durchführen musst, damit die Änderung auch wirksam wird !

6.3 - NAT

Hinweis: Ab OpenBSD 3.2 sind die NAT Funktionen in die Datei */etc/pf.conf* integriert und nicht mehr in der separaten Datei */etc/nat.conf*, die noch in OpenBSD 3.0 und 3.1 genutzt wurde.

Hinweis: Packet Filter ist das Filtersystem in OpenBSD 3.0 und höher. Wenn du nach der IPF/IPNAT FAQ für OpenBSD 2.9 und vorher suchst, klicke hier.

6.3.1 NAT Einführung

Kapiteleinführung

Gemäß RFC 1631 bietet NAT einen einfachen Weg, um interne Netzwerke auf eine einzige routebare ("reale") Internetadresse umzulegen. Dies ist sehr nützlich, wenn man nicht für jeden Rechner des internen Netzwerkes eine offizielle Adresse zugewiesen bekommen hat. Wenn man ein privates/internes Netzwerk besitzt, dann kann man diese (in RFC 1918 definierten) Adressbereiche benutzen:

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)

Es wird angenommen, dass der User bereits eine OpenBSD-Maschine mit zwei Netzwerkkarten installiert und eingerichtet hat (eine hin zum Internet und die anderen für das lokale Netzwerk).

Konfiguration

Als Beispiel nehmen wir das unten beschriebene System. Dein Setup wird höchstwahrscheinlich davon abweichen, sei also sehr vorsichtig damit, einfach irgendetwas von hier abzutippen und dann zu erwarten, dass das auch funktioniert.

Intel EtherExpress Pro/100 **fxp0** verbunden mit dem EXTERNEN LAN (oder WAN) **IP Addresse:** 24.5.0.5 **Netmaske:** 255.255.255.0
Compaq Netelligent 10/100Mb **tl0** Verbunden mit dem INTERNEN LAN **IP Addresse:** 192.168.1.1 **Netmaske:** 255.255.255.0

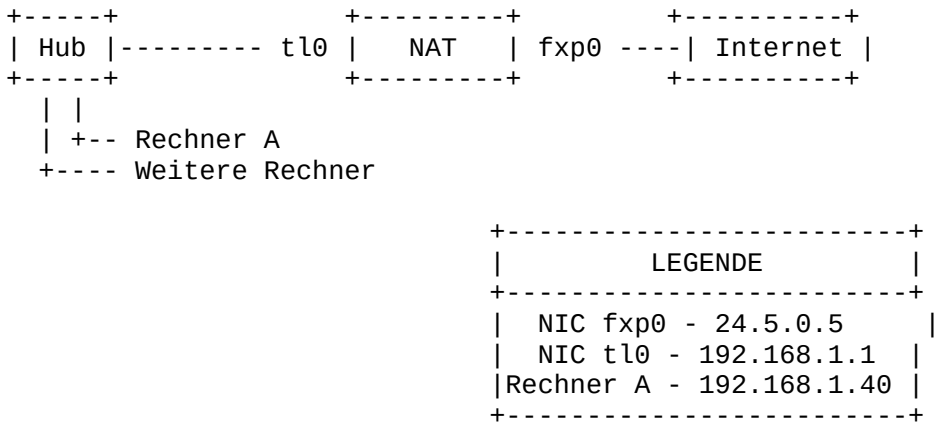
Externe, Internet-routebare IP (vom ISP bereitgestellt, in diesem Beispiel ein Kabelmodemprovider)

IP Addresse: 24.5.0.5 **Netmaske:** 255.255.255.0 **Gateway:** 24.5.0.1

LAN - Lokales Netzwerk

In diesem Beispiel-Szenario werden die Rechner im LAN das IP Adressierungsschema 192.168.1.xxx (wobei xxx eine eindeutige Nummer darstellt) verwenden. Es kann eine große Anzahl verschiedener Betriebssysteme im internen LAN wie Windows 98, Windows NT, OpenBSD und Linux geben, aber das Betriebssystem des Client ist wirklch keine Sache des NAT. Für dieses Beispiel wird der Rechner im LAN die IP Adresse 192.168.1.40 annehmen.

Diagramm der Konfiguration



6.3.2 Network Address Translation

Einführung zu NAT

Jede Station im Internet benötigt eine einzigartige IP Addresse. Zumindest mit IPv4 gibt es ziemlich begrenzte Anzahl von einzigartigen und verfügbaren IP-Adressen, und als Ergebnis davon sind sie nicht frei erhältlich. Die meisten Low-Cost ISPs begrenzen daher eine Site mit 1 bis 30 Adressen, und obwohl Organisationen mit grösserem Budget sich grössere Blöcke leisten können, ist es in den meisten Fällen ohne grossen Vorteil aber mit höherem Risiko verbunden, jeden Computer mit einer individuellen Adresse im Internet zu haben.

Network Address Translation, oder NAT, (auch bekannt als "IP Masquerading" wenn du aus einem Linux-Hintergrund kommst) erlaubt es

mehreren Computern, sich "hinter" einer (oder einer kleinen Anzahl von) IP Addressen zu verbergen. Jeder "interne" Computer hat eine lokal zugewiesene, unregistrierte IP Addresse (gemäß RFC 1918), und alle benutzen gleichzeitig die gleiche externe IP Addresse.

NAT arbeitet auf ziemlich einfache Weise. Wenn ein Rechner im LAN sich mit einem Rechner im Internet verbinden will, sendet er ein TCP Paket mit einer Verbindungsanfrage. Innerhalb des TCP Paketdatenkopfes ("headers") steht die Rechner IP Adresse (hier 192.168.1.40) und die erwünschte Server IP Adresse (z. B. 123.45.67.89). Die Maschine mit NAT fängt das TCP Paket ab und ändert die Rechner IP Adresse in die Adresse des Rechners, der mit dem Internet verbunden ist (z. B. 24.5.0.5). Dies täuscht den Server eigentlich, indem es ihn glauben macht, daß die Verbindung vom NAT Rechner und nicht vom eigentlichen Client kommt. Der Server schickt dann die Antworten zurück zum NAT Rechner. Wenn der NAT Rechner die Antwort erhält, dann übersetzt er die Zieladresse zurück von seiner eigenen IP zu der vom Client und schickt das Paket an den Client weiter. Der Client bekommt von alledem nichts mit und die vorgetäuschte Internetverbindung ist für den Benutzer und seine Anwendung total transparent.

Das folgende Beispiel zeigt NAT noch ein bißchen deutlicher:

```
Client ----------------- tl0 [ NAT ] fxp0 ---------- Internet Host
192.168.1.40 --- 192.168.1.1 [ NAT ] 24.5.0.5 --- 123.45.67.89

AUSGEHENDES TCP Paket                    AUSGEHENDES TCP Paket
Von: 192.168.1.40   >>=== NAT ===>>     Von: 24.5.0.5
Nach: 123.45.67.89                       Nach: 123.45.67.89

INCOMING TCP Paket                      EINGEHENDES TCP Paket
Von:  123.45.67.89                        Von:  123.45.67.89
Nach: 192.168.1.40   <<=== NAT ===<<     Nach: 24.5.0.5
```

Warum NAT verwenden

In meiner neuen Wohnung bekam ich ein Kabelmodem und somit ein kleines Problem. Wie können meine Zimmerkameraden Internetanschluß bekommen, wenn das Kabelmodem in meinem Zimmer steht? Ich hatte ein paar Alternativen wie zusätzliche IP Adressen kaufen, einen Proxyserver aufsetzen oder eben NAT einsetzen. (Laß dich nicht vom Kabelmodembeispiel täuschen: NAT ist fähig, ein großes Netzwerk mit hunderten oder auch tausenden Computern zu maskieren!)

Es gibt viele Gründe, warum ich NAT aufsetzen wollte. Nummer eins: Geld sparen. Zwei Zimmerkameraden in meinem Haus haben ihre eigenen PCs und ich besitze insgesamt 3 Computer. Mein ISP erlaubt nur drei IP Adressen pro Haushalt. D. h., daß wir nicht genug IPs hätten, um jedem Rechner Internetzugang zu verschaffen.

Mit NAT hat jeder Rechner eine eindeutige (interne) IP Adresse, aber alle teilen sich eine IP Adresse von meinem ISP. Die Kosten sinken.

Setup

Damit NAT auf deinem OpenBSD Rechner läuft, mußt du zunächst PF aktivieren. Dies geschieht einfach, indem du die unten angeführten Dateien editierst (ändere die Dateien gemäß den folgenden Optionen):

/etc/rc.conf (diese Datei wird beim Booten fürs Starten von Diensten gelesen)

```
pf=YES
```

/etc/sysctl.conf

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

Nachdem diese Änderungen durchgeführt wurden, ist der Rechner endlich für die Konfiguration von NAT bereit.

Konfiguration

Der erste Schritt ist die Konfiguration der PF Regeldatei (*/etc/pf.conf*). Dafür werden wir in diesem Dokument jeglichen Datenverkehr durch diese Firewall passieren lassen. Die Datei sollte so aussehen:

```
pass in all
pass out all
```

Siehe wiederum FAQ 6, Packet Filter für weitere Informationen.

Die NAT Konfigurationsdatei (*/etc/nat.conf*) folgt einer sehr einfachen Syntax. Um obige Konfiguration fortzusetzen, sollte die Datei die folgenden Einträge enthalten:

```
nat on fxp0 from 192.168.1.0/24 to any -> 24.5.0.5
```

Hier eine Erklärung für die obigen Zeilen.
"nat"

Das zeigt an, dass du eine NAT Regel aufsetzt.

"fxp0"

Die Netzwerkkarte, die mit dem Internet verbunden ist.

"192.168.1.0/24"

Die IP Adresse und Netzmaske (die Netzmaske ist im CIDR Format). Kombiniert besagen sie, daß "jede IP Adresse von 192.168.1.1 bis 192.168.1.254" umgelegt werden soll.

"24.5.0.5"

Auf diese IP Adresse sollen die LAN IP Adressen umgelegt werden.

Betrieb

Ist die Konfiguration erst einmal komplett, dann gibt es zwei Möglichkeiten, um NAT zu aktivieren. Die erste (und beste), ist, wenn möglich, deinen OpenBSD Rechner zu rebooten. Dies geschieht mit dem Befehl "*reboot*"

Wenn du NAT von der Kommandozeile starten willst, dann benutze die folgenden Befehle:

```
# pfctl -N /etc/nat.conf
# pfctl -e
```

Die erste Zeile dient dazu einen Satz von NAT Regeln in PF zu laden (und jegilche alten Regeln zu löschen), die zweite Zeile schaltet PF ein. Trotzdem ist ein Reboot der bessere Test, um einfach sicherzustellen, dass auch dann alles glatt läuft.

NB: Um die NAT Einstellungen neu zu laden (im Falle, daß du die Datei editiert hast, aber nicht rebooten willst), mußt du nur den ersten Befehl wiederholen. Die Einstellungen werden neu geladen.

6.3.3 NAT Grundwissen

Überprüfen des NAT Status

Um herauszufinden, was NAT tut oder um sicherzustellen, daß die gewünschten Einstellungen auch in Kraft sind, kannst du die "-ss" Option verwenden. Diese Option listet alle Einstellungen und laufende Sitzungen:

```
# pfctl -ss
TCP  192.168.1.40:2132 -> 24.5.0.5:53136 -> 65.42.33.245:22       TIME_WAIT:TIME_WAIT
TCP  192.168.1.40:2492 -> 24.5.0.5:55011 -> 65.42.33.245:22
ESTABLISHED:ESTABLISHED
UDP  192.168.1.40:2491 -> 24.5.0.5:60527 -> 24.2.68.33:53       2:1
```

Erklärungen (nur die erste Zeile, die anderen sind entsprechend):

"192.168.1.40:2132"

Dies zeigt dir die IP Adresse (192.168.1.40) des LAN Rechners, der NAT benutzt. Die Portnummer (2132) der Verbindung wird anschließend gezeigt.

"24.5.0.5:53136"

Dies zeigt, daß die Verbindung via der Adresse 24.5.0.5 und der Portnummer 53136 ins Internet geht.

"65.42.33.245:22"

IP Adresse und Port zu dem verbunden worden ist.

"TIME_WAIT:TIME_WAIT"

Der Status, in dem sich die Verbindung, nach Glauben von PF, befindet.

Probleme mit FTP und NAT

Es gibt ein paar Einschränkungen bei NAT, die am häfigsten anzutreffende ist bei FTP. Man kann FTP in zwei Weisen benutzen: aktiv oder passiv. Von diesen beiden wird passiv allgemein als die sicherere angesehen.

Wenn ein User bei aktivem FTP sich mit einem entfernten FTP-Server verbindet und Informationen oder eine Datei anfordert, sendet der FTP Client dem Server eine zufällige Port-Nummer, auf die sich der FTP-Server beim Client verbindet um dort die Information hinzuschicken. Das ist ein Problem für all die User, die aus dem LAN eine FTP-Verbindung aufbauen wollen. Wenn der FTP-Server seine Daten senden will, verbindet er sich mit einem zufälligen Port auf der externen Netzwerk-Karte. Die NAT-Maschine empfängt die Daten, hat aber kein Mapping dafür, und liefert das Paket nicht aus, sondern verwirft es.

Mit passivem FTP (dem Standard mit dem OpenBSD ftp(1) Client) fragt der Client den Server nach einem zufälligen Port, auf dem der Server in Erwartung der Datenverbindung lauschen wird. Der Server informiert den Client über den ausgewählten Port, und der Client verbindet sich dorthin, um Daten zu übertragen. Dummerweise ist das nicht immer möglich oder wünschenswert. ftp(1) benutzt diesen Modus standardmässig; um aktives FTP zu erzwingen benutze die -A Option von ftp, oder schalte den passiven Modus mit folgendem Befehl am ftp> prompt

ab:

```
passive off
```

Packet Filter bietet eine weitere Lösung für diese Situation, umleiten von FTP Traffic durch einen FTP Proxy-Server, ein Prozess, der deinen FTP-Traffic durch die Filter "leitet". Der FTP-Proxy, der von OpenBSD und PF genutzt wird, ist ftp-proxy(8). Um ihn zu aktivieren, musst du sowas wie das hier in deine */etc/nat.conf* Datei tun:

```
rdr on tl0 proto tcp from any to any port 21 -> 127.0.0.1 port 8021
```

Eine Kurzerklärung dieser Zeile ist: "Verkehr auf dem inneren Interface wird auf den Proxy-Server umgeleitet, der auf dieser Maschine auf dem Port 8081 läuft."

Dazu muss der Proxy-Server natürlich gestartet werden und auf der OpenBSD Maschine laufen, das macht man, indem man die folgende Zeile in die */etc/inetd.conf* einfügt:

```
127.0.0.1:8081 stream tcp nowait root /usr/libexec/ftp-proxy ftp-proxy
```

und entweder das System neu botet, oder dem inetd(8). ein 'HUP' Signal schickt. Ein Weg dazu ist das folgende Kommando:

```
kill -HUP `cat /var/run/inetd.pid`
```

Du musst sicherstellen, dass der FTP-Proxy auf Port 8081 läuft, dem selben Port an den das rdr Statement FTP Traffic sendet. Die Wahl von Port 8081 ist zufällig, obwohl 8081 eine gute Wahl ist, da er nicht für andere Anwndungen belegt ist.

Datenverkehr umleiten

Für manche Anweungen ist es notwendig, ein- oder ausgehenden Verkehr umzuleiten für ein bestimmtes Protokoll und/oder einen bestimmten Port zu einer bestimmten Maschine hinter dem Filtersystem umzuleiten. Ein Beispiel dafür ist ein Rechner im LAN, der einen Webserver beinhaltet, der von aussen erreichbar sein muss. Ein Beispiel wäre ein Webserver in deinem interne Netzwerk, der von aussen erreichbar sein muss, (oder natürlich der bereits erwähnte FTP-Proxy(8)). Eingehende Verbindungen zu deiner gültigen Internetadresse können keine Verbindung herstellen, es sei denn, auf der NAT Box läuft auch ein Webserver. Daher benutzen wir die NAT 'rdr' Direktive in der Regeldatei, um festzulegen, wohin eine bestimmte Verbindung hingeleitet werden soll.

Für unser Beispiel nehmen wir an, daß ein Webserver im LAN mit der IP Adresse 192.168.1.80 existiert. Die NAT Regeldatei benötigt eine neue Direktive. Füge eine Zeile wie folgende in */etc/nat.conf* ein:

```
rdr on fxp0 proto tcp from any to any port 80 -> 192.168.1.80 port 80
```

Die Erklärung:

"rdr"

Diesen Befehl gibst du NAT: NAT soll eine Verbindung umleiten.

"on fxp0"

Die mit dem Internet verbundene Netzwerkkarte.

"from any to any"

Das gibt an, welche IP Addressen umgelitet werden sollen (von jeder, die auf fxp0 hereinkommt, so wie oben beschrieben, zu jeder Ziel-IP)

"port 80"

Der hereinkommende Port (80), der umgeleitet werden soll. Die Zahl "80" muß nicht explizit benutzt werden. Du kannst auch "port www" verwenden. Wenn du den Namen anstelle der Nummer verwendest, dann müssen auch Name und die korrespondierende Zahl in der Datei /etc/services existieren.

"192.168.1.80 port 80"

Die IP Addresse der Maschine im LAN, zu der Pakete umgeleitet werden sollen. Denk daran, dass der Ziel-Port NICHT dem port entsprechen muss, auf dem das Paket hereinkommt. Das folgende ist beispielsweise sowohl korrekt, als auch potentiell nützlich:

```
rdr on fxp0 proto tcp from any to any port 8080 -> 192.168.1.35 port 80
```

Diese Zeile lenkt hereinkommende Verbindungen auf Port 8080 zu einem Webserver auf dessen Port 80 um, der im internen Netzwerk läuft.

Danach lade die NAT Regeln neu und die Umleitungen werden sofort in Kraft treten.

Verneinung

Manchmal muss man Ausnahmen für eine NAT oder Umleitungs-Regel festlegen. Hier ein Beispiel: AOL Instant Messenger ist bekannt dafür , dass er sich über jeden verfügbaren Port durch eine Firewall schleicht. Du kannst entdecken, dass sich der FTP-Proxy nicht mit dem AIM verträgt, sobald der sich den Port 21 aussucht. Wenn du das auch als schlecht betrachtest (viele Leute verbringen jede Menge Zeit damit AIM zu blocken!), kannst du die IP-Adressen der AIM-Server einfach vom Traffic ausnehmen, der von deiner FTP-Proxy Zeile umgeleitet wird. Das geht etwa wie folgt:

```
rdr on tl0 proto tcp from any to ! 64.12.163.199 port 21 -> 127.0.0.1 port 8021
```

Interpretation: Leite eingehenden Traffic auf tl0 Richtung Port 21 um, aber NICHT von 64.12.163.199 (die AIM Server. mit denen die User Probleme hatten) zu localhost port 8081 (wo sinnvollerweiser der FTP-Proxy wartet). Leider gibt es recht viele AIM-Server, wenn dich diese Anwendung interessiert, wirst du ein wenig mit den IP-Adressen herumpobieren müssen (64.12.0.0/16 könnte produktiver sein, aber auch einige Nicht-AOL-Sites treffen).

NAT versus Proxy

Der Unterschied zwischen NAT und einem applikationsbasierten Proxy liegt darin, daß die Proxysoftware als Mittelsmann zwischen Internet und den im LAN verbundenen Rechnern agiert. Nur muß für jede Anwendung und Internetverbindung ein dafür geeigneter Proxy bereitstehen. Nicht alle Anwendungen können dies (vor allem Spiele nicht). NAT "mapped" dein internes Netzwerk transparent, sodass es mit dem Internet verbunden ist. Der einzige Sicherheitsvorteil von Proxysoftware gegenüber NAT ist, daß die Proxysoftware sicherheitstechnisch im Vorteil ist, wenn man den Inhalt filtern kann; z. B. Makroviren für Windowsrechner filtert, oder gegen Buffer-overflows schützt, udgl. Diese Filter zu administrieren bedeutet viel Arbeit.

Redirection und reflection

Oft werden redirection Regeln benutzt, um hereinkommende Verbindungen vom Internet zu einem lokalen Server mit privater Adresse im LAN umzuleiten, wie etwa hier:

```
rdr on $ext_if proto tcp from any to $ext_if port 80 -> $server port 80
```

Wenn aber die redirection Regel von einem Client aus dem LAN getestet wird, bemerkt man, dass sie nicht funktioniert. Der Grund ist, dass redirection Regeln nur für Regeln gelten, die auch über das entsprechende Interface hereinkommen ($ext_if, das externe Interface, in diesem Fall). Sich mit der externen Adresse der Firewall von einem Host aus dem LAN zu verbinden, heisst aber nicht zwangsläufig, dass die Pakete auch durch das externe Interface kommen. Der TCP/IP stack auf der Firewall vergleicht die Zieladresse des einkommenden Paketes mit seiner eigenen Adresse und Aliasen und entdeckt Verbindungen zu sich selbst, sobald sie das interne Interface passiert haben. Solche Pakete gehen physikalisch nicht durch das externe Interface, und der Stack simuliert das auch in keiner Weise. pf sieht diese Pakete nie auf dem externen Interface, und daher greift die redirection Regel auch nicht.

Eine zweite redirection Regel für das interne Interface einzufügen hat nicht fen gewünschten Effekt. Wenn der lokale Client sich zur externen Addresse der Firewall verbindet, kommt das erste Paket des TCP handshake bei der Firewall durch das interne Interface an. Die redirection Regel greift und die Ziel-Adresse wird mit der des internen Servers ersetzt. Das Paket wird über das interne Interface weitergeleitet, und erreich den internen Server. Aber die Quell-Adresse wurde nicht verändert, und enthält somit immernoch die lokale Adresse, somit sendet der Server seine Antwort direkt daran. Die Firewall bekommt die Antwort nie zu Gesicht, und hat daher keine Chance, die Änderung sauber durchzuführen. Der Client hingegen erh&aulm;lt eine Antwort von einer Maschine, die er nie erwartet hat, und verwirft das Paket, der TCP handshake schlägt fehl, und keine Verbindung kann zustande kommen.

Trotzdem ist es oft wüschenswert für die Clients aus dem LAN sich transparent mit dem selben internen Server als externe Clients zu verbinden. Für dieses Problem gibt es mehrere Lösungen.

Split horizon DNS

Es ist möglich, DNS-Server zu konfigurieren, dass die Anfragen lokaler Clients anders beantwortet werden, als die externer Clients, und zwar so, dass die internen Clients auch die interne Adresse des Servers erhalten. Sie verbinden sich direkt mit dem Server, ohne überhaupt mit der Firewall in Berührung zu kommen. Das reduziert den lokalen Traffic, da die Pakete ja gar nicht mehr bis zur Firewall müssen.

Den Server in ein separates lokales Netzwerk stellen

Ein zusätzliches Netzwerk-Interface einzubauen und den lokalen Server in ein eigenes, dafür gedachtes Netzwerk (DMZ) zu stellen, erlaubt das Umleiten der Verbindungen von lokalen Clients in der selben Weise wie das Umleiten externer Verbindungen. Die Benutzung verschiedener Netzwerke hat verschiedene Vorteile, einschliesslich der Verbesserung der Sicherheit durch die Isolierung des Servers von den anderen Maschinen. Sollte der Server (der ja in unserem Fall aus dem Internet erreichbar ist) jemals kompromittiert werden, kann er die lokalen Clients nicht direkt erreichen, da alle Verbindungen durch die Firewall gehen müssen.

TCP proxying

Ein generischer TCP Proxy kann auf der Firewall aufgesetzt werden, der entweder auf dem Port lauscht, der weitergeleitet werden soll oder der die Verbindungen auf dem internen Interface umlenkt. Wenn sich ein lokaler Client mit der Firewall verbindet, akzeptiert der Proxy die Verbindung, öffnet eine zweite Verbindung, und zwar hin zum Server, und leitet dann die Daten zwischen den beiden Verbindungen jeweils weiter.

Einfache Proxies kann man mittels inetd(8) und nc(1) erzeugen. Der folgende */etc/inetd.conf* Eintrag erzeugt einen horchenden Socket, der an die loopback-Adresse und Port 5000 gebunden ist. Die Verbindungen werden nach Port 80 auf dem Server 192.168.1.10 weitergeleitet.

```
127.0.0.1:5000 stream tcp wait nobody /usr/bin/nc nc -w 20 192.168.1.10 80
```

Die folgende redirection Regel leitet von port 80 auf dem internen Interface an den Proxy weiter:

```
rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> 127.0.0.1 port 5000
```

RDR und NAT Kombination

Mit einer zusätzlichen NAT-Regel auf dem internen Interface kann die fehlende Adress-Umsetzung, die oben beschrieben wurde, ergänzt werden.

```
rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> $server
no nat on $int_if proto tcp from $int_if to $int_net
nat on $int_if proto tcp from $int_net to $server port 80 -> $int_if
```

Das führt dazu, dass der erste Paket vom Client erneut übersetzt wird, wenn es durch das interne Interface zurückgeschickt wird, wobei die Quell-Adresse des Clients mit der internen Adresse der Firewall ersetzt wird. Der interne Server wird dann an die Firewall seine Antwort schicken, die dann mittels NAT und RDR wiederum an den Client weiterleitet. Dieses Konstrukt ist recht komplex, das es zwei separate states für jede solche Verbindung erzeugt. Man muss sicherstellen, dass die NAT Regel keinen anderen Teil der Pakete betrifft, zum Beispiel Verbindungen von externen Clients (durch andere Weiterleitungen) oder der Firewall selbst. Bedenke, dass die obige rdr Regel den TCP/IP Stack dazu veranlasst, Pakete, die auf dem inneren Interface ankommen, so zu sehen, dass sie eine Zieladresse aus dem inneren Netzwerk haben. Um zu verhindern, dass der Stack ICMP redirect messages versendet (und damit dem Client erzählt, dass sein Ziel direkt erreichbar ist, und daher die reflection kaputtmacht), schalte redirects auf dem Gateway mittels

```
# sysctl -w net.inet.ip.redirect=0
```

ab. Im allgemeinen sollten man eine der anderen genannten Lösungen vorziehen.

6.3.4 Links und Querverweise

OpenBSD Dateien:

- /etc/nat.conf - NAT Regeldatei
- /etc/rc.conf - muss man editieren, damit NAT und PF beim Booten gestartet werden
- /etc/sysctl.conf - editieren, damit IP Forwarding aktiviert wird

NAT Internet Links:

- nat.conf man page
- pfctl man page
- http://www.geektools.com/rfc/rfc1631.txt

6.4 - DHCP

6.4.1 DHCP Klient

Um den DHCP Klient dhclient(8) zu benutzen, der Teil von OpenBSD ist, editiere /etc/hostname.xl0 (wenn deine Hauptethernetkarte xl0 ist. Deine kann ep0 oder fxp0 oder irgendeine andere sein!). Alles, was du in dieser Datei zu schreiben hast, ist 'dhcp'.

```
# echo xl1 xl2 xl3 >/etc/dhcpd.interfaces
```

Dies wird OpenBSD veranlassen, den DHCP Klient automatisch beim Booten zu starten. OpenBSD wird sich seine IP Adresse, sein Standardgateway und seine DNS Server vom DHCP Server besorgen.

Wenn du den DHCP Klient von der Befehlszeile starten willst, stelle sicher, daß /etc/dhclient.conf existiert, dann versuche:

```
# dhclient fxp0
```

Wobei fxp0 die Netzwerkkarte ist, auf der du DHCP empfangen willst.

Wie du auch immer dhclient startest, du kannst die /etc/dhclient.conf Datei immer so editieren, daß dein DNS **nicht** erneuert wird aufgrund der neuen DNS Informationen, indem du die 'request' Zeilen auskommentierst (Es gibt Beispiele in den Standardeinstellungen, aber du mußt die Standardeinstellungen von dhclient überschreiben.).

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

und dann **entferne** domain-name-servers. Natürlich kannst du auch hostname oder andere Einstellungen entfernen.

6.4.2 DHCP Server

Wenn du OpenBSD als DHCP Server dhcpd(8), einstetzen willst, editiere /etc/rc.conf. Setze dhcpd_flags="-q" anstelle von dhcpd_flags=NO. Und die Netzwerkkarten, auf denen dhcpd(8) **lauschen** soll, stehen in /etc/dhcpd.interfaces.

```
# echo xl1 xl2 xl3 >/etc/dhcpd.interfaces
```

Dann editiere /etc/dhcpd.conf. Die Optionen sind selbsterklärend.

```
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;
}
```

Dies teilt deinen DHCP Klienten mit, daß die an DNS Anfragen anzuhängende Domäne xyz.mil ist (d. h., wenn der Benutzer schreibt 'telnet joe', dann wird an joe.xyz.mil gesendet). Es wird auf die DNS Server 192.168.1.3 und 192.168.1.5 verwiesen. Für Hosts, die sich im selben Netzwerk wie die Netzwerkkarte des OpenBSD Rechners befinden, welche im 192.168.1.0/24 Adressbereich liegt, wird der DHCP Server ihnen eine IP Adresse zwischen 192.168.1.32 und 192.168.1.127 und als Standardgateway 192.168.1.1 zuweisen.

Wenn du den dhcpd(8) von der Befehlszeile starten willst, nachdem du /etc/dhcpd.conf editiert hast, versuche:

```
# dhcpd -q fxp0
```

Wobei `fxp0` die Netzwerkkarte ist, auf der DHCP serviert werden soll. Die `-q` Option setzt die Ausgabe von dhcpd(8) auf ruhig, ansonsten ist sie sehr ausführlich.

Wenn du DHCP Dienste für einen Windows Rechner bereitstellst, dann willst du vielleicht auch eine 'WINS' Serveradresse liefern. Dafür füge einfach die folgenden Zeilen zu deiner /etc/dhcpd.conf:

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

(wobei `192.168.92.55` die IP deines Windows oder Samba Servers ist.) Siehe auch dhcp-options(5) für weitere Optionen, die deine DHCP Klienten wünschen.

6.5 - PPP

Das "Point-to-Protocol" wird verwendet, um eine Verbindung zu deinem ISP mit deinem Modem herzustellen. OpenBSD bietet dafür 2 Möglichkeiten.

- pppd(8) - der Kernel PPP Dämon.
- ppp(8) - der Userland PPP Dämon.

Den ersten, den wir behandeln, wird der Userland PPP Dämon sein. Um zu beginnen, benötigen wir einige einfache Informationen über deinen ISP. Hier eine Liste hilfreicher Informationen, die du brauchen wirst.

- Die Einwahlnummer deines ISP
- Deinen Nameserver
- Deinen Benutzernamen und Password
- Dein Gateway

Einige von diesen benötigst du nicht unbedingt, aber sie wären hilfreich. Der Userland PPP Dämon benutzt die Datei /etc/ppp/ppp.conf als seine Konfigurationsdatei. Es gibt viele hilfreiche Dateien in **/etc/ppp**, die verschiedene Einstellungen für verschiedene Situationen zeigen. Du solltest dir dieses Verzeichnis ansehen und es durchforsten.

Solltest du keinen GENERIC Kernel verwenden, dann stelle sicher, daß du folgende Zeile in deiner Kernelkonfigurationsdatei hast:

```
pseudo-device   tun             2
```

Erste Einstellungen - für PPP(8)

Die ersten Einstellungen für den Userland PPP Dämon bestehen im Erstellen deiner **/etc/ppp/ppp.conf** Datei. Diese Datei existiert nicht standardmäßig, aber du kannst einfach **/etc/ppp/ppp.conf.sample** editieren, um deine eigene **ppp.conf** Datei zu kreieren. Hier werde ich mit dem einfachsten und gebrächlichsten Einstellungen beginnen. Hier eine schnelle **ppp.conf** Datei, die uns einfach zu deinem ISP verbindet und die Standardrouten und Nameserver setzt. Für diese Datei brauchst du nur die Telefonnummer deines ISP sowie deinen Benutzernamen und dein Passwort.

```
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"
```

Der Absatz unter der **default:** Bezeichnung wird jedes Mal ausgeführt. Hier stehen alle wichtigen Informationen. Mit "set log" stellen wir die Loglevel ein. Um dies zu ändern, siehe ppp(8) für weitere Info. Unsere Schnittstelle wird mit "set device" eingestellt. Dies ist die Schnittstelle, mit der das Modem verbunden ist. In diesem Beispiel hängt das Modem auf COM Port 2. Daher wird COM Port 1 auf /dev/cua00 gesetzt. Mit "set speed" setzen wird die Geschwindigkeit unserer Dialup Verbindung und mit "set dial" setzen wir unsere Dialup Parameter, mit denen wir die timeout Zeit, usw. setzen können. Diese Zeile sollte eigentlich ziemlich genau so, wie sie jetzt ist, bleiben.

Nun können wir die ISP spezifischen Informationen eintragen. Wir tun dies, indem wir unter **default:** einen weiteren Absatz hinzufügen. Dieser kann als alles benannt werden, am einfachsten nimmst du den Namen deines ISP. Hier werde ich **myisp:** als Verweis auf unseren ISP

nehmen. Hier ist ein einfaches Beispiel, das alles beinhaltet, um uns zu verbinden.

```
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
```

Hier stehen alle wichtigen Informationen für unseren spezifischen ISP. Die erste Option "set phone" setzt die Telefonnummer deines ISP. "set login" setzt unsere login-Optionen. Hier haben wir die timeout auf 5 gesetzt, was bedeutet, daß wir unseren login-Versuch nach 5 Sekunden abbrechen, wenn wir kein Trägersignal bekommen. Ansonsten wird er auf "login:" warten und dann deinen Benutzernamen und Passwort senden. In diesem Beispiel ist unser username = ppp und das Password = ppp. Diese Werte müssen geändert werden. Die Zeile "set timeout" setzt den Idle timeout für die gesamte Verbindungsdauer auf 120 Sekunden. Die "set ifaddr" Zeile ist ein bißchen schwieriger. Hier ist eine genauere Erklärung.

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

Die obigen Zeile folgt dem Format von "**set ifaddr [meineAdr[/nn] [seineAdr[/nn] [netzmaske [startAdr]]]]**". Daher ist die erste spezifizierte IP diejenige, die wir als unsere IP wollen. Wenn du eine statische IP Adresse hast, dann kannst du sie hier einsetzen. In unserem Beispiel benutzen wir /0, was besagt, daß kein Bits von dieser IP Adresse übereinstimmen muß und der gesamte Ausdruck ersetzt werden kann. Die zweite IP behandelt die von uns erwartete IP unserer Gegenstelle. Wenn du sie weißt, dann kannst du sie hier angeben. Wiederum wissen wir nicht in unserer Zeile, welche IP dies wird, also lassen wir sie um wieder mitteilen. Die dritte Option ist unsere Netzmaske, hier auf 255.255.255.0 gesetzt. Wenn startAdr spezifiziert ist, dann wird diese anstelle von meineAdr während der initialen IPCP Verhandlung; aber es wird nur eine Adresse aus dem meineAdr-Adressbereich akzeptiert.

Die nächste Option "add default HISADDR" setzt unsere Standardroute zu deren IP. Dies ist 'klebrig', d. h falls deren IP sich ändern sollte, dann wird unsere Route auch automatisch upgedatet. Mit "enable dns" teilen wir unserem ISP mit, unsere Nameserveradresse zu authentifizieren. Tu dies NICHT, wenn du deinen eigenen lokalen DNS laufen hast, da PPP dies umgehen wird, indem es einige Zeilen in /etc/resolv.conf schreibt.

PPP(8) verwenden

Nun, da wir unsere **ppp.conf** Datei fertig eingerichtet haben, können wir beginnen, eine Verbindung zu unserem ISP aufzubauen. Hier einige Details über häufig verwendete Parameter.

- **ppp -auto myisp** - Startet PPP, konfiguriert deine Schnittstellen und wird dich mit deinem ISP verbinden und dann in den Hintergrund verschwinden.
- **ppp -ddial myisp** - Ähnlich wie -auto, aber wenn deine Verbindung abbricht, wird PPP versuchen, sich erneut zu verbinden.

Mit dem Aufruf von **/usr/sbin/ppp** ohne Optionen kommst du in den interaktiven Modus. Hier kannst du direkt mit dem Modem interagieren, was sich hervorragend eignet, um Probleme in deiner **ppp.conf** Datei zu debuggen.

ppp(8) Extras

In einigen Situationen möchstest du Befehle ausführen, wenn die Verbindung gerade errichtet oder beendet wurde. Für diese Fälle gibts es zwei Dateien, die du kreieren kannst: **/etc/ppp/ppp.linkup** und **/etc/ppp/ppp.linkdown**. Beispielskonfigurationen kannst du hier finden:

- ppp.linkup
- ppp.linkdown

Weitere Informationen findest du im FreeBSD Handbook entry on User PPP.

6.6 - Netzwerkparameter tunen

6.6.1 - Wie kann ich den Kernel einstellen, damit es eine höhere Anzahl an Verbindungsversuchen und längere Timeouts für TCP Sitzungen gibt?

Du solltest dies normalerweise nur verwenden, wenn du Routing- oder Verbindungsprobleme hast. Natürlich sollten - um die beste Wirkung zu erzielen - beide Seiten der Verbindung dieselben Werte verwenden.

Um dies zu tunen, verwende `sysctl` und erhöhe die Werte von:

```
net.inet.tcp.keepinittime
net.inet.tcp.keepidle
net.inet.tcp.keepintvl
```

Mittels `sysctl -a` kannst du die derzeitigen Werte dieser (und vieler anderer) Parameter sehen. Um einen Wert zu verändern, verwende `sysctl -w`, wie z. B. `sysctl -w net.inet.tcp.keepidle=28800`.

6.6.2 - Wie kann ich "directed broadcasts" aktivieren?

Normalerweise willst du dies nicht tun. Dies erlaubt jemand, Datenverkehr zu der broadcast Adresse deines verbundenen Netzwerkes zu schicken, wenn du deinen OpenBSD Rechner als Router verwendest.

Aber manchmal kann dies (in geschlossenen Netzwerken) nützlich sein, vor allem wenn man ältere Implementierungen des NetBIOS Protokolles verwendet. Wiederum mit sysctl. `sysctl -w net.inet.ip.directed-broadcast=1` aktiviert dies. Beachte aber Smurfangriffe, wenn du wissen willst, warum dies standardmäßig nicht aktiviert ist.

6.6.3 - Der Kernel soll Ports nicht dynamisch allozieren

Auch dafür gibt es einen eigenen sysctl Befehl. Siehe sysctl(8):

```
Setze die Liste der reservierten TCP ports, die nicht dynamisch vom
Kernel vergeben werden sollen. Das kann man benutzen, um daemons davon
abzuhalten, einen speziellen port zu benutzen, den ein anderes Programm
braucht, damit es funktionieren kann. Listen-Elemente können
mit Kommata und/oder Leerzeichen getrennt werden.

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

Es ist ebenso möglich ports aus der aktuellen Liste hinzuzufügen
oder zu entfernen.

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

6.7 - Einfache NFS Anleitung

NFS, oder Network File System (Netzwerkdateisystem), wird verwendet, um ein Dateisystem über das Netzwerk zu verwenden. Du solltest vorher noch folgende Manualseiten lesen, bevor du versuchst, einen eigenen NFS Server aufzusetzen:

- nfsd(8)
- mountd(8)
- exports(5)

Dieses Kapitel zeigt die Schritte, um ein einfaches NFS System aufzusetzen: Ein Server im LAN und Klienten im LAN, die NFS verwenden. Es behandelt nicht, wie man NFS sicher macht. Wir nehmen an, daß du bereits Paketfilterung oder irgendeinen anderen Firewallschutz eingerichtet hast, damit von außerhalb nicht auf NFS zugegriffen werden kann. Wenn du Zugriff via NFS von außerhalb erlauben willst und sensible Daten dort gespeichert hast, dann empfehlen wir dir wärmstens den Gebrauch von IPsec. Ansonsten können andere Leute möglicherweise deinen NFS Datenverkehr sehen. Jemand könnte auch vortäuschen, die IP Adresse zu sein, der du Zugriff auf den NFS Server läßt. Es gibt mehrere Angriffe, die möglich sind. Wenn IPsec richtig konfiguriert ist, dann schützt es gegen die Art von Angriffen.

Noch eine wichtige Anmerkung wegen Sicherheit. Füge niemals ein Dateisystem zu /etc/exports ohne eine Liste mit Rechnern, die explizit Zugriff haben sollen. Ohne einer solchen Liste, die ein bestimmtes Verzeichnis mounten können, kann jeder, der den Rechner erreichen kann, deine NFS exports mounten.

NFS hängt von portmap(8) ab, bevor es funktionieren kann. Portmap(8) ist ab OpenBSD 3.2 standardmässig abgeschaltet, du musst es also in rc.conf(8) wieder einschalten, indem du die `portmap` Zeile wie folgt änderst:

```
portmap=YES
```

und ein Reboot ist notwendig, damit die Änderung wirksam wird.

Der Server hat die IP **10.0.0.1**. Dieser Server soll nur NFS für Rechner innerhalb dieses Netzwerkes bereitstellen. Der erste Schritt ist deine */etc/exports* Datei zu erstellen. Diese Datei listet die Dateisysteme auf, die du über NFS freigeben willst, und definiert, wer auf sie zugreifen darf. Es gibt viele Optionen, die du in deiner */etc/exports* Datei haben kannst, und am besten ist, du liest exports(5) Für dieses Beispiel sieht */etc/exports* so aus:

```
#
# 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
```

D.h., daß das lokale Dateisystem **/work** via NFS zugänglich gemacht wird. **-alldirs** bedeutet, daß Klienten jedes Verzeichnis unter dem **/work** Mount-point mounten können. **-ro** bedeutet, daß nur Leseberechtigung gestattet wird. Die letzten zwei Argumente bedeuten, daß nur Klienten innerhalb des 10.0.0.0 Netzwerkes mit einer Netzmaske von 255.255.255.0 dieses Dateisystem mounten dürfen. Dies ist wichtig für einige Server, die von verschiedenen Netzwerken zugänglich sind.

Ist einmal deine */etc/exports* Datei eingerichtet, kannst du weitergehen und deinen NFS Server aufsetzen. Du solltest zuerst sicherstellen, daß deine Kernelkonfiguration die Optionen NFSSERVER & NFSCLIENT enthält. (Der GENERIC Kernel beinhaltet diese Optionen.) Dann solltest du **nfs_server=YES** in */etc/rc.conf* eintragen. Dies wird sowohl nfsd(8) und mountd(8) starten, wenn du rebootest. Nun kannst du fortschreiten und die Dienste selber starten. Diese Dienste müssen als root gestartet werden und du mußt sicherstellen, daß portmap(8) auf deinem System läuft. Hier ein Beispiel von nfsd(8), der sowohl mit TCP als auch mit UDP bedient mittels 4 Diensten. Du solltest eine angemessenene Anzahl von NFS Serverdiensten einsetzen, um die maximale Anzahl von gleichzeitigen Klientenanfragen, die du bedienen willst, zu bewerkstelligen.

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

Du mußt nicht nur den nfsd(8) Server starten, sondern auch mountd(8). Dies ist der Dienst, der eigentlich die Mountanfragen auf NFS bedient. Um mountd(8) zu starten stelle sicher, dass eien leere mountdtab Datei existiert, und starte den Daemon:

```
# echo -n >/var/db/mountdtab
# /sbin/mountd
```

Wenn du Änderungen an /etc/exports durchführst, während NFS bereits läuft, mußt du mountd dies mitteilen, indem du den Dienst neu startest!

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

NFS Status überprüfen

Um zu überprüfen, ob alle Dienste laufen und bei RPC registriert sind, verwende 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
```

Für den Normalgebrauch gibt es ein paar Hilfsprogramme, mit denen du den Status von NFS überprüfen kannst. Eines ist showmount(8) das dir anzeigt, was und wer gerade mountet. Dann gibt es auch noch nfsstat(8), das genauere Statistiken anzeigt. Für showmount(8), versuche **/usr/bin/showmount -a host**. Z. B.:

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

NFS Dateisysteme mounten

NFS Dateisysteme sollten mittels mount(8) geladen werden, oder genauer mount_nfs(8). Um ein Dateisystem /work von Host 10.0.0.1 auf dem lokalen Dateisystem /mnt zu laden, tue folgendes (NB: du mußt nicht IP Adressen verwenden, mount wird Hostnamen auflösen):

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

Damit dein System dies beim Hochfahren wieder tut, füge folgendes zu deiner */etc/fstab*:

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

Es ist wichtig, daß du `0 0` am Ende dieser Zeile verwendest, damit dein Rechner nicht versucht, das NFS Dateisystem beim Hochfahren mit fsck zu überprüfen!!!! Die anderen Sicherheitsoptionen wie noexec, nodev und nosuid, sollten auch immer - wenn anwendbar - verwendet werden. Z. B.:

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

Mit diesen Optionen können keine Geräte oder setuid Programme auf dem NFS Server Sicherheitsmaßnahmen auf dem NFS Klient untergraben. Wenn du keine Programme auf diesem NFS Dateisystem auf dem NFS Klient ausführen willst, füge noexec hinzu:

6.8 - Domain Name Service - DNS, BIND und named

6.8.1 Was ist DNS?

Domain Name Service bietet die Möglichkeit, Name-zu-IP Adresse Auflösung und IP Adresse-zu-Namen Auflösung auf eine Anfrage zu generieren. Deine OpenBSD Installation ist standardmäßig als DNS Klient, aber nicht als DNS Server konfiguriert. D.h., deine OpenBSD Installation kann eine DNS Anfrage an einen Domain Name Server für die Adresse einer Maschine stellen, aber sie kann nicht selbst solche DNS Anfrage beantworten, bis du dies nicht selbst so konfigurierst.

Meine OpenBSD Maschine ist derzeit mit dem Internet durch meinen ISP verbunden, so daß ich mit nslookup(8) DNS Anfragen ausführen kann:

```
$ 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 ist der Nameserver, der geantwortet hat, weil es der Nameserver ist, den mein ISP mir zu meinem Konto zugeteilt hat und in /etc/resolv.conf eingetragen ist. Aber die Antwort war nicht authoritativ. Für eine authoritative Antwort müssen wir den DNS Server für die *openbsd.org* Domäne finden und ihn nach der Adresse von *www.openbsd.org* fragen:

```
# Identifiziere die Nameserver für openbsd.org
# mit der Hilfe des Nameservers meines ISP.
$ 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 Adresse = 199.185.137.3
gandalf.sigmasoft.com   internet Adresse = 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

# Verwende die gefundenen Informationen, um eine Anfrage
# für eine authoritative Auflösung zu stellen:
# befrage zeus.theos.com.
$ nslookup www.openbsd.org zeus.theos.com
Server:  zeus.theos.com
Adresse:  199.185.137.1

Name:    www.openbsd.org
Address:  129.128.5.191
```

Auf *zeus.theos.com* läft OpenBSD und ist korrekt als DNS server für die *openbsd.org* Domäne konfiguriert.

6.8.1.1 Wo kann ich alles über DNS und seine Implementationen unter OpenBSD lernen?

- Siehe die RFCs 1033, 1034 und 1035 für weitere Informationen über DNS zu erhalten.
- Lies das O'Reilly Associates Buch *DNS and BIND* .
- Lies die OpenBSD Manualseiten vorallem die Seiten von
  - dig(1)
  - nslookup(8)
  - gethostbyname(3)
  - named(8)

- resolver(3)
- resolver(5)

Der dig(1) Befehl ist besonders nützlich, weil er eine Domäne befragen kann und Informationen zurückliefert, die einem Format unterliegen, das BIND Konfigurationsdateien sehr ähnlich ist. Du kannst mit dig(1) Nameserver untersuchen, von denen du weißt, daß sie richtig funktionieren, und sie mit deinen Einstellungen vergleichen.

6.8.2 Muß meine Maschine ein Domain Name Server sein?

Wenn du dir nicht sicher bist, ob dein Rechner die Rolle eines DNS Server spielen soll, dann konfiguriere ihn nicht als solchen. Die OpenBSD Installation konfiguriert nicht standardmäßig deine Maschine als einen Domain Name Server, obwohl alle notwendigen Dateien dafür installiert werden. Für die meisten Arbeitsplatzrechner genügt die /etc/hosts Datei, um IP Adressen lokaler Rechner zu benennen und die /etc/resolv.conf Datei, um die DNS Server im Intranet oder Internet einzustellen.

Aber wenn du vielleicht doch deinen Rechner als Domain Name Server konfigurieren mußt:

- Wenn du ein IP LAN betreibst, für das du nicht auf jedem Rechner die "hosts" Dateien mit den lokalen IP Adressen. In so einem Fall kannst du deinen OpenBSD Rechner als DNS Server konfigurieren und Anfragen der anderen Maschinen aus deinem LAN bedienen.
  - **Anmerkung:** Es gibt keine praktische Einschräkung bzgl. der Anzahl von DNS Servern in einem LAN. Einige oder alle Maschinen im LAN können DNS Dienste anbieten, wenn sie so konfiguriert sind. Ob einige dieser Server als authoritativ von außerhalb deines LANs betrachtet werden (oder sie überhaupt außerhalb deines LANs bekannt sind), ist ein Konfigurationsfaktor, der typischerweise eine Ebene oberhalb deines LANs in der Domänenhierarchie bestimmt wird.
- Wenn du ein IP LAN mit Rechnern hast, die auch via DNS von Rechnern anderer IP LANs und WANs auffindbar sein sollen.
- Wenn du Schwierigkeiten hast, lokale Rechnernamen auf eine IP Adresse aufzulösen oder andere lokale Namen zu IP Adressen, obwohl du korrekte */etc/hosts* und */etc/resolv.conf* Dateien hast (z.B.: Netscape auf OpenBSD hat manchmal dieses Verhalten, weil es seinen eigenen DNS Auflöser verwendet, anstatt einfach *gethostbyname(3)* zu benützen, um IP Adressen nachzusehen.)

Eine weitere Überlegung ist die Ausführungsgeschwindigkeit. Da die Namensauflösung ein iterativer Prozess ist, in dem der Nameserver wiederholende Anfragen an andere Nameserver in entfernten Domänen stellt, kann die Namensauflösung länger dauern, wenn du eine Modemverbindung ins Internet hast und deinen DNS Server nach anderen, entfernten IP Adressen auf der Modemleitung befragst (die ihrerseits wieder andere entfernte DNS Server befragen), als wenn du den Nameserver deines ISP befragst (der wahrscheinlich eine schnellere Verbindung zu entfernten Nameservern hat).

6.8.3 Was sind die Softwarekomponenten der DNS Server?

- named *("name daemon")*
- Konfigurationsdateien in der Verzeichnishierarchie unter */var/named/*

6.8.3.1 Welche Versionen von BIND werden unterstützt?

BIND ist der Name einer Spezifikation eines Domänennamensservers mit einem bestimmten Verhalten. Die DNS Komponenten ergeben gemeinsam die Implementation von BIND.

Es gibt drei getrennte BIND Specifikationen:

1. BIND 4
2. BIND 8
3. BIND 9

Standardmässig unterstützt OpenBSDs **named** BIND 4.x.

6.8.3.2 Welche Alternativen zu der Standard-BIND 4.x-Implementation gibt, um DNS Dienste bereitzustellen?

- Die BIND 9.x Implementation ist in */usr/ports/net/bind9*.(Siehe ports)

6.8.3.2.1 Sicherheitsanmerkung

Wenn du diese alternativen Implementationen von DNS Diensten in betracht ziehst, dann stellst du einen kritischen Netzwerkdienst zur Verfügung, dessen Software nicht dem selben Niveau an Überprüfung wie durch Sicherheit dem **named** name daemon in der Basisinstallation zu Teil wurde. Dies ist eine signifikante Überlegung, da, falls ein DNS Server kompromittiert wird, die Klients zu betrügerischen Webseiten umgeleitet werden können.

6.8.4 Wieviel muß ich installieren?

Wenn die standardmässige Netzwerkinstallation korrekt bei der Installation von OpenBSD eingerichtet hast, dann ist bereits alles installiert. Du mußt nur mehr den Nameserverdienst ("named") konfigurieren.

6.8.5 Wie konfiguriere ich DNS?

Du konfigurierst OpenBSD DNS, indem du Dateien editierst und/oder erstellst, die den Nameserverdienst named steuern. Diese Dateien liegen standardmäßig im Verzeichnis */var/named* und dessen Unterverzeichnisse, hauptsächlich in der Datei */var/named/named.boot*, das die Initialisierungsdatei für **named** ist. Weiterhin gibt es ein paar andere notwendige Konfigurationsschritte in */etc*.

In diesem Dokument werden wir den Nameserverdienst auf *nemo.yewtopia.com* konfigurieren, der der primäre Nameserver für die (sehr kleine!) Domäne *yewtopia.com* sein wird. Die Adresse von *nemo.yewtopia.com* ist *192.168.1.9*. Zwei andere Maschinen befinden sich im selben Subnet, *crater.yewtopia.com* auf 192.168.1.1 und *earhart.yewtopia.com* auf 192.168.1.2.

6.8.5.1 Konfiguration in */var/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
; 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
```

Dies teilt dem Initialisierungsprozeß mit, in welchem Unterverzeichnis und unter welchem Dateinamen die Konfigurationsdateien für *yewtopia.com* zu finden sind.

6.8.5.1.2 */var/named/namedb/localhost.rev*

```
; Reverse lookup für localhost interface
@       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.
1       IN      PTR     localhost.yewtopia.com.
```

6.8.5.1.3 */var/named/namedb/yewtopia*

```
; yewtopia.com domain database
@      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
```

```
@       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 Konfiguration in */etc*

6.8.5.2.1 */etc/resolv.conf*

Stelle sicher, daß */etc/resolv.conf* nun auf die Domäne des lokalen Rechners (anstatt auf, z. B., den Nameserver deines ISPs) zeigt, so daß die Namensauflösungsanfragen auch wirklich zu dem **named** geschickt werden, den du konfiguriert hast!

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

6.8.5.2.2 */etc/hosts*

Wenn du vorher die Adressen von diversen Rechnern zu der */etc/hosts* Datei hinzugefügt hattest, dann solltest du in Betracht ziehen, deine */etc/hosts* Datei wieder auf Standardgröße zu kürzen:

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

Damit **named** nicht zugunsten von (möglicherweise veralteten) Adressen in der */etc/hosts* Datei übergangen wird. Stelle sicher, daß du zumindest den Standardeintrag*localhost* hast, oder dein Netzwerk wird nicht richtig starten!! Auch *nemo* muß in seiner eigenen hosts-Datei aufscheinen, oder du wirst eine (eher harmlosen) Fehlermeldung zu Bootzeit bemerken, wenn */etc/netstart* route(8) aufruft, um *nemo* hinzuzufügen (dessen Name in */etc/myname* aufscheint).

6.8.5.3 Mittels dig(1) die Ergebnisse untersuchen.

```
$ dig @nemo.yewtopia.com yewtopia.com 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.com, type = ANY, class = ANY

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

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

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

6.8.6 Wie kann ich DNS starten und stoppen?

6.8.6.1 DNS starten

Der Nameserverdienst **named** wird beim Systemstart von /etc/rc gestartet, wenn die folgende Zeile (standardmäßig vorhanden) sich in /etc/rc.conf befindet.

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

verändere in

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

Beachte auch diese Zeilen in /etc/rc.conf:

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

Diese Standardeinstellungen werden für beinahe alle Installationen korrekt sein.

Um **named** händisch zu starten, verwende den ndc(8) Befehl. Z. B.:

```
# ndc start
          oder
# ndc reload
```

6.8.6.2 DNS Stoppen

Der beste Weg, um den Nameserverdienst zu stoppen, ist den ndc(8) Befehl zu verwenden. Z. B.:

```
# ndc stop
```

Wenn dies fehlschlägt, finde die Prozeß-ID von named und verwende den kill(1) Befehl, um diesen Prozeß zu beenden. Die PID für **named**, solange er läuft, kannst du in der ersten Zeile der Datei */var/named/named.pid* finden.

```
# 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

6.8.7 How do I block AXFR queries?

Beispiel:

```
garden:/home/jeremy$  host -l openssh.com
openssh.com.            NS      zeus.theos.com.
openssh.com.            NS      cvs.openbsd.org.
openssh.com.            NS      gandalf.sigmasoft.com.
openssh.com.            NS      cs.colorado.edu.
openssh.com.            NS      ns.appli.se.
openssh.com.            A       199.185.137.4
cvs.openssh.com.        A       199.185.137.4
localhost.openssh.com.  A       127.0.0.1
```

Diese Information ist nützlich für das Debuggen von DNS, aber in manchen Fällen willst du diesen Output nicht in aller Welt zeigen. Wenn du klassenloses in-addr(rfc2317) für reverse benutzt, könnte 'host -l' jede Domain anzeigen, die dein System hostet! Dies kann man einfach mit der 'allow-transfer' Formulierung in deinem zone file verhindern.

Wenn du Bind8 benutzt, musst du die Hosts spezifizieren, denen du den "Zonen-Transfer" erlauben willst, und zwar in deinen individuellen Zone Datei(en):

```
zone "foo.com" in {
        type master;
        file "directory/zonefile";
        allow-transfer {
          127.0.0.1;
          10.0.0.6;
```

```
            10.0.255.12;
          };
};
```

Du kannst auch Transfers für alle Domains stoppen, indem du /var/named.conf anpasst und und den 'allow-transfer' Parameter zur options Sektion der Konfigurationsdatei hinzufügst:

```
   options {
        allow-transfer { 127.0.0.1; };
    };
```

Die Bind8 Methode funktioniert auch mit Bind9.
Bei Bind 4 (Standard in OpenBSD) kannst du /var/named/named.boot anpassen und die 'xfrnets' Option nutzen.
xfrnets 209.142.221.5 12.7.96.7 ; type domain source host/file backup file cache . root.cache primary 0.0.127.IN-ADDR.ARPA localhost.rev
Bind 4 erlaubt Transfers von ganzen Klassen, ist also nicht so exakt. Typischerweise sind die einzigen Hosts, die Transfers durchführen müssen deine DNS Slaves und Hosts von denen du vielleicht 'debug'en willst (127.0.0.1 ist meist ein guter Host, dem man Transfers erlauben sollte!) AXFR queries zu blocken fügt einen zusätzlichen Level an Privatsphere ein, kann aber ein sinnvolles DNS debugging behindern. (Danke an Nicholas Tang für diesen Tip)

6.8.8 Was hast du mir nicht über das Aufsetzen von DNS erzählt?

Es gibt eine Menge von Dingen, die ich dir nicht erzählt habe, z.B. wie man DNS so aufsetzt, dass Anfragen für Intranet Domains, die von der root der Domain-Hierarchie aus nicht sichtbar sind, zu Servern innerhalb deines Unternehmens weitergeleitet werden. Lies die empfohlenen Dokumente, um mehr Informationen über DNS zu erhalten.

6.9 - Eine PPTP Verbindung mit OpenBSD aufsetzen

HINWEIS: Dies bezieht sich nicht auf **ALLE** ADSL Provider, aber viele der Informationen können aus diesem Setup übernommen werden. Dieses Setup funktioniert auf jeden Fall bei Inode, einem ADSL Provider in Österreich.

Als Anfang musst du pptp installiert haben. Der Port befindet sich unter */usr/ports/net/pptp*. Lies FAQ 8, Ports um mehr Informationen über den OpenBSD ports tree zu bekommen.

Wegen des Konflikts der "Im-Kernel" gre(4) Unterstützung und pptp wirst du deinen Kernel neu kompilieren müssen und die Unterstützung für gre(4) entfernen müssen.

Patch, um die GRE(4) Unterstützung zu entfernen.

```
Index: GENERIC
===================================================================
RCS file: /cvs/src/sys/conf/GENERIC,v
retrieving revision 1.86
diff -u -r1.86 GENERIC
--- GENERIC     14 Mar 2002 00:42:25 -0000      1.86
+++ GENERIC     17 May 2002 01:52:17 -0000
@@ -87,7 +87,7 @@
 pseudo-device  enc     1       # option IPSEC needs the encapsulation interface
 pseudo-device  bridge  2       # network bridging support
 pseudo-device  vlan    2       # IEEE 802.1Q VLAN
-pseudo-device  gre     1       # GRE encapsulation interface
+#pseudo-device gre     1       # GRE encapsulation interface
 #pseudo-device strip   1       # Starmode Radio IP interface

 pseudo-device  pty     64      # pseudo-terminals
```

Um deinen Kernel neu zu kompilieren mache einen "check out" von OpenBSD via cvs (siehe die OpenBSD Stable Webseite) , benutze den folgenden Patch, und baue einen neuen Kernel wie unter FAQ 5, Building a kernel.

Nachdem du das **pptp** package und einen neuen Kernel installiert hast, musst du ein paar Dateien für deine neue Verbindung editieren. Diese packages benutzen das standarmässige OpenBSD ppp(8) wenn du dich also mit ppp(8) auskennst, kommt dir vieles bekannt vor. Siehe auch FAQ 6, Setup.

- 1 - /etc/ppp/options
- 2 - /etc/ppp/pap-secrets

Für eine */etc/ppp/options* Datei wird ein Setup wie das unten vermutlich alles notwendige tun:

```
# cat /etc/ppp/options
name "LOGINNAME"
noauth
```

```
noipdefault
defaultroute
debug
```

`LOGINNAME` sollte mit deiner User-ID ersetzt werden.

In */etc/ppp/pap-secrets* gehört eine Zeile wie diese hier:

```
# cat /etc/ppp/pap-secrets
LOGINNAME 10.0.0.138 PASSWORD
```

Wobei LOGINNAME deine User-ID und PASSWORD dein Password ist. 10.0.0.138 ist die zugewiesene IP deines Modems im Falle, dass du ADSL nutzt, etc. Stelle sicher, dass diese Datei nur von root gelesen werden kann (mode 600).

6.9.1 - Deinem Network Interface eine Adresse zuweisen

Im obigen Beispiel hatte unser Modem eine vorkonfigurierte Adresse von 10.0.0.138. Jetzt müssen wir UNSEREM Interface noch eine Adresse zuweisen. Es ist am besten eine IP zu wählen, die nahe an der deines MODEMS liegt, oder einfach die statische Adresse zu benutzen, die dir zugewiesen wurde. Mehr darüber, wie man Interfaces IP-Adressen zuweist, kannst du in FAQ 6.1 lesen.

Wenn dein Interface eingerichtet ist, solltest du eine pptp Verbindung mit dem Kommando

```
# /usr/local/sbin/pptp 10.0.0.138 &
```

aufbauen können.

Da hier auch der "in-house" OpenBSD ppp(8) benutzt wird, werden hier zwei Prozesse gestartet. Du kannst pptp beenden, indem du diesen beiden Prozesse killst:

```
# kill -9 [pid of pppd]
$ kill -9 [pid of pptp]
```

Wir empfehlen `/var/log/messages` in einem weiteren Terminalfenster zu öffnen, um mögliche Probleme zu erkennen.

```
# tail -f /var/log/messages
```

Wir schlagen vor, die Startsequenz in */etc/rc.local* unterzubringen, so dass bei jedem reboot die Verbindung automatisch aufgebaut wird.

6.10 - Aufsetzen einer Bridge mit OpenBSD

Eine Bridge ist ein Link zwischen zwei oder mehr separaten Netzwerken. Anders als ein Router reisen Pakete durch die Bridge "unsichtbar" -- logisch erscheinen die beiden Netzwerksegmente als eines für Rechner auf beiden Seiten der Bridge. Die Bridge wird nur Pakete weiterleiten, die auch von einem Segment in das andere müssen, sie bieten also auch einen einfachen Weg den Traffic in einem komplexen Netzwerk zu reduzieren und erlauben trotzdem den Zugriff jedes Rechners zu jedem anderen, falls nötig.

Denk daran, dass aufgrund dieser "unsichtbaren" Natur ein Interface in einer Bridge eine IP-Adresse haben kann, aber nicht muss. Wenn sie eine hat, hat die Karte effektiv zwei Betriebsmodi, nämlich eine als Teil der Bridge, die andere als normale, stand-alone Netzwerk-Karte. Wenn keine der Karten eine IP-Adresse hat, wird die Bridge einfach Netz-Daten verschieben, aber nicht von extern administrierbar oder wartbar sein (was auch ein Feature sein kann).

Ein Beispiel einer Bridge Anwendung

Eines meiner Computer Racks hat eine Anzahl alter Systeme, von denen keines eine eingebaute 10BASE-TX Netzwerk-Karte hat. Während sie alle einen AUI oder AAUI Stecker haben, sind die Transceiver auf Koax beschränkt. Eine der Maschinen in diesem Rack ist ein OpenBSD-basierender Terminal-Server, der dauerhaft eingeschaltet und auch immer mit einem High-Speed-Netzwerk verbunden ist. Das Hinzufügen einer zweiten Netzwerk-Karte mit einem Koax-Port erlaubt mir, diese Maschine als Bridge in das Koax-Netzwerk zu benutzen.

Dieses System hat jetzt zwei Netzwerk-Karten (NICs), eine Intel EtherExpress/100 (`fxp0`) und eine 3c590-Combo Karte (`ep0`) für den Koax Port. `fxp0` ist der Link in mein restliches Netzwerk und wird daher eine IP-Adresse haben, `ep0` macht nur Bridging und hat daher keine. Maschinen, die an das Koax-Segment angeschlossen sind, sollen genauso kommunizieren, als wenn sie im Rest meines Netzwerkes wären. Wie also bewerkstelligen wir das ?

Die Datei `hostname.fxp0` enthält die Konfigurationsdaten für die `fxp0` Karte. Diese Maschine soll DHCP machen, also sieht die Datei etwa so aus:

```
$ cat /etc/hostname.fxp0
dhcp NONE NONE NONE NONE
```

Noch keinerlei Überraschungen

Die `ep0` Karte ist ein wenig anders, wie du dir denken kannst:

```
$ cat /etc/hostname.ep0
up media 10base2
```

Hier sagen wir dem System, es möge das Interface mittels ifconfig(8) aktivieren und auf 10BASE-2 (Koax) setzen. Keine IP Addresse oder ähnliche Information muss für dieses Interface spezifiziert werden. Die Optionen, die von der `ep` Karte akzeptiert werden, sind detailliert in der man page aufgeführt.

Jetzt müsen wir die Bridge aufsetzen. Bridges werden durch die Existenz einer Datei namens `bridgename.bridge0`. initialisiert. Hier ist zum Beispiel ein Datei für meine Situation:

```
$ cat /etc/bridgename.bridge0
add fxp0
add ep0
up
```

Das sagt aus, es soll eine Bridge aus zwei NICs aufgesetzt und aktiviert werden, fxp0 und ep0. Es ist egal, in welche Reihenfolge die Karten aufgeführt werden. Denke daran, die Bridge ist symmetrisch -- Pakete fliessen ja in beide Richtungen.

Das war es! Reboote, und du wirst eine funktionierende Bridge haben.

Filtern auf der Bridge

Während es sicher auch eine Menge Anwendungen für eine solch einfache Bridge gibt, ist es doch wahrscheinlich, dass du etwas mit den ganzen Paketen TUN willst, während sie durch deine Bridge laufen. Wie zu erwarten, kann man Packet Filter dazu benutzen, den Traffic einzuschränken, der durch deine Bridge fliesst.

Denke daran, dass wegen der Natur der Bridge die gleichen Daten über beide Interfaces fliessen, aber du nur auf einem Interface zu filtern brauchst. Deine "Pass all" Statements würden dann wie folgt aussehen:

```
pass in  on ep0  any
pass out on ep0  any
pass in  on fxp0 any
pass out on fxp0 any
```

Sagen wir nun, ich wollte den Traffic filtern, der diesen alten Maschinen trifft. Ich möchte, dass nur Web und SSH-Traffic zu Ihnen durchkommt. In diesem Fall lassen wir jeglichen Traffic nach draussen zu, filtern aber auf dem fxp0 Interface, indem wir keep state für die Antwort-Daten benutzen:

```
# Pass all traffic through ep0
pass in quick on ep0 all
pass out quick on ep0 all

# Block fxp0 traffic
block in  on fxp0 all
block out on fxp0 all

pass in quick on fxp0 proto tcp from any to any port {22, 80} \
     flags S/SA keep state
```

Denke daran, dass diese Regelwerk jeglichen Netzwerk-Verkehr mit Ausnahme von hereinkommendem HTTP und SSH-Traffic zur Bridge selbst und den Maschinen "dahinter" verhindert. Andere Resultate werden erzielt, wenn man auf dem anderen Interface filtert.

Um die Bridge zu überwachen und zu kontrollieren, benutze das brconfig(8) Kommando, mit dem man eine Bridge auch nach dem Booten erzeugen kann.

Tips zum Bridging

- Es wird WÄRMSTENS empfohlen, nur auf einem Interface zu filtern. Wenn es auch möglich ist, auf beiden zu filtern, muss man das vorher jedoch sehr gut verstanden haben.
- Durch die Benutzung der *blocknonip* Option von brconfig(8) oder in bridgename.bridge0, kannst du jeglichen nicht-IP Traffic (wie etwa IPX oder NETBEUI) davon abhalten, sich um deine Filter herumzustehlen. Das kann in einigen Situationen sehr wichtig sein, aber du solltest wissen, dass Bridges für jeglichen Traffic funktionieren, nicht nur für IP.
- Für Bridging müssen die NICs im "Promiscuous mode" sein -- sie lauschen einfach am GESAMTEN Netzwerk-Verkehr, nicht nur an dem, der an das Interface gerichtet ist. Das hat einen höheren Load für CPU und Bus zur Folge, als man denkt. Einige NICs funktionieren leider nicht sauber in diesem Modus, der TI ThunderLAN Chip (tl(4)) ist leider so ein Beispiel, der nicht als Teil einer Bridge funktioniert.

[FAQ Index] [Zu Kapitel 5 - Kernelkonfiguration] [Zu Kapitel 7 - Tastatureinstellungen]


www@openbsd.org

Originally [OpenBSD: faq6.html,v 1.168]
$Translation: faq6.html,v 1.70 2003/04/09 18:38:29 jufi Exp $
$OpenBSD: faq6.html,v 1.54 2003/04/27 11:40:31 jufi Exp $

7 - Tastatur- und Display-Kontrollen

Inhaltsverzeichnis

7.1 - Wie verändere ich die Belegung der Tastatur? *(wscons)*

Die ports, die den wscons(4) Console Treiber benutzen: alpha, i386, und macppc.

Bei wscons(4) Konsolen werden die meisten Optionen mit dem wsconsctl(8) Utility kontrolliert. Beispielsweise würde man die Tastaturbelegung mit Hilfe von wsconsctl(8) wie folgt ändern:

```
# wsconsctl -w keyboard.encoding=de
```

Im nächsten Beispiel werden wir die "Caps Lock" Taste neu belegen, und zwar so, dass sie nun "Control L" ist:

```
# wsconsctl -w keyboard.map+="keysym Caps_Lock = Control_L"
```

7.2 - Gibt es so etwas wie gpm für OpenBSD?

Für die Plattformen the alpha und i386 bietet OpenBSD den wsmoused(8), einen port des moused(8) von FreeBSD. Er kann beim jedem Booten automatisch aktiviert werden, indem du die passende Zeile in rc.conf(8) editierst.

7.3 - Wie lösche ich die Konsole, sobald sich ein User ausloggt?

Dazu musst du eine Zeile in /etc/gettytab(5) einfügen. Ändere die Sektion, die so aussieht:

```
P|Pc|Pc console:\
```

```
        :np:sp#9600:
```

Füge die Zeile "`:cl=\E[H\E[2J:`" am Ende hinzu, so dass das ganze schlussendlich so aussieht:

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

7.4 - Auf den Konsolen 'Scrollback Puffer' zugreifen *(alpha/macppc/i386)*

OpenBSD bietet einen console scrollback buffer. Das macht es möglich Informationen zu sehen, die bereits wieder oben aus dem Bildschirm gescrollt sind. Um sich auf- und abwärst zu bewegen, benutze einfach die Tastenkombinationen `[SHIFT]+[PGUP]` und `[SHIFT]+[PGDN]`

Der Standard-Scrollback-Puffer und somit auch die Anzahl der Seiten, die du so betrachten kannst, beträgt 8.

7.5 - Wie wechsele ich die Konsolen?*(i386 spezifisch)*

Auf dem i386 bietet OpenBSD sechs virtuelle Terminal an, /dev/ttyC0 bis /dev/ttyC5. ttyC4 ist fü die Benutzung durch das X Window System reserviert, was noch fünf Text-Konsolen übrgi lässt. Zwischen ihnen hin- und herspringen kannst du, indem du Tastaturkombinationen `[CTRL]+[ALT]+[F1]`, `[CTRL]+[ALT]+[F2]`, `[CTRL]+[ALT]+[F3]`, `[CTRL]+[ALT]+[F4]` und `[CTRL]+[ALT]+[F6]` benutzt.

Die X Umgebung benutzt ttyC4, `[CTRL]+[ALT]+[F5]`. Wenn du X benutzt, bringen dich die `[CTRL]+[ALT]+[F`*n*`]` Tasten auf die Text-Konsolen, `[CTRL]+[ALT]+[F5]` bringt dich wieder auf die grafische Öberfläche. Wenn du mehr als die standardmässige Anzahl von Text-Konsolen haben willst, kannst du den wsconscfg(8) Befehl benutzen, um Bildschirme für ttyC6, ttyC7 und weitere zu erzeugen. Zum Beispiel:

```
wsconscfg -t 80x25 6
```

erzeugt ein virtuelles Terminal für ttyC6, was man mit `[CTRL]+[ALT]+[F7]` erreicht. Vergiss nicht, diesen Befehl deiner `rc.local(8)` Datei hinzuzufügen, wenn der zusätzliche Bildschirm auch nach dem nächsten Reboot noch vorhanden sein soll.

Denk daran, dass du keinen "`login:`" Prompt auf der neu erzeugten virtuellen Konsole erhälst, bis du sie in `/etc/ttys(5)`, auf "on" setzt, und entweder mit init(8) rebootest, oder ein HUP Signal mittels kill(1) sendest.

7.6 - Wie bekomme ich eine Konsolen-Auflösung von 80x50? *(i386)*

i386 User haben normalerweise eine Konsole mit 25 Zeilen und jede hat 80 Buchstaben. Viele VGA Grafik-Karten sind aber in der Lage, eine höhere Textauflösung von 50 Zeilen mit 80 Buchstaben darzustellen.

Zunächst einmal kann man einen Font, der die gewünschte Auflösung unterstützt, mittels des wsfontload Befehls laden. Der Standard-80x25-Text-Bildschirm benutzt 8x16 pixel Fonts, um die vertikale Auflösung zu verdoppeln, müssen wir 8x8 pixel Fonts benutzen.

Danach müssen wir die virtuelle Konsole löschen und in der gewünschten Auflösung neu erzeugen, und zwar mit dem wsconscfg Befehl.

Das kann während des Bootens automatisch am Ende deiner `rc.local` Datei geschehen:

```
wsfontload -h 8 -e ibm /usr/share/misc/pcvtfonts/vt220l.808
wsconscfg -dF 5
wsconscfg -t 80x50 5
```

Wie bei jeder Änderung deiner Systemkonfiguration solltest du etwas Zeit mit den passenden man pages verbringen, um zu verstehen, was diese Befehle eigentlich tun.

Die erste Zeile dort oben lädt den 8x8 Font. Die zweite Zeile löscht den virtuellen Bildschirm Nummer 5 (den man mit `[CTRL]-[ALT]-[F6]` erreichen würde). Die dritte Zeile erzeugt einen neuen Bildschirm 5 mit der Auflösung 50 Zeilen mit je 80 Buchstaben. Wenn du alles so gemacht hast, werden deine anderen virtuellen Bildschirme ganz normal im 80x25 Modus erscheinen, und der neue Bildschirm 5 mit 80x25 ist mit `[CTRL]-[ALT]-[F6]` erreichbar.

Denke daran, dass `[CTRL]-[ALT]-[F1]` hier Bildschirm 0 ist. Wenn du die anderen Schirme auch ändern willst, wiederhole einfach die Schritte "delete" und "add screen" für jeden Bildschirm, den du mit 80x50 betreiben willst.

Du solltest aber Bildschirm 4 (ttyC4, `[CTRL]-[ALT]-[F5]`) unverändert lassen, da dieser von X als grafischer Schirm genutzt wird.

Natürlich können all diese Befehle auch als root an der Konsole eingegeben werden, oder besser mittels sudo(8).

Hinweis: Das funktioniert nicht mit allen Grafik-Karten. Dummerweise unterstützen nicht alle Grafik-Karten die von wscons benötigten Fonts für den 80x50 Text-Modus. In diesen Fällen solltest du über den Einsatz von X nachdenken.

7.7 - Wie kann ich eine serielle Konsole benutzen ?

Es gibt viele Gründe, warum du auf deinem OpenBSD System eine serielle Konsole benutzen willst:

- Aufnehmen der Ausgabe der Konsole (zur Dokumentation).
- Remote Management.
- Einfachere Wartung einer grossen Anzahl von Maschinen
- Ein sauberes dmesg von Maschinen bekommen, von denen das sonst nicht so einfach ist.
- Eine saubere "trace" und "ps" Ausgabe bieten, wenn dein System gecrasht ist, so dass die Entwickler eine Chance haben, das Problem zu beheben.

OpenBSD unterstützt auf den meisten Plattformen eine serielle Konsole, die Details weichen aber stark ab.

Bedenke, dass serielles Arbeiten/Interfacing KEINE einfache Sache ist -- du wirst oft ungewöhnliche Kabel benötigen und ports sind oft nicht zwischen den Maschinen standardisiert, und manchmal noch nicht mal auf einer einzelnen Maschine konsistent. Ein komplettes Tutorial über serielles Arbeiten/Interfacing geht weit über die Möglichkeiten dieses Artikels hinaus, geben wir die trotzdem einen guten Rat: Nur weil der Stecker hineinpasst, heisst das noch lange nicht, dass es auch funktioniert.

/etc/ttys Änderung

Es gibt zwei Punkte, um eine funktionierende serielle Konsole unter OpenBSD zu erhalten. Erstens musst du OpenBSD haben, um deinen seriellen Port als Konsole für Status-Ausgaben und single User-Modus benutzen zu können. Zweitens muss dein serieller Port eingeschaltet sein, um als interaktives Terminal benutzbar zu sein, so dass ein User sich einloggen kann, wenn die Maschine im Multi-User-Betrieb läuft. Dieser Teil ist zwischen den verschiedenen Plattformen relativ gleich und wird hier besprochen.

Terminal Sessions werden von der Datei */etc/ttys* kontrolliert. Bevor OpenBSD dir auf der seriellen Konsole einen `"login:"` Prompt serviert, musst du das Ganze in */etc/ttys* einschalten, normalerweise gibt es ja andere Verwendungen für die serielle Schnittstelle. Bei Plattformen, die normalerweise eine angeschlossene Tastatur und Bildschirm haben, ist die serielle Konsole standardmässig abgeschaltet. Wir benutzen hier die i386-Plattform als Beispiel. In diesem Fall musst du die Zeile editieren, die wie folgt aussieht:

```
tty00   "/usr/libexec/getty std.9600"   unknown off
```

Ist zu ändern in:

```
tty00   "/usr/libexec/getty std.9600"   vt100   on secure
```

`tty00` ist hier der serielle Port, den wir als Konsole nutzen wollen. Das "`on`" aktiviert das *getty* für den seriellen port, so dass "`login:`" Prompt präsentiert wird, das "`secure`" erlaubt ein root (uid 0) Login auf dieser Konsole (das kann man wollen, oder auch nicht) und das "`9600`" ist die Baud-Rate des Terminals. Denke draran, dass du die serielle Konsole auch ohne diese Schritte für Installationen benutzen kannst, da das System im Singel-User-Modus läuft, und *getty* gar nicht erst für den Login benutzt wird.

Auf einigen Plattformen und einigen Konfigurationen ist es notwenidg das System im Single-User-Modus zu booten, damit die Änderungen durchgeführt werden können.

i386

Um den Boot-Prozess dazu zu bewegen, den seriellen Port als Konsole zu verwenden, erzeuge oder editiere deine */etc/boot.conf* Datei, damit sie folgende Zeile enthält:

```
set tty com0
```

angenommen, du möchtest den erste seriellen Port als Konsole verwenden. Die Standard-Baud-Rate ist 9600bps, das kann mittels der stty Option in */etc/boot.conf* angepasst werden. Diese Datei wird auf deinem Boot-Laufwerk abgelegt, was auch deine Installations-Floppy sein kann, oder das Kommando kann ebenso gut am `boot>` Prompt vom OpenBSD second-stage boot loader eingegeben werden, wenn es nur für ein einzelnes Mail (oder zum ersten Mal) genutzt werden soll.

i386 Hinweise:

- OpenBSD zählt die seriellen Ports beginned mit *tty00*, DOS/Windows mit *COM1*. Denke als daran, dass *tty02 COM3* ist, und nicht *COM2*
- Einige Systeme können sogar ohne eingesteckte Grafikkarte funktionieren, aber sicher nicht alle -- viele Systeme sehen das als Fehler an. Manche Systeme verweigern sogar ohne eingesteckte Tastatur den Dienst.
- Einige i386 Systeme sind in der Lage, den seriellen Port als das Console Device zu benutzen. Deine Ergebnisse könen hiervon abweichen -- dieser Port ist nach dem Booten vielleicht nicht dem Betriebssystem zugänglich, und es kann daher notwendig sein, ZWEI serielle Ports zu verwenden.
- PC kompatible Computer sind nicht entworfen worden, um von einer seriellen Konsole aus bedient zu werden, im Gegensatz zu einigen anderen Plattformen. Sogar die Systeme, die eine seriellen Konsole unterstützen haben dafür im Normalfall eine Option im BIOS - und sollte diese Information verloren gehen, wird das System wieder nach normal angeschlossener Tastatur und Monitor suchen. Du solltest immer einen Monitor und eine Tastatur griffbereit haben, um im Notfall darauf zurückgreifen zu können.
- Du wirst */etc/ttys* wie oben anpassen müssen.

SPARC und UltraSPARC

Diese Maschinen sind so entworfen worden, dass sie vollständig über eine serielle Konsole gewartet werden können. Entferne einfach die Tastatur von der Maschine, und das System läuft über die serielle Konsole.

SPARC und UltraSPARC Hinweise

- Die zwei seriellen ports auf SPARC heissen *ttya* und *ttyb*.
- Anders als auf anderen Plattformen, ist es nicht notwendig irgendwelche Änderungen an */etc/ttys* zu machen, um die serielle Konsole benutzen zu können.
- Die SPARC/UltraSPARC Systeme interpretieren ein BREAK Signal auf dem Konsolen-Port als das selbe

wie ein STOP-A Kommando, und bringt das System zurück zum Forth Prompt, und hält jede Anwendung und das Betriebssystem an diesem Punkt an. Das ist recht nützlich, wenn man es braucht, aber unglücklicherweise senden einige serielle Terminals beim Power-Down und einige RS-232 Switche etwas, was der Computer als Break-Signal interpretiert und halten damit die Maschine an. Überprüfe das, bevor du damit in einer Produktionsumgebung arbeitest.

- Wenn du Tastatur und Monitor angeschlossen hast, kannst du trotzdem die serielle Konsole benutzen, indem du die folgenden Befehle am `ok` Prompt eingibst:

```
ok setenv input-device ttya
ok setenv output-device ttya
ok reset
```

  Wenn Tastatur und Monitor (ttyC0) in */etc/ttys* (siehe oben) aktiv sind, kannst du sie zusätzlich benutzen, obwohl einige Kombinationen von SPARC Hardware dann mit der Tastatur Probleme bereiten können, mit OpenBSD 3.3 sollte das aber behoben sein.

MacPPC

Die MacPPC Maschinen sind durch OpenFirmware für die Benutzung mittels serieller Konsole vorbereitet. Benutze die folgenden Befehle:

```
ok setenv output-device scca
ok setenv input-device scca
ok reset-all
```

Setze deine serielle Konsole auf 57600bps, 8N1.

MacPPC Hinweise

- Unglücklicherweise ist die serielle Konsole auf den meisten MacPPCs nicht direkt erreichbar. Während die meisten dieser Maschinen zwar serielle Hardware haben, ist sie nicht von aussen zugänglich. Einige Firmen bieten jedoch Zusatz-Hardware für verschiedene Macintosh-Modelle, so dass dann der Port als serielle Konsole (oder anderweitig) nutzbar ist. Verwende die Suchmaschine deines Vertrauens und suche nach etwas wie "Macintosh internal serial port".
- Du wirst `tty00` in */etc/ttys* auf `on` ändern müssen und die Geschwidigkeit auf 57600 anstelle der standardmässigen 9600 setzen müssen, wie oben beschrieben, und zwar im Single-User-Modus, bevor du eine funktionierende serielle Konsole hast.

Mac68k

Die serielle Konsole wird im *Booter* Programm ausgewählt, unter dem "`Options`" Pull-Down Menü, dann "`Serial Ports`". Prüfe den "Serial Console" Knopf, dann wähle den Modem oder Drucker Port. Du wirst ein Macintosh Modem oder Drucker-Kabel an den seriellen Port des Mac anschliessen müssen. Wenn das in Zukunft deine Standardeinstellung sein soll, musst du dem Booter-Programm sagen, es soll deine Optionen speichern.

Mac68k Hinweise

- Der Modem Port ist *tty00*, der Drucker Port ist *tty01*.
- Der Mac68k schaltet seinen seriellen Port nicht ein, bis man ihn danach fragt, also wird deine breakout box keinerlei Signale anzeigen, bis der OpenBSD Boot Prozess gestartet ist.
- Du wirst den Port in (*tty00* oder *tty01*) wie oben beschrieben einschalten müssen.

7.8 - Wie schalte ich meinen Konsolen-Bildschirmschoner ein ? (wscons)

Wenn du möchtest, dass sich deine Konsole nach einer gewissen Zeit der Inaktivität von X abschaltet, kannst du die folgenden wscons(4) Variablen ändern:

- **`display.vblank`** auf `on` gesetzt schaltet das vertical sync Signal ein, was einige Monitore dazu veranlasst, in den "energy saver" Modus zu gehen. Es wird hinterher mehr Zeit beanspruchen, den Monitor wieder in den Normalmodus zu bringen, aber es reduziert den Energieverbrauch und die Hintzentwicklung neuerer Monitore. Im Modus `off` wird zwar der Bildschirm schwarz, aber der Monitor empfängt weiter das honrizontale und vertikale sync Signal, und die Rückkehr zum Normalbetrieb geschieht augenblicklich.
- **`display.screen_off`** gibt die Verzögerungszeit im tausendstel Sekunden an, also wäre 60000 eine Verzögerung von einer Minute.
- **`display.kbdact`** gibt an, ob Aktivitäten auf der Tastatur wieder zur Rückkehr in den Normalmodus führen. Normalerweise will man das.
- **`display.outact`** gibt an, ob Ausgaben auf dem Bildschirm wieder zur Rückkehr in den Normalmodus führen.

Du kannst diese Variablen auf der Kommandozeile mittels des wsconsctl(8) Befehls setzen:

```
# wsconsctl -w display.screen_off=60000
display.screen_off -> 60000
```

oder sie dauerhaft setzen, indem du sie in der Datei */etc/wsconsctl.conf* einträgst, so dass sie beim nächsten Bootvorgang aktiviert werden:

```
display.vblank=on               # enable vertical sync blank
display.screen_off=600000       # set screen blank timeout to 10 minutes
display.kbdact=on               # Restore screen on keyboard input
display.outact=off              # Restore screen on display output
```

Der Bildschirmschoner wird entweder aktiv, wenn `display.kbdact` oder `display.outact` auf "`on`" gesetzt sind.

[FAQ Index] [Zum Kapitel 6 - Netzwerken] [Zum Kapitel 8 - Allgemeine Fragen]

---

www@openbsd.org

Originally [OpenBSD: faq7.html,v 1.51]
$Translation: faq7.html,v 1.32 2003/05/01 15:04:47 jufi Exp $
$OpenBSD: faq7.html,v 1.29 2003/05/01 15:38:38 jufi Exp $

8 - General Questions

Table of Contents

8.2 - How do I change virtual terminals?

See this article.

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.
   # **fsck -p / && mount -uw /**
3. If /usr is not the same partition that / is (and it shouldn't be) then you will need to mount it, also
   # **fsck -p /usr && mount /usr**
4. run passwd(1)
5. boot into multiuser 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:

```
option APERTURE
```

is in your kernel configuration. [It is already in the GENERIC kernel]

Then you need to edit */etc/sysctl.conf* and set **machdep.allowaperture=2**. This will allow X to access the aperture driver. This would already be set if you said that you would be running X when asked during the install. 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 Online Documentation at http://www.xfree86.org/support.html.

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

CVS is the tool that 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.cvshome.org/. 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

You can retrieve the sources from one of the OpenBSD AnonCVS servers. These servers are listed on http://www.openbsd.org/anoncvs.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 FAQ 8, Ports of the OpenBSD FAQ.
- *XF4* - The XF4 module contains the source to compile XFree 4.

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 ssh(1), but some AnonCVS servers allow for the use of rsh. For those of you behind a firewall there are also the options of pserver and some AnonCVS servers run ssh on port 2022. Be sure to check http://www.openbsd.org/anoncvs.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.

```
$ export CVSROOT=anoncvs@anoncvs.usa.openbsd.org:/cvs
$ 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 environment 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:

```
$ cvs -d anoncvs@anoncvs.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 then others, so again check http://www.openbsd.org/anoncvs.html. In this example I will be updating my www module from anoncvs.usa.openbsd.org. Notice the -q option that I use, this makes the output not so verbose

coming from the server.

```
$ echo $CVSROOT
anoncvs@anoncvs.usa.openbsd.org:/cvs
$ cvs -q up -Pd 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/3.3/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 FAQ 8, Packages.

Important note about keeping your system and ports in sync

OpenBSD has three "active" versions at any point in time:

- Release: What is on the CD.
- Stable: Release, plus security and reliability enhancements
- Current: The development version of OpenBSD.

DO NOT mix versions of Ports and OpenBSD!

If your system is Release, use the Release version of the ports tree. Don't try to use a -Current version of the Ports tree on a -Release or -Stable system. Not only is it not likely to work, you will irritate people when you ask for help about why "nothing seems to work!" Note that there is a -Stable branch of the Ports tree as well, where critical fixes to -Release ports will be made.

Yes, this really does mean a wonderful new port will not typically work on your "older" system -- even if that system was -current just a few weeks ago.

If you do not have the ports tree installed, you can download it via any of OpenBSD's FTP servers, or of course, from the CDROM. The file is ports.tar.gz, and you want to untar this in the /usr directory, which will create /usr/ports, and all the directories under it. For example:

```
$ ftp ftp://ftp.openbsd.org/pub/OpenBSD/3.3/ports.tar.gz
$ sudo cp ports.tar.gz /usr
$ cd /usr; sudo tar xzf ports.tar.gz
```

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. If you are installing a snapshot of OpenBSD, you should use a matching snapshot of ports. Again, make sure you keep your ports tree and your OpenBSD system in sync.

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

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':

```
$ make search key="samba"
  [...snip...]
Port:   amanda-client-2.4.2.2
Path:   misc/amanda,-client
Info:   network-capable tape backup (client only)
Maint:  Tom Schutter
Index:  misc
```

```
L-deps:
B-deps: :devel/gmake gnuplot-*:math/gnuplot gtar-*:archivers/gtar
samba-*:net/samba/stable
R-deps:
Archs:  any

Port:   samba-2.2.8a
Path:   net/samba/stable
Info:   SMB and CIFS client and server for UNIX
Maint:  The OpenBSD ports mailing-list
Index:  net
L-deps: popt::devel/popt
B-deps: :devel/autoconf/2.13 :devel/metaauto
R-deps:
Archs:  any
  [...snip...]
```

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.

```
$ sudo make install
===>  Checking files for xfig-3.2.4
>> xfig.3.2.4.full.tar.gz doesn't seem to exist on this system.
>> Attempting to fetch /usr/ports/distfiles/xfig.3.2.4.full.tar.gz from http://w
ww.xfig.org/xfigdist/.
100% |**************************************************|  5042 KB    00:31
>> Checksum OK for xfig.3.2.4.full.tar.gz. (sha1)
===>  xfig-3.2.4 depends on: jpeg.62 - jpeg.62 missing...
===>  Verifying install for jpeg.62 in graphics/jpeg
===>  Checking files for jpeg-6b
>> jpegsrc.v6b.tar.gz doesn't seem to exist on this system.
>> Attempting to fetch /usr/ports/distfiles/jpegsrc.v6b.tar.gz from ftp://ftp.uu
.net/graphics/jpeg/.
'EPSV': command not understood.
100% |**************************************************|   598 KB    00:06
>> Checksum OK for jpegsrc.v6b.tar.gz. (sha1)
===>  Extracting for jpeg-6b
===>  Patching for jpeg-6b
===>  Configuring for jpeg-6b
checking for gcc... cc
checking whether the C compiler (cc -O2   ) works... yes
checking whether the C compiler (cc -O2   ) is a cross-compiler... no
checking whether we are using GNU C... yes
  [...snip...]
```

Using Flavors

Many of the applications in the ports tree support different install options, called *flavors*. If a port comes in multiple flavors, you can use these options simply by setting an environment variable before you compile the port. If multiple features are needed, the FLAVOR variable can be set to a space-delimited list of the supported and desired flavors. Currently, many ports have flavors that include database support, support for systems without X, or network additions like SSL and IPv6.

```
$ pwd
/usr/ports/net/mtr
$ make show=FLAVORS
no_x11
$ env FLAVOR="no_x11" make
===>  mtr-0.49-no_x11 depends on: gmake-3.80 - not found
```

```
===>  Verifying install for gmake-3.80 in devel/gmake
===>  Checking files for gmake-3.80
>> make-3.80.tar.gz doesn't seem to exist on this system.
>> Attempting to fetch /usr/ports/distfiles/make-3.80.tar.gz from ftp://ftp.gnu.
org/gnu/make/.
Unknown command.
100% |***********************************************|  1183 KB    00:07
>> Checksum OK for make-3.80.tar.gz. (sha1)
  [...snip...]
$ sudo env FLAVOR="no_x11" make install
===>  Faking installation for mtr-0.49-no_x11
  [...snip...]
===>  Building package for mtr-0.49-no_x11
Creating package /usr/ports/packages/i386/All/mtr-0.49-no_x11.tgz
Using SrcDir value of /usr/ports/net/mtr/w-mtr-0.49-no_x11/fake-i386-no_x11/usr/
local
Creating gzip'd tar ball in '/usr/ports/packages/i386/All/mtr-0.49-no_x11.tgz'
===>  Installing mtr-0.49-no_x11 from /usr/ports/packages/i386/All/mtr-0.49-no_a
x11.tgz
```

Listing Installed ports/packages

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

```
$ /usr/sbin/pkg_info
zsh-4.0.6            The Z shell.
screen-3.9.13        A multi-screen window manager.
emacs-21.2           GNU editing macros.
tcsh-6.12.00         An extended C-shell with many useful features.
bash-2.05b           The GNU Borne Again Shell.
zip-2.3              Create/update ZIP files compatible with pkzip.
ircII-20030314       An enhanced version of ircII, the Internet Relay Chat client
ispell-3.2.06        An interactive spelling checker.
tin-1.4.6            TIN newsreader (termcap based)
procmail-3.22        A local mail delivery agent.
strobe-1.06          Fast scatter/gather TCP port scanner
lsof-4.67            Lists information about open files.
ntp-4.1.74           Network Time Protocol Implementation.
ncftp-3.1.5p0        ftp replacement with advanced user interface
nmh-1.0.4p1          The New MH mail handling program
bzip2-1.0.2          A block-sorting file compressor
```

Other Information

More information about the ports can be found in the ports(7) man page and on the Ports page.

Our ports tree is constantly being expanded, and if you would like to help please see: http://www.openbsd.org/porting.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 CDs 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.org/ftp.html. Packages are located at */pub/OpenBSD/3.3/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-3.9.13 will be installed.

Installing via local disk

```
$ sudo pkg_add -v screen-3.9.13.tgz
Requested space: 749864 bytes, free space: 2239117312 bytes in
/var/tmp/instmp.cpsHA27596
Running install with PRE-INSTALL for `screen-3.9.13'
extract: Package name is screen-3.9.13
extract: CWD to /usr/local
extract: /usr/local/bin/screen-3.9.13
extract: execute 'ln -sf screen-3.9.13 /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 .
Running mtree for `screen-3.9.13'
mtree -q -U -f +MTREE_DIRS -d -e -p /usr/local
Running install with POST-INSTALL for `screen-3.9.13'

+---------------
| 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-3.9.13' registered in `/var/db/pkg/screen-3.9.13'
```

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/3.3/packages/i386/screen-3.9.13.tgz
>>> ftp -o - ftp://ftp.openbsd.org/pub/OpenBSD/3.3/packages/i386/screen-3.9.13.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
mpg123-0.59r       mpeg audio 1/2 layer 1, 2 and 3 player
nmap-3.00          port scanning large networks
ircII-20030314     enhanced version of ircII (internet relay chat)
screen-3.9.13      multi-screen window manager
unzip-5.50         extract, list & test files in a ZIP archive
ntp-4.1.74         Network Time Protocol implementation
icb-5.0.9p1        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-3.9.13

+---------------
| To completely deinstall the screen-3.9.13 package you need to perform
| this step as root:
|
|           rm -f /etc/screenrc
|
| Do not do this if you plan on re-installing screen-3.9.13
| 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 if that fails, it will try **/bsd.old**. 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.

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

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 in 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. The following algorithms are available:

- md4
- md5
- sha1
- rmd160.

Setting up S/Key - The first steps

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

```
# mkdir /etc/skey
```

Once that directory 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 S/Key *secret password*, and is **NOT** your system password. Your secret password must be at least 10 characters. We suggest using a memorable phrase containing several words as the secret password. Here is an example user being added.

```
$ 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. Starting with OpenBSD 3.0, S/Key logins work differently. For OpenBSD 3.0 and above you append **:skey** to your login name. For versions of OpenBSD previous to 3.0 you use **s/key** for the password at which time you are prompted for your S/Key

password (the exception to this is ftpd(8) which will always present an S/Key challenge for S/Key-enabled user prior to OpenBSD < 3.0). The examples below assume OpenBSD 3.0 or higher.

```
$ ftp localhost
Connected to localhost.
220 oshibana.shin.ms FTP server (Version 6.5/OpenBSD) ready.
Name (localhost:ericj): ericj:skey
331- otp-md5 96 oshi45820
331 S/Key Password:
230- OpenBSD 3.3 (GENERIC) #44: Sat Mar 29 13:22:05 MST 2003
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.
```

Note that I appended ":skey" to my username. This tells ftpd that I want to authenticate using S/Key. Some of you might have noticed that my sequence number has changed to *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).

```
$ skeyinfo
95 oshi45820
```

An even better way is to use **skeyinfo -v**, which outputs a command suitable to be run in the shell. For instance:

```
$ skeyinfo -v
otp-md5 95 oshi45820
```

Not only is *otp-md5* a description of the hash used, it is also an alternate name for the skey(1) command. So, the simplest way to generate the next S/Key password is simply:

```
$ `skeyinfo -v`
Reminder - Do not use this program while logged in via telnet or rlogin.
Enter secret password:
NOOK CHUB HOYT SAC DOLE FUME
```

Note the backticks in the above example.

I'm sure many of you won't always have a secure connection 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.

```
$ otp-md5 -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
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--you simply tack ":skey" to the end of your username. Example:

```
$ telnet localhost
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

OpenBSD/i386 (oshibana) (ttyp2)

login: ericj:skey
otp-md5 98 oshi45821
S/Key Password: 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 3.3 (GENERIC) #44: Sat Mar 29 13:22:05 MST 2003

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.
$
```

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 ntpd (found in the ports collection) and just deal with the occasional lossage. Sometimes the lossage is so bad that even ntpd 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 - Does OpenBSD support SMP? (Symmetric Multi-Processor)

Not at this time. Work is on-going, there is an OpenBSD SMP project, though there is nothing usable yet, nor is there any time schedule for OpenBSD SMP support.

On most platforms, OpenBSD will run on an SMP system, but only utilizing one processor. The exception to this is the SPARC platform -- OpenBSD/sparc will not run on some multi-processor SPARC systems at all (General rule: one MBus module always works. Multiple MBus modules will work as long as they are single processor modules and supported, of course). Multi-processor SPARC64 systems run as long as the base machine is supported.

How can I help?

- Write code. There is lots of work to do.
- Donate hardware. The goal of the OpenBSD SMP project is to support as many platforms as possible, including SPARC,

SPARC64, Alpha, MacPPC and of course, the i386. Donations of desired hardware are welcome!

- Don't offer to "test". The SMP project is not part of the core OpenBSD project right now for a reason -- it isn't ready for production use yet, or even general "testing". Informing the developers that it didn't work on your hardware without code to make it work isn't useful at this point. Send in your dmesg, if a developer sees something interesting in your hardware, they can contact you for testing.

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

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 - What web browsers are available for OpenBSD?

Lynx, a text-based browser, is in the base system, and has SSL support. Other browsers in the ports tree, include (in no particular order):

Graphical (X) Browsers

- Dillo Minimal feature set, very fast and small, runs well on slower hardware.
- Konqueror Installed as part of the KDE desktop environment.
- Konqueror-embedded (konq-e) Konqueror, using only the KDE libraries rather than all of KDE.
- Netscape 4 For sparc and i386 only, not Open Source, no package available.
- Opera Commercial browser.
- Amaya The W3C's browser and editor.
- Links+ Another fast and small graphical browser. (Also has a text-only mode)

Console (Text mode) Browsers

- w3m Has table and frame support (also has a graphical mode).
- links Has table support.

Mozilla does not currently work on OpenBSD, either natively or in emulation. There has been some success running Mozilla as a statically-compiled application (which makes it much larger and slower to load). See the mailing list archives for details.

You will find all these in the ports collection. All the above mentioned browsers are located in `/usr/ports/www/` after the installation of the ports tree. Most are also available as pre-compiled packages, available on the FTP servers and on the CDROM. As most of the graphical browsers are very large and require quite some time to download and compile, one should seriously consider the use of packages where available.

8.15 - How do I use the mg editor?

Mg is a micro Emacs-style text editor included in OpenBSD. 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 Window System

  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, add the line *XTerm*loginShell: true* to the file *.Xdefaults* in your home directory. Note, this file does not exist by default, you may have to create it.

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

  You may not have had to do this on other systems, as some installations of X Window System come with this setting as default. OpenBSD has chosen to follow the XFree86 behavior.

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

The */etc/motd* file is edited upon every boot of the system, replacing everything up to, but not including, the first blank line with the system's kernel version information. When editing this file, make sure that you start after this blank line, 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.

8.19 - I'm having problems with my PCI devices being detected

There exists a condition where some machines might not detect some PCI devices properly, or might freeze while detecting multiple NIC's in one machine. This is the fault of PCIBIOS, and involves a simple workaround to make it work properly. Simply enter the boot time configuration and disable PCIBIOS. An example is below:

```
boot> boot -c
Copyright (c) 1982, 1986, 1989, 1991, 1993
        The Regents of the University of California.  All rights reserved.
Copyright (c) 1995-2002 OpenBSD. All rights reserved.  http://www.OpenBSD.org

OpenBSD 3.3 (GENERIC) #44: Sat Mar 29 13:22:05 MST 2003
    deraadt@i386.openbsd.org:/usr/src/sys/arch/i386/compile/RAMDISK_CD
cpu0: Intel Pentium III (Coppermine) ("GenuineIntel" 686-class, 128KB L2 cache)
1 GHz
cpu0: FPU,V86,DE,PSE,TSC,MSR,PAE,MCE,CX8,SYS,MTRR,PGE,MCA,CMOV,PAT,PSE36,MMX,FXS
R,SIMD
real mem  = 267956224 (261676K)
avail mem = 243347456 (237644K)
using 3296 buffers containing 13500416 bytes (13184K) of memory
User Kernel Config
UKC> disable pcibios
UKC> quit
[... snip ...]
```

Once this is done, you can follow the directions in FAQ 5, Building a Kernel to create a new kernel so that you don't have to worry about this in the future.

8.20 - Antialiased and TrueType fonts in XFree86

See this document.

8.21 - Does OpenBSD support any journaling filesystems?

No it doesn't. We use a different mechanism to achieve similar results that is called Soft Updates. Please read in FAQ 14 to get more details.

8.22 - Reverse DNS

- or -

Why is it taking so long for me to log in?

Many new users to OpenBSD experience a two minute login delay when using services such as ssh, ftp, or telnet. This can also be experienced when using a proxy, such as ftp-proxy, or when sending mail out from a workstation through sendmail.

This is almost always due to a reverse-DNS problem. DNS is Domain Name Services, the system the Internet uses to convert a name, such as "www.openbsd.org" into a numeric IP address. Another task of DNS is the ability to take a numeric address and convert it back to a "name", this is "Reverse DNS".

In order to provide better logging, OpenBSD performs a reverse-DNS lookup on any machine that attaches to it in many different ways, including ssh, ftp, telnet, sendmail or ftp-proxy. Unfortunately, in some cases, the machine that is making the connection does not have a proper reverse DNS entry.

An example of this situation:

A user sets up an OpenBSD box as a firewall and gateway to their internal home network, mapping all their internal computers to one external IP using NAT. They may also use it as an outbound mail relay. They follow the installation guidelines, and are very happy with the results, except for one thing -- every time they try to attach to the box in any way, they end up with a two minute delay before things happen.

What is going on:

From a workstation behind the NAT of the gateway with an unregistered IP address of 192.168.1.35, the user uses ssh to access the gateway system. The ssh client prompts for username and password, and sends them to the gateway box. The gateway then tries to figure out who is trying to log in by performing a reverse DNS lookup of 192.168.1.35. The problem is 192.168.0.0 addresses are for private use, so a properly configured DNS server outside your network knows it should have no information about those addresses. Some will quickly return an error message, in these cases, OpenBSD will assume there is no more information to be gained, and it will quickly give up and just admit the user. Other DNS servers will not return ANY response. In this case you will find yourself waiting for the OpenBSD name resolver to time out, which takes about two minutes before the login will be permitted to continue. In the case of ftp-proxy, some ftp clients will timeout before the reverse DNS query times out, leading to the impression that ftp-proxy isn't working.

This can be quite annoying. Fortunately, it is an easy thing to fix.

Fix, using `/etc/hosts`:

The simplest fix is to populate your `/etc/hosts` file with all the workstations you have in your internal network, and ensure that your `/etc/resolv.conf` file contains the line `lookup file bind` which ensures that the resolver knows to start with the `/etc/hosts` file, and failing that, to use the DNS servers specified by the "nameserver" lines in your `/etc/resolv.conf` file.

Your `/etc/hosts` file will look something like this:

```
::1 localhost.in.example.org localhost
127.0.0.1 localhost.in.example.org localhost
192.168.1.1 gw.in.example.org gw
192.168.1.20 scrappy.in.example.org scrappy
192.168.1.35 shadow.in.example.org shadow
```

Your `resolv.conf` file will look something like this:

```
search in.example.org
nameserver 24.2.68.33
nameserver 24.2.68.34
lookup file bind
```

A common objection to this is "But, I use DHCP for my internal network! How can I configure my `/etc/hosts`?" Rather easily, actually. Just enter lines for all the addresses your DHCP server is going to give out, plus any static devices:

```
::1 localhost.in.example.org localhost
127.0.0.1 localhost.in.example.org localhost
192.168.1.1 gw.in.example.org gw
192.168.1.20 scrappy.in.example.org scrappy
192.168.1.35 shadow.in.example.org shadow
192.168.1.100 d100.in.example.org d100
192.168.1.101 d101.in.example.org d101
192.168.1.102 d102.in.example.org d102
        [... snip ...]
192.168.1.198 d198.in.example.org d198
192.168.1.199 d199.in.example.org d199
```

In this case, I am assuming you have the DHCP range set to 192.168.1.100 through 192.168.1.199, plus the three static definitions as listed at the top of the file.

If your gateway must use DHCP for configuration, you may well find you have a problem -- dhclient will overwrite your `/etc/resolv.conf` every time the lease is renewed, which will remove the "lookup file bind" line. This can be solved by putting the line "lookup file bind" in the file `/etc/resolv.conf.tail`.

Fix, using a local DNS server

Details on this are somewhat beyond the scope of this document, but the basic trick is to setup your favorite DNS server, and make sure it knows it is authoritative for both forward and reverse DNS resolution for all nodes in your network, and make sure your computers (including your gateway) know to use it as a DNS server.

8.23 - Why do the OpenBSD web pages not conform to HTML4/XHTML?

The present web pages have been carefully crafted to work on a wide variety of actual browsers going back to browser versions 4.0 and later. We do not want to make these older pages conform to HTML4 or XHTML until we're sure that they will also work with older browsers; it's just not a priority. We welcome new contributors, but suggest you work on writing code, or on documenting new aspects of the system, not on tweaking the existing web pages to conform to newer standards.

8.24 - Why is my clock off by twenty-some seconds?

When using rdate(8) to synchronize your clock to a NTP server, you may find your clock is off by twenty-some seconds from your local definition of time.

This is caused by a difference between the UTC (Coordinated Universal Time, based on astronomical observations) time and TAI (International Atomic Time, based on atomic clocks) time. To compensate for variations in the earth's rotation, "leap seconds" are inserted into UTC, but TAI is unadjusted. These leap seconds are the cause of this discrepancy. For a more detailed description, search the web for "`leap seconds UTC TAI`".

Addressing the problem is fairly simple. In most countries you will get the correct time if you use the "`-c`" parameter to rdate(8) and use a time zone out of the directory `/usr/share/zoneinfo/right/`. For example, if you are located in Germany, you could use these commands:

```
# cd /etc && ln -sf /usr/share/zoneinfo/right/CET localtime
# rdate -ncv ptbtime1.ptb.de
```

In other countries, the rules may differ.

[FAQ Index] [To Section 7 - Keyboard and Display Controls] [To Section 9 - Tips for Linux users]


www@openbsd.org
$OpenBSD: faq8.html,v 1.123 2003/05/01 01:47:41 nick Exp $

9 - Umstieg von Linux

Inhaltsverzeichnis

Weitere Informationen für Linux Benutzer gibt es unter http://sites.inka.de/mips/unix/bsdlinux.html.

9.1 - Einfache Tips für Linux (und andere freie Unix-artige BS) User

Es gibt einige Unterschiede zwischen OpenBSD und Linux. Diese Unterschiede sind unter anderem die Bootup-Sequenz, Netzwerkkarten Benutzung und das Festplatten- Management. Die meisten Unterschiede sind gut dokumentiert, aber benötigen ein Suchen in den man-pages. Dieses Dokument versucht als Index dieser Unterschiede zu dienen.

- OpenBSD hat einen ports tree. Das wirkt der Tatsache entgegen, dass bis jetzt nicht allzuviele Anwendungen rein für die OpenBSD-Umgebung geschrieben wurden. Dies ist sowohl ein Versuch mehr Anwendungen für OpenBSD verfügbar zu machen, als auch darauf hinzuwirken, dass mehr OpenBSD-kompatible Anwendungen entwickelt werden. Eventuell wird dieser ports tree dazu verwendet, eine nette Zusammenstellung von Binärpaketen zu machen.
- OpenBSD benutzt CVS für Source-Änderungen. Bei Linux ist der Source-Code zwischen verschiedenen Distributionen verteilt und zum Teil unterschiedlich. OpenBSD hat das 'anonymous CVS' zuerst eingeführt, was jedermann erlaubt, den vollen 'source tree' für jede Version von OpenBSD herunterzuladen, (von 2.0 bis -current, und alle Zwischenversionen von allen Dateien zwischendurch) und das zu jeder Zeit! Es gibt auch ein sehr bequemes und einfach zu bedienendes Webinterface zu CVS.
- OpenBSD gibt regelmässig 'snapshots' für verschiedene Architekturen heraus und alle 6 Monate gibt es eine stabile offizielle Version auf CD.
- OpenBSD enthält STARKE KRYPTOGRAFIE, die Betriebssysteme aus den USA nicht enthalten können. (Siehe http://www.openbsd.org/de/crypto.html) OpenBSD wurde auch einer starken Sicherheitsüberprüfung unterzogen und viele Sicherheitsfeatures wurden bereits in den Source tree eingefügt. (IPSEC, KERBEROS).
- Der Kernel von OpenBSD's heisst /bsd.
- Die Namen von Festplatten sind für gewöhnlich */dev/wd* (IDE) und */dev/sd* (SCSI oder ATA Geräte, die SCSI Festplatten emulieren)
- /sbin/ifconfig ohne Argumente ergibt unter Linux den Status aller Netzwerkinterfaces. Unter OpenBSD benötigst du das -a Flag.
- /sbin/route ergibt unter Linux den Status aller aktiven Routen. Unter OpenBSD brauchst du den "*show*" Parameter, oder nimm einfach **netstat -r**.
- OpenBSD unterstützt KEINE Journaling Filesysteme wie ReiserFS, IBMs JFS oder SGIs XFS. Stattdessen benutzen wir Soft Updates.
- OpenBSD enthält das Paket Filter Paket (pf), nicht ipfw, ipchains, netfilter, iptables, oder IP Filter. Das bedeutet:
  - Network Adress Translation (=NAT, in Linux unter IP-Masquerading bekannt) wird durch pfctl realisiert

(mittels -N). (pfctl(8))
    - ipfwadm wird durch pfctl realisiert. (pfctl(8), pf(4), pf.conf(5))
    - Du solltest dir Kapitel 6 der FAQ für detailliertere Konfigurationshilfen und Informationen ansehen.
- Die Interface-Adresse wird in /etc/hostname.<interfacename> aufbewahrt. Sie kann auch ein Hostname anstatt einer IP Addresse sein.
- Der Name deiner Maschine ist in */etc/myname*
- Das Standard-Gateway ist in */etc/mygate*
- OpenBSD's Standard-Shell ist /bin/sh, also die Korn shell. Shells wie bash und tcsh können als Pakete hinzugefügt oder aus dem ports tree installiert werden.
- Das Passwort-Management ist deutlich verändert. Die Hauptdateien sind verschieden. (passwd(1), passwd(5))
- Devices werden nach dem Treiber benannt und nicht nach ihrem Typ. Zum Beispiel gibt es keine eth* Devices. Aber es gibt ne0 für eine NE2000 Ethernet-Karte, und xl0 für 3Com Etherlink XL und Fast Etherlink XL Ethernet Device, etc.
- Die OpenBSD Entwickler haben grosse Anstrengungen unternommen, um die 'manual pages' auf den aktuellen Stand zu bringen und aufzuräumen. Unter man(1) gibt es weitere Informationen dazu.

9.2 - Dual boot mit Linux und OpenBSD

Ja! Es ist machbar!

Lies INSTALL.linux

9.3 - Deine Linux (oder andere System-7-artige) Passwort-Datei in BSD-Style konvertieren.

Zuerst finde heraus, ob deine Linux Passwortdatei 'geshadowed' wird oder nicht. Wenn ja, besorge dir John the Ripper und benutze das 'unshadow' Werkzeug, das darin enthalten ist, um deine `passwd` und `shadow` Dateien in eine System 7-artige Datei zu verwandeln..

In deiner Linux Passwort-Datei, wir nennen sie `linux_passwd`, musst du nun ::0:0 zwischen den Feldern 4 und 7 einfügen. Awk kann das für dich erledigen.

```
# 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
```

An diesem Punkt an:195 gelangt, solltest du jetzt die `new_passwd` Datei ändern und root und andere Systemeinträge löschen, die bereits in deiner OpenBSD Passwortdatei enthalten sind, oder die es in OpenBSD gar nicht gibt (und zwar alle davon). Stelle auch sicher, dass es keine doppelten Usernamen oder User IDs zwischen `new_passwd` und dem `/etc/passwd` auf deiner OpenBSD-Kiste gibt. Der einfachste Weg ist, einfach mit einer frischen `/etc/passwd` anzufangen.

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

Der letzte Schritt, `pwd_mkdb` ist notwendig, um die /etc/spwd.db und /etc/pwd.db Dateien neu zu erzeugen. Es erzeugt auch eine System 7-artige Passwortdatei (ohne verschlüsselte Passwörter) unter /etc/passwd für die Programme, die darauf zugreifen. OpenBSD benutzt eine stärkere Verschlüsselung für Passwörter, nämlich 'blowfish', die man wohl kaum auf Systemen mit vollen System7-artigen Passwort-Dateien finden wird. Um zu dieser stärkeren Verschlüsselung zu wechseln, müssen die User einfach 'passwd' benutzen (bzw. tippen) und ihr Passwort ändern. Das neu eingegebene Passwort wird mit deiner Standardeinstelung verschlüsselt (normalerweise 'blowfish', es sei denn, du hättest /etc/login.conf verändert). Oder du machst es als root mit `passwd` *`username`*.

9.4 - Wie man OpenBSD und Linux zur Zusammenarbeit bewegt

Wenn du von Linux zu OpenBSD wechselst, denk daran, dass im OpenBSD GENERIC Kernel die Option COMPAT_LINUX standardmässig aktiviert ist. Um Linux Binaries zum Laufen zu bringen, die nicht statisch gelinkt sind (und die meisten sind es nicht), solltest du den Anweisungen auf der compat_linux(8) manual page folgen. Ein einfacher Weg, die meisten nützlichen Linux Libraries zu kriegen, ist, den `redhat/base` port aus der 'ports collection' zu installieren. Um mehr über die 'ports collection' zu erfahren, lies einfach FAQ 8 - Ports. Wenn du den Ports tree schon installiert hast, benutze diese Kommandos, um die Linux Libraries zu installieren:

```
# cd /usr/ports/emulators/redhat/base
# make install
```

OpenBSD unterstützt das EXT2FS Dateisystem. Benutze **disklabel (8)** ***disk*** (wobei ***disk*** der 'device name' für deine Festplatte ist.) um zu überprüfen, was OpenBSD denkt, was deine Linux Partition ist (aber benutze disklabel oder fdisk **nicht** um daran Änderungen vorzunehmen). Für weitere Informationen über die Benutzung von disklabel lies FAQ 14 - Disklabel.

[FAQ Index] [Zum Kapitel 8 - Allgemeine Fragen] [Zum Kapitel 10 - System Administration]

---

www@openbsd.org

Originally [OpenBSD: faq9.html,v 1.49]
$Translation: faq9.html,v 1.29 2003/05/01 12:35:29 jufi Exp $
$OpenBSD: faq9.html,v 1.30 2003/05/01 12:48:46 jufi Exp $

10 - System Management

Table of Contents

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 - )
```

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/rc.conf.local - Configuration file you can use to override settings in /etc/rc.conf so you don't have to touch /etc/rc.conf itself, which is convenient for people who upgrade often.
- /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* (or */etc/rc.conf.local*), */etc/rc.local* and */etc/rc.shutdown*. To get a look of how the rc(8) procedure works, here is the flow:

After the kernel is booted, */etc/rc* is started:

- Filesystems are checked. This will 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* and, afterwards, */etc/rc.conf.local*. Settings in rc.conf.local will override those in 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
- Performs various other checks (quotas, savecore, etc)
- Local daemons are run, via */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 to start 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 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"".

A good approach is to never touch */etc/rc.conf* itself. Instead, create the file */etc/rc.conf.local*, copy just the lines you are about to change from */etc/rc.conf* and adjust them as you like. This may make future upgrading easier -- all the changes are in the one file.

Starting up local daemons and configuration

For other daemons that you might install with the system via ports or other ways, you should use the */etc/rc.local* file. For example, I've installed a daemon which lies 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
```

(If the daemon does not automatically detach on startup, remember to add a "&" at the end of the command line.)

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 at 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 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 `head -1 /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 writable 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,...)! There are several POP servers you can install right away from the ports collection. If possible, use popa3d which is available in the OpenBSD base install. 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 valuable for the POP daemon.)

PS: Notice, OpenBSD does not have a group name of "mail". You need to create this in your */etc/group* file if you need it. An entry like:

```
mail:*:6:
```

would be sufficient.

10.6 - Why does Sendmail ignore `/etc/hosts file`?

By default, Sendmail uses DNS for name resolution, not the `/etc/hosts` file. The behavior can be changed through the use of the `/etc/mail/service.switch` file.

If you wish to query the hosts file before DNS servers, create a `/etc/mail/service.switch` file which contains the following line:

```
hosts       files dns
```

If you wish to query ONLY the hosts file, use the following:

```
hosts       files
```

Send Sendmail a HUP signal:

```
# kill -HUP `head -1 /var/run/sendmail.pid`
```

and the changes will take effect.

10.7 - Setting up a Secure HTTP server with SSL(8)

OpenBSD ships with an SSL-ready httpd and RSA libraries. For use with httpd(8), you must first have a certificate created. This will be kept in */etc/ssl/* with the corresponding key 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 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** flag (see the section about rc(8) in this faq), enabling https transactions with your machine on port 443.

10.8 - I edited /etc/passwd, but the changes didn't seem to take place. Why?

If you edit */etc/passwd* directly, 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 introduced the master.passwd file, which has an extended format (with additional options beyond those 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.

10.9 - What is the best way to add and delete users?

OpenBSD provides two commands for easily adding users to the system:

- adduser(8)
- user(8)

The easiest 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. 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.
```

Adding users via user(8)

These tools are less interactive than the adduser(8) command, which makes them easier to use in scripts.

The full set of tools is:

- group(8)
- groupadd(8)
- groupdel(8)
- groupinfo(8)
- groupmod(8)
- user(8)
- useradd(8)
- userdel(8)
- userinfo(8)
- usermod(8)

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$06$YOdOZM3.4m6MObBXjeZtBOWArqC2.uRJZXUkOghbieIvSWXVJRzlq
```

Now that we have our encrypted password, we are ready to add the user.

```
# user add -p '$2a$06$YOdOZM3.4m6MObBXjeZtBOWArqC2.uRJZXUkOghbieIvSWXVJRzlq' -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
```

Removing users

To remove users with the user(8) hierarchy of commands, you will use userdel(8). This is a very simple, yet usable 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.

10.10 - 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 */usr/bin/false* into */etc/shells*. Then when you set a users shell to /usr/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!

10.11 - Setting up Quotas

Quotas are used to limit user's space that they have available to them on your disk drives. It can be very helpful in situations where you have limited resources. Quotas can be set by user and/or by group.

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 filesystem 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 overridden by specifying the file name with the quota option in /etc/fstab, such as "userquota=/var/quotas/quota.user". Here is an example /etc/fstab that has one filesystem with userquotas enabled, and the quota file in a non-standard location:

```
/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)
```

Note that the quota allocation is in 1k blocks. In this case, the softlimit is set to 1000k, and the hardlimit is set to 1050k. 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 using quota(1). Using a command of "quota <user>" will give that user's information. When called with no arguments, the quota(1) 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              27       0       0
```

By default quotas set in /etc/fstab will be started on boot. To turn them off use

```
# quotaoff -a
```

10.12 - Setting up KerberosV Clients and Servers

OpenBSD includes KerberosV as a pre-installed component of the default system.

For more information on KerberosV, from your OpenBSD system, use the command:

```
 # info heimdal
```

10.13 - 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 usable 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 /usr/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 /usr/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*.

```
# adduser
Use option ``-silent'' if you don't want 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_]: 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 []:
Set the password so that user cannot logon? (y/n) [n]: 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.

```
# pwd
/home
# 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* and made inetd re-read its configuration file.

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.

10.14 - 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 3.3 (GENERIC) #44: Sat Mar 29 13:22:05 MST 2003
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> 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: ftpchroot,v 1.3 1996/07/18 12:12:47 deraadt 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
src
226 Transfer complete.
ftp> quit
```

```
221 Goodbye.
```

10.15 - Applying patches in OpenBSD

The OpenBSD source tree is constantly changing and improving, along with this fixes to common problems are often made and patches released to the public. These patches appear on the errata web page located at http://www.openbsd.org/errata.html, and are separated into categories. These categories correspond to patches that should be applied to different architectures or architecture independent patches.

Note, however, that patches aren't made for new additions to OpenBSD, and are only done for important reliability fixes or security problems that should be addressed right away, although the choice to do so is, as always, up to the administrator.

For the examples I will be patching talkd(8) with a security fix from the patch obtained from errata.html.

How are these patches different from what I would find in the CVS tree?

All patches posted at http://www.openbsd.org/errata.html are patches directly against the latest release's source tree. Patches against the latest CVS tree might also include other changes that wouldn't be wanted on a release system.

Getting your system ready to be patched.

Patches for the OpenBSD Operating System are distributed as diffs, which are text files that hold differences to the original source code. They are **NOT** distributed in binary form. This means that to patch your system you must have the source code from the **RELEASE** version of OpenBSD readily available. This does not mean that you must have ALL source code to the OpenBSD operating system to patch your system, but must have all code for the program which you are patching. For instance, if you are patching the kernel you must have all source for the kernel on hand.

cvs(1) is a very handy tool that can be used to grab only the source that you need via any of the anonymous cvs servers located around the world. You can get a listing of these servers at http://www.openbsd.org/anoncvs.html.

To retrieve the source code for talkd(8) from *3.3-release* using cvs(1), you would use the following line:

```
$ cvs -d anoncvs@anoncvs5.usa.openbsd.org:/cvs co -rOPENBSD_3_3_BASE
src/libexec/talkd/
cvs server: Updating src/libexec/talkd
U src/libexec/talkd/announce.c
U src/libexec/talkd/talkd.c
U src/libexec/talkd/talkd.h
```

To find the CVS path to the code that you need, you can find this in the patch on the *Index:* line. In this case, the CVS path was *src/libexec/talkd/*. Always check out the revision of OPENBSD_version_number_BASE. Without "_BASE" you will be checking out the stable branch, which might contain other changes that will interfere. If you are already tracking the patch branch, the patches should already be in that source, however you should always check and make sure. You can always look at http://www.openbsd.org/plus.html to see which patches have been applied to the patch branch. If the patches haven't been applied yet, you will need to grab the latest release source using the commands above.

Also, for those users that bought official OpenBSD CDs, you can get the source code directly off of the CD. Refer to the CD insert on how to extract the source from the CD. In which case you won't need to obtain the source via anoncvs.

```
Apply by doing:
        cd /usr/src
        patch -p0 < 026_talkd.patch
        cd libexec/talkd
        make obj && make depend && make && make install

Index: libexec/talkd/announce.c <------ Path to sources
===================================================================
RCS file: /cvs/src/libexec/talkd/announce.c,v
retrieving revision 1.8
retrieving revision 1.9
diff -u -r1.8 -r1.9
--- libexec/talkd/announce.c    1998/08/18 03:42:10     1.8
+++ libexec/talkd/announce.c    2000/07/06 00:01:45     1.9
@@ -160,6 +160,6 @@
                *(bptr++) = '\n';
        }
```

```
            *bptr = '\0';
    -       fprintf(tf, big_buf);
    +       fprintf(tf, "%s", big_buf);
            fflush(tf);
     }
```

Once you've obtained the proper sources, you can obtain the patch and place it in *src/*

Applying Patches

```
$ cd /usr/src
$ patch -p0</path/to/026_talkd.patch
Hmm...  Looks like a unified diff to me...
The text leading up to this was:
--------------------------
|Apply by doing:
|       cd /usr/src
|       patch -p0 < 026_talkd.patch
|       cd libexec/talkd
|       make obj && make depend && make && make install
|
|Index: libexec/talkd/announce.c
|===================================================================
|RCS file: /cvs/src/libexec/talkd/announce.c,v
|retrieving revision 1.8
|retrieving revision 1.9
|diff -u -r1.8 -r1.9
|--- libexec/talkd/announce.c   1998/08/18 03:42:10     1.8
|+++ libexec/talkd/announce.c   2000/07/06 00:01:45     1.9
--------------------------
Patching file libexec/talkd/announce.c using Plan A...
Hunk #1 succeeded at 160. <------------ Patch Succeeded
done
$ cd /usr/src/libexec/talkd/
$ ls
CVS             announce.c      print.c         table.c         talkd.c
Makefile        announce.c.orig process.c       talkd.8         talkd.h
$ make obj && make depend && make
making /home/ericj/lsrc/src/libexec/talkd/obj
mkdep -a /home/ericj/lsrc/src/libexec/talkd/talkd.c
/home/ericj/lsrc/src/libexec/talkd/announce.c
/home/ericj/lsrc/src/libexec/talkd/process.c
/home/ericj/lsrc/src/libexec/talkd/table.c /home/ericj/lsrc/src/libexec/talkd/print.c
cc -O2     -c /home/ericj/lsrc/src/libexec/talkd/talkd.c
cc -O2     -c /home/ericj/lsrc/src/libexec/talkd/announce.c
cc -O2     -c /home/ericj/lsrc/src/libexec/talkd/process.c
cc -O2     -c /home/ericj/lsrc/src/libexec/talkd/table.c
cc -O2     -c /home/ericj/lsrc/src/libexec/talkd/print.c
cc   -o ntalkd talkd.o announce.o process.o table.o print.o
nroff -Tascii -mandoc /home/ericj/lsrc/src/libexec/talkd/talkd.8 > talkd.cat8
$ sudo make install
install -c -s -o root -g bin  -m 555 ntalkd /usr/libexec
install -c -o root -g bin -m 444 talkd.cat8 /usr/share/man/cat8/talkd.0
/usr/share/man/cat8/ntalkd.0 -> /usr/share/man/cat8/talkd.0
```

Once you have done that, you should restart that service.

10.16 - Tell me about this chroot() Apache?

Since OpenBSD 3.2, the Apache httpd(8) server has been chroot(2)ed by default. While this is a tremendous boost to security, it can create issues, if you are not prepared.

What is a chroot?

A chroot(2)ed application is locked into a particular directory and unable to wander around the rest of the directory tree, and sees that directory as its "/" (root) directory. In the case of httpd(8), the program starts, opens its log files, binds to its TCP ports (though, it doesn't accept data yet), and reads its configuration. Next, it locks itself into */var/www* and drops privileges, then starts to accept requests. This means all files served and used by Apache must be in the */var/www* directory. This helps security tremendously -- should there be a security issue with Apache, the damage will be confined to a single directory with only "read only" permissions and no resources to cause mischief with.

What does this mean to the user?

Put bluntly, chroot(2)ing Apache is something new, and many older applications and system configurations will not work as before.

- **Historic file system layouts:** Servers upgraded from older versions of OpenBSD may have web files located in user's directories, which clearly won't work in a chroot(2)ed environment, as httpd(8) can't reach the */home* directory. Administrators may also discover their existing /var/www partition is too small to hold all web files. Your options are to restructure or do not use the chroot(2) feature. You can, of course, use symbolic links in the user's home directories pointing to subdirectories in */var/www*, but you can NOT use links in */var/www* pointing to other part of the file system -- that is prevented from working by the chroot(2)ing. Note that if you want your users to have chroot(2)ed FTP access, this will not work, as the FTP chroot will (again) prevent you from accessing the targets of the symbolic links. A solution to this is to not use */home* as your home directories for these users, rather use something similar to */var/www/home*.
- **Log Rotation:** Normally, logs are rotated by renaming the old files, then sending httpd(8) a SIGUSR1 signal to cause Apache to close its old log files and open new ones. This is no longer possible, as httpd(8) has no ability to open its own log files once privileges are dropped. httpd(8) must be stopped and restarted:

  ```
  # apachectl stop && apachectl start
  ```

  There are also other strategies available, including logging to a pipe(2), and using an external log rotator at the other end of the pipe(2).
- **Existing Apache modules:** Virtually all will load, however some may not work properly in chroot(2), and many have issues on "`apachectl restart`", generating an error, which causes httpd(8) to exit.
- **Existing CGIs:** Most will NOT work as is. They may need programs or libraries outside */var/www*. Some can be fixed by compiling so they are statically linked (not needing libraries in other directories), most may be fixed by populating the */var/www* directory with the files required by the application, though this is non-trivial and requires considerable programming knowledge -- most users will find it easier to just disable the chroot(2) feature until they are updated.

In some cases, the application or configuration can be altered to run within the chroot. In other cases, you will simply have to disable this feature using the `-u` option for httpd(8) in */etc/rc.conf*.

10.17 - I don't like the standard root shell!

The default shell for *root* on OpenBSD is csh, due primarily to tradition. There is no requirement that OpenBSD have csh(1) for the root login (though keep reading before changing it).

Some users who come from other Unix-like operating systems find csh(1) unfamiliar, and ask if and how they can change it. There are a few options:

- **Don't login as *root!*** Between su and sudo, there should be few reasons for users to log in as *root* for most applications after initial setup.
- **Invoke your favorite shell after login:** If you like ksh(1) or any other shell, just invoke it from the default shell.
- **Change the root shell:** This can be done using chsh or vipw.

A traditional Unix guideline is to only use statically compiled shells for root, because if your system comes up in single user mode, non-root partitions won't be mounted and dynamically linked shells won't be able to access libraries located in the `/usr` partition. This isn't actually a significant issue for OpenBSD, as the system will prompt you for a shell when it comes up in single user mode, and the default is sh. The three standard shells in OpenBSD (csh, sh and ksh) are all statically linked, and thus usable in single user mode.

It is sometimes said that one should never change the root shell, though there is no reason not to in OpenBSD. But again, this shouldn't be an issue -- just don't log in as root.

10.18 - What else can I do with *ksh*?

In OpenBSD, ksh is pdksh, the Public Domain Korn Shell, and is the same binary as sh.

Users comfortable with *bash*, often used on Linux systems, will probably find ksh very familiar. Ksh(1) provides most of the commonly used features in *bash*, including tab completion, command line editing and history via the arrow keys, and CTRL-A/CTRL-E to jump to

beginning/end of the command line. If other features of *bash* are desired, *bash* itself can be loaded via either ports or packages.

The command prompt of *ksh* can easily be changed to something providing more information than the default "$ " by setting the `PS1` variable. For example, inserting the following line:

```
export PS1='$PWD $ '
```

in your `/etc/profile` produces the following command prompt:

```
/home/nick $
```

See the file `/etc/ksh.kshrc`, which includes many useful features and examples, and may be invoked in your user's `.profile`.

[FAQ Index] [To Section 9 - Migrating from Linux] [To Section 11 - Performance Tuning]


www@openbsd.org

$OpenBSD: faq10.html,v 1.89 2003/05/05 03:49:58 david Exp $

OpenBSD

11.0 - Performance Tuning

Inhaltsverzeichnis

Wenn du einen vielbesuchten Server, ein Gateway oder eine Firewall administrierst, möchtest oder musst du vielleicht einige der standardmässigen Parameter anpassen, um eine optimale Performance zu erhalten. Die options(4) man page berichtet über die angebotenen Kerneloptionen. Diese Optionen werden in der Kernel-Konfigurationsdatei plaziert, bevor du einen eigenen Kernel kompilierst. Mehr Details dazu erhälst du in der FAQ 5.

11.1 - Netzwerk

Ein Parameter, der bei einem besonders belasteten Server, Gateway oder Firewall vielleicht angepasst werden muss, ist NMBCLUSTERS. Er kontrolliert die Grösse der kernel mbuf cluster map. Wenn du Meldungen wie "mb_map full" auf deinem Computer bekommst, musst du die Werte für diesen Parameter vergrössern. Wenn ohne ersichtlichen Grund der Traffic auf einem Netzwerk aufhört, kann das ebenfalls ein Zeichen für zu kleine NMBCLUSTERS sein. Ein sinnvoller Wert beim i386 port mit mindestens 100Mbps ethernet Interfaces (egal wieviele davon die Maschine hat) ist 8192.

option NMBCLUSTERS=8192

Die Standard-Anzahl von NMBCLUSTERS variiert von Plattform zu Plattform, und reicht von 256 bis 2048. Sie werden in einer plattform-abhängigen Header-Datei gesetzt, es sei denn, sie werden von einer "option"-Zeile in einer Kernel-Konfigurations-Datei überschrieben.

11.2 - Festplatten I/O

Festplatten I/O Geschwindigkeit ist ein wichtiger Faktor in der Gesamtgeschwindigkeit deines Computers. Sie wird umso wichtiger, wenn dein Computer eine Multi-User-Umgebung beheimatet (User aller Arten, von solchen, die sich einloggen, bis zu denen die Serverdienste nutzen). Datenspeicher brauchen ständige Aufmerksamkeit. Insbesondere, wenn deine Partition überläuft, oder deine Platten versagen. OpenBSD kennt verschiedene Optionen, um die Geschwindigkeit deiner Festplattenoperationen zu erhöhen und Fehlertoleranz zu bieten.

Inhaltsverzeichnis

11.2.1 - CCD

Die erste Option ist die Benutzung des ccd(4), des Concatenated Disk Driver. Das erlaubt dir, mehrere Partitionen in eine virtuelle Platte zu verwandeln (und damit kannst dafür sorgen, dass mehrere Festplatten wie eine einzige aussehen). Dieses Konzept ist ähnlich wie das von LVM (logical volume management), das in mehreren kommerziellen Unix-Arten zu finden ist.

Wenn du GENERIC benutzt, ist ccd bereits eingeschaltet (in `/usr/src/sys/conf/GENERIC`). Wenn du einen veränderten Kernel benutzt, musst du es vielleicht wieder in deine Kernel-Konfiguration einfügen. Wie auch immer, auf jeden Fall muss sich eine Zeile wie die folgende in deiner Konfigurationsdatei befinden:

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

Das obige Beispiel gibt die bis zu 4 ccd Devices (virtuelle Platten). Jetzt musst du festlegen, welche Partitionen auf deinen realen Festplatten du in den ccd einbinden willst. Benutze disklabel, um diese Partitionen als 'ccd'-Typ zu markieren. Auf einigen Architekturen erlaubt dir disklabel das vielleicht nicht. In diesem Fall markiere sie einfach als 'ffs'.

Wenn du ccd dazu benutzt, um mittels striping Performance zu gewinnen, solltest du wissen, dass du keine optimale Performance bekommst, bis du das gleiche Festplatten-Modell mit den gleichen disklabel Einstellungen benutzt.

Editiere /etc/ccd.conf, bis sie etwa so aussieht : (Mehr Informationen über das Konfigurieren von ccd findest du unter ccdconfig(8))

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

Um die Änderungen wirksam zu machen, führe das hier aus:

```
# ccdconfig -C
```

Solange /etc/ccd.conf existiert, wird sich ccd automatisch beim Booten konfigurieren. Jetzt hast du eine neue Festplatte, ccd0, eine Kombination von /dev/sd2e und /dev/sd3e. Benutze disklabel einfach wie gewöhnlich, um die Partition oder Partitionen zu erzeugen, die du benutzen willst. Nutze erneut die 'c' Partition nicht, um darauf irgendetwas zu speichern. Stelle sicher, dass deine benutzten Partitionen mindestens einen Zylinder vom Anfang der Disc weg ist.

11.2.2 - RAID

Eine weitere Lösung ist raid(4), wofür du raidctl(8) nutzen musst, um deine RAID Geräte zu kontrollieren. OpenBSD's RAID basiert auf Greg Oster's NetBSD port der CMU RAIDframe Software. OpenBSD hat Unterstützung für die RAID-Level 0, 1, 4, und 5.

Für RAID muss, wie auch bei ccd, Unterstützung im KERNEL sein. Diese Treiber-Unterstützung für RAID ist im Gegensatz zu ccd allerdings nicht im GENERIC-Kernel enthalten, also muss sie extra in deinen Kernel einkompiliert werden (RAID-Unterstützung vergrössert deinen i386 Kernel um gute 500k!)

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

Ein RAID aufzusetzen ist mit einigen Betriebssystemen verwirrend und schmerzhaft, um es sanft auszudrücken. Nicht jedoch mit RAIDframe. Lies die raid(4) und raidctl(8) man pages für die

kompletten Details. Es gibt dafür viele Optionen und mögliche Konfiguationen, und ein detaillierter Überblick sprengt den Rahmen dieses Dokumentes.

11.2.3 - Filesystem Buffer

Fileserver, die noch Speicher überhaben, können die BUFCACHEPERCENT erhöhen. Das heisst, welcher Prozentsatz deines RAM als "file system buffer" (Dateisystem-Puffer) genutzt werden soll. Diese Option wird vielleicht geändert, wenn der Unified Buffer Cache komplett und ein Teil von OpenBSD ist. In der Zwischenzeit solltest du eine Zeile wie diese zu deiner Kernel-Konfiguration hinzufügen, um BUFCACHEPERCENT zu erhöhen:

option BUFCACHEPERCENT=30

Natürlich kannst du ihn auch auf 5 Prozent setzen (dem Standard) oder auch so hoch wie 50 Prozent (oder auch mehr.)

11.2.4 - Soft Updates

Ein weiteres Tool zum Erhöhen der Systemgeschwindigkeit sind Soft Updates. Eine der langsamsten Operationen im traditionellen BSD Dateisysteme ist das Updaten der Metainfos (was unter anderem immer dann geschieht, wenn du Dateien oder Verzeichnisse erzeugst oder löschst.) Soft Updates versucht die Metainfo im RAM upzudaten, statt jedes einzelne Metainfo-Update auf die Platte zu schreiben. Ein weiterer Nebeneffekt ist, dass die Metainfos auf der Festplatte immer auf dem aktuellen Stand sind. Das heisst, ein Systemcrash sollte kein fsck(8) beim folgende Booten benötigen, sondern eine einfache Hintergrund-Version von fsck, die Änderungen an den Metainfos im RAM macht (a la softupdates). Das heisst, dass Reboots viel schneller sind, da nicht mehr auf fsck gewartet werden muss! (OpenBSD hat dieses Feature leider noch nicht.) Mehr über Soft Updates findest du im Soft Updates FAQ Eintrag.

11.2.5 - Grösse des namei() cache

Hinweis: Vorher hat die options(4) manual page empfohlen, die `NVNODE=integer` Kernel Option zu setzen. Das wird nicht mehr empfohlen; du solltest stattdessen das sysctl(8) Kommando benutzen.

Die name-to-inode Übersetzung (a.k.a., namei()) cache kontrolliert die Geschwindigkeit der pathname zu inode(5) Übersetzung. Standardmässig hat dieser Cache `NPROC * (80 + NPROC / 8)` Einträge. `NPROC` ist auf `20 + 16 * MAXUSERS` gesetzt; in der config(8) manual page steht eine Erklärugn der `maxusers` Kernel-Konfigurations-Parameter. Ein sinnvoller Weg zum Herausfinden der passenden Grösse des Cache wäre, eine große Anzahl von namei() cache misses vorrausgesetzt, die man mit einem Tool wie systat(1) messen könnte, wäre eine Untersuchung des momentanen berechneten Wertes mittels sysctl(8), (das diesen Parameter "`kern.maxvnodes`" nennt) und diesen Wert zu vergössern, bis sich entweder die namei() cache hit rate verbessert, oder es bewiesen ist, dass das System nicht wesentlich von einer Erhöhung der Grösse des namei() cache profitiert. Nachdem der Wert gestgestellt wurde, kannst du ihn für die nächsten Systemstarts mit sysctl.conf(5) setzen.

11.4 - Hardware Auswahl

(Hinweis - diese Sektion dreht sich fast ausschliesslich um die i386 oder PC Architektur. Andere Architekturen geben dir sozusagen keine so grosse Auswahl!)

Die Performance deiner Anwendungen hängt stark von deinem Betriebssystem und den Fähigkeiten ab, die es

bereitstellt. Das mag ein Grund dafür sein, dass du OpenBSD benutzt. Die Performance deiner Anwendungen hängt aber auch stark von deiner Hardware ab. Für viele Leute ist das Preis-Leistungs-Verhältnis eines brandneuen PC mit einem Intel Pentium IV oder AMD Athlon Prozessors viel besser als das Preis-Leistungs-Verhältnis einer Sun UltraSparc 60! Der Preis von OpenBSD ist natürlich unschlagbar.

Wenn du einen neuen PC kaufen willst, ob nun in einem Komplettangebot, oder Einzelteil für Einzelteil, solltest du sicherstellen, dass du unbedingt nur zuverlässige Teile bekommst. In der Welt der PCs ist das leichter gesagt als getan. **Schlechte oder sonstige unzuverlässige oder unpassende Teile können dazu führen, dass OpenBSD schlecht läuft und oft abstürzt**. Der beste Rat, den wir geben können, ist, vorsichtig zu sein, und Marken und Teile zu kaufen, die von jemandem empfohlen werden, dem du trauen kannst. Wenn du nur auf den Preis eines PCs achtest, wirst du wahrscheinlich auch an Qualitält verlieren!

Es gibt ein paar Dinge, die dir helfen können, die maximale Performance aus deiner Hardware zu holen:

- Benutze mehrere Festplatten.

  Statt nur eine 20GB Platte zu kaufen, kaufe mehrere 9GB Platten. Wenn das auch mehr kostet, wird es doch die Last auf mehrere Spindeln verteilen, und somit die Zeitspanne verringern, die ein Datenzugriff benötigt. Ausserdem kannst du mit mehreren Platten auch mehr Zuverlässigkeit und schnelleren Datenzugriff mit RAID bekommen.

- Benutze SCSI, wenn du hohe Festplatten-IO-Geschwindigkeit brauchst.

  IDE Festplatten laufen normalerweise mit 5400 RPM bis 7200 RPM. Selbst bei hochwertigen IDE Platten ist es manchmal zu viel verlangt, wenn man mehr als 15 bis 20 MB pro Sekunde an Datendurchsatz von einer einzelnen Platte verlangt. Mit hochwertigen SCSI-Platten (10k RPM oder 15k RPM) kannst du mehr Durchsatz bekommen. Im Gegensatz dazu ist es eine Verschwendung von Geld, wenn du mittlere oder langsame SCSI-Platten benutzt, da dann IDE die gleiche oder bessere Leistung bringt.

  Wenn du einen Server baust, und mehr als 20 GB Plattenplatz brauchst, solltest du über SCSI nachdenken. IDE beschränkt dich auf zwei Platten pro Controller. Gleichzeitige Zugriffe auf diese Platten haben vermutlich einen negativen Effekt auf die I/O Performance dieser Platten. Mit Wide SCSI kannst du 15 Platten pro Controller anschliessen, und es hat bessere Unterstützung für gleichzeitigen Zugriff als IDE.

- Benutze SDRAM statt DRAM.

  Diese Option trift fast nur auf PCs zu. Bei den meisten anderen Architekturen hast du keinerlei Auswahl welche Art von RAM du benutzen kannst. Bei den meisten PCs schon. Mit SDRAM bekommst du eine bessere Performance als mit DRAMs (SIMMs). Wenn dein System RDRAM unterstützt, oder vielleicht DDR oder eine andere neue Art von RAM bist du sogar noch besser dran..

- Benutze ECC oder parity RAM.

  Parity fügt einen Mechanismus hinzu, der prüft, ob die Daten im RAM noch in Ordnung sind. ECC baut das noch dahingehend aus, dass es versucht, Fehler bei einzelnen Bits automatisch zu korrigieren. Diese Option gibt es wieder fast nur bei PCs. Die meisten anderen Architekturen brauchen einfach ECC oder parity RAM. Einige nicht-PC-Computer booten nicht einmal mit nicht-parity-RAM. Wenn du kein ECC/parity RAM benutzt, kann es zu Daten-Korrumpierung und anderen Abnormitäten kommen. Einige Herstelle von "billigem PC RAM" stellen nicht einmal eine ECC-Variante her! Das hilft dir, sie zu vermeiden! PC Hersteller verkaufen oftmals mehrere Produktlinien, in "Server" und "Workstations" aufgeteilt. Die Server haben parity (und jetzt ECC) seit vielen Jahren beinhaltet. Unix-Workstation-Hersteller benutzen parity (und nun ECC) seit vielen Jahren in all ihren Produktlinien.

- Vermeide ISA-Karten.

  Während die meisten Leute ISA-Karten schon deshalb meiden, weil sie veraltet und zudem noch schwer zu konfigurieren sind, gibt es aber trotzdem noch eine ganze Menge davon. Wenn du den ISA Bus für deine Festplatte oder Netzwerk-Karte benutzt (oder noch schlimmer, für beides) denke daran, dass der ISA Bus vermutlich ein Flaschenhals ist. Wenn du Geschwindigkeit brauchst, benutze PCI. Natürlich gibt es noch zahllose ISA-Karten, die einfach gut funktionieren. Unglücklicherweise sind das meist Sound-Karten oder solche für serielle Ports.

- Vermeide billige PCI Netzwerk-Karten.

OpenBSD unterstützt eine ganze Menge von billigen PCI Netzwerk-Karten. Diese Karten funktionieren prima in einfachen Heim-Systemen, oder solchen mit wenig oder moderater Netzwerk-Last im Geschäfts- oder Forschungs-Bereich. Aber, wenn du hohen Durchsatz brauchst, und wenig Belastung deines Servers, bist du mit einer Qualitäts-Netzwerk-Karte besser dran. Unglücklicherweise sind einige Serien von teuren Marken-Herstellern (wie die 3com XL Serie) nicht besser als die billigen Karten. Ein echter Favorit unter den 10/100Mbps Netzwerk-Karten ist dagegen die Intel EtherExpress PRO/100.

11.5 - Wieso benutzen wir keine async mounts?

Frage: "Ich gebe einfach ein "mount -u -o async /"ein, was ein Paket, was ich brauche, benutzbar macht. (das darauf besteht alle paar Momente ein paar hundert Dateien zu ändern.) Wieso wird asynchrones mounting abgelehnt und ist nicht standardmässig aktiviert (wie in manchen anderen Unixen) ? Wäre das nicht ein einfacherer, und daher auch sichererer Weg, die Performance mancher Applikation zu erhöhen ?"

Antwort: "Asynchrone mounts sind tatsächlich schneller als synchrone mounts, aber sie sind unsicherer. Was passiert im Falle eine Stromausfalls? Oder bei einem Hardwareproblem ? Die Suche nach Geschwindigkeit darf nicht auf Kosten von Stabilität und Zuverlässigkeit des Systems gehen. Siehe auch die man page von 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 guaran-
        tee to keep a consistent file system structure on the
        disk.  You should not use this flag unless you are pre-
        pared 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 in-
        crease.
```

Auf der anderen Seite, wenn du sowieso nur mit temporären Daten umgehst, die du nach einem Crash wieder rekonstruieren kannst, kannst du mehr Geschwindigkeit erhalten, indem du eine separate Partition nur für diese Daten benutzt, die asynchron gemountet ist. Tue das aber *nur*, wenn dir der Verlust aller Daten in der Partition nach irgendeinem Problem nichts ausmacht. Daher sind mfs(8) Partitionen asynchron gemountet, weil sie ja nach jedem Reboot sowieso gelöscht und neu erzeugt werden.

11.6 - Tunen deiner Monitorauflösung unter XFree86

Es ist durchaus mit vielen multi-Sync-Monitoren möglich, einen X Server in einer azeptablen Auflösung zum Laufen zu kriegen. Mit den Standard-Konfigurations-Werkzeugen xf86config oder XF86Setup ist es aber recht schwierig, ein gutes Ergebnis zu erhalten. Einer der schmerzvolleren Punkte ist es, deinen Monitor zur gewünschten Auflösung zu bewegen, und dann eine vertikale Scan-Rate von mindestens 72-75 Hz zu bekommen, eine Rate, bei der das Bildschirmgeflacker wesentlich geringer sichtbar für menschliche Augen ist. Was passiert aber, wenn du die vertikale Scan-Rate sehr niedrig einstellst? So könntest du den Bildschirm zum Beispiel ohne Flackern auf Video filmen, aber auch dazu sind die Methoden mit den Standard-Werkzeugen von XFree86 eher nicht-intuitiv.

Schlussendlich ist es bei den Auflösungen, (800x600, 1024x768, 1152x900,1280x1024), die die meisten Leute heute mit preiswerten VGA-Monitoren benutzen (zumindest mit neueren Modellen) bestens möglich, vertikale Wiederholungsraten von 85 Hz und mehr zu bekommen, um ein wirklich klares und ansehnliches Bild zu erhalten. Der XFree86 X Server hat einen Mechanismus, der dir erlaubt, im Detail den Grafik-Modus zu beschreiben, den du benutzen willst, dies nennt sich ModeLine. Eine ModeLine hat vier Sektionen, eine einzelne Nummer für die Pixel Clock, vier Nummern für horizontales Timing, vier Nummern für vertikales Timing, und eine optionale Sektion mit einer Liste von Flags für weitere Charakteristika wie etwa den Modus (z.B. Interlace, DoubleScan, und weitere.. mehr Details gibt es in der XF86Config(5) manual page)

Das Erzeugen einer ModeLine ist eine schwarze Kunst.. Glücklicherweise gibt es mehrere Skripte, die das für dich

erledigen können. Eines davon ist der Colas XFree86 ModeLine Generator. Ein weiteres ist der The XFree86 Modeline Generator , der bei SourceForge gehostet wird, und es gibt weitere bei Freshmeat. Bevor du diese ModeLine-Generatoren benutzen kannst, musst du die vertikalen und horizontalen sync Limits für deinen Monitor herausfinden. Diese Angaben finden sich oftmals im Handbuch, oder auf der Webseite des Monitor-Herstellers. Wenn du sie dort nicht finden kannst, suche einfach im Web nach deinem Modell und Hersteller, viele Leute waren so freundlich, Listen mit den entsprechenden Angaben zu erstellen.

Sagen wir zum Beispiel, du hättest einen Dell D1226H Monitor. Du hast auf Dell's Website herausgefunden, das er einen Bereich von 30-95 kHz horizontal und 50-160 Hz vertikal hat. Besuche die ModeLine Generator Page, und gib diese Informationen ein. Als nächstes musst du die minimale vertical scan rate eingeben, die du haben willst. Jede Rate ab 72 Hz und grösser sollte im allgemeinen wenig flackern. Je mehr, desto besser wird das Bild.

Mit all diesen Informationen wird das Skript eine ModeLine für jede mögliche 4x3 Auflösung generieren, die dein Monitor unterstützen kann. Wenn jemand die Dell Spezifikationen von oben und eine minimale vertikale Rate von 75 Hz eingibt, gibt das Skript etwas ähnliches wie das folgende aus:

```
ModeLine "320x240" 20.07 320 336 416 448 240 242 254 280 #160Hz
ModeLine "328x246" 20.86 328 344 424 456 246 248 260 286 #160Hz
...
ModeLine "816x612" 107.39 816 856 1056 1136 612 614 626 652 #145Hz
ModeLine "824x618" 108.39 824 864 1064 1144 618 620 632 658 #144Hz
ModeLine "832x624" 109.38 832 872 1072 1152 624 626 638 664 #143Hz
...
ModeLine "840x630" 109.58 840 880 1080 1160 630 632 644 670 #141Hz
ModeLine "848x636" 110.54 848 888 1088 1168 636 638 650 676 #140Hz
...
ModeLine "1048x786" 136.02 1048 1096 1336 1432 786 788 800 826 #115Hz
ModeLine "1056x792" 136.58 1056 1104 1344 1440 792 794 806 832 #114Hz
ModeLine "1064x798" 137.11 1064 1112 1352 1448 798 800 812 838 #113Hz
...
ModeLine "1432x1074" 184.07 1432 1496 1816 1944 1074 1076 1088 1114 #85Hz
ModeLine "1576x1182" 199.86 1576 1648 2008 2152 1182 1184 1196 1222 #76Hz
ModeLine "1584x1188" 198.93 1584 1656 2016 2160 1188 1190 1202 1228 #75Hz
```

Dieser Monitor gibt nun vor, 1600x1200 @ 75 Hz machen zu können, aber das Skript sagt nicht, dass das innerhalb der 75 Hz sei. Wenn du also exakt 1600x1200 haben willst, geh ein wenig mit deiner minimalen vertikalen Rate herunter.. (Hier z.B. kannst du bis 70 Hz heruntergehen)

```
ModeLine "1592x1194" 197.97 1592 1664 2024 2168 1194 1196 1208 1234 #74Hz
ModeLine "1600x1200" 199.67 1600 1672 2032 2176 1200 1202 1214 1240 #74Hz
ModeLine "1608x1206" 198.65 1608 1680 2040 2184 1206 1208 1220 1246 #73Hz
ModeLine "1616x1212" 197.59 1616 1688 2048 2192 1212 1214 1226 1252 #72Hz
ModeLine "1624x1218" 199.26 1624 1696 2056 2200 1218 1220 1232 1258 #72Hz
ModeLine "1632x1224" 198.15 1632 1704 2064 2208 1224 1226 1238 1264 #71Hz
ModeLine "1640x1230" 199.81 1640 1712 2072 2216 1230 1232 1244 1270 #71Hz
ModeLine "1648x1236" 198.64 1648 1720 2080 2224 1236 1238 1250 1276 #70Hz
```

Hier sehen wir, dass der Monitor tatsächlich 1600x1200 @ 74 Hz macht, wenn die dot clock (Bandbreite) auf 200MHz begrenzt ist. Setze die Bandbreite gemäss der Grenzen, die vom Monitor definiert werden.

Nachdem du einmal die ModeLines hast, schreibe sie in deine /etc/XF86Config Datei. Kommentiere die alten ModeLines aus, so dass du sie noch benutzen kannst, falls die neuen nicht funktionieren. Als nächstes wähle aus, mit welcher Auflösung du nun arbeiten willst. Als erstes musst du nun herausfinden, ob X im "accelerated mode" läuft, oder nicht (das tut es mit den meisten Grafik-Karten), so dass du auch weisst, welche `"Screen"` Sektion der XF86Config-Datei du modifizieren musst. Alternativ kannst du natürlich einfach alle Screen-Sektionen modifizieren.

```
Section "Screen"
   Driver          "Accel"
   Device          "Primary Card"
   Monitor         "Primary Monitor"
   DefaultColorDepth 32
   SubSection "Display"
      Depth        32
      Modes        "1280x1024" "1024x768"
   EndSubSection
```

Die erste Auflösung nach dem "Modes" Stichwort ist die Auflösung, in der X startet. Mit dem Drücken von CTRL-ALT-KEYPAD MINUS oder CTRL-ALT-KEYPAD PLUS kannst du zwischen den hier aufgeführten Auflösungen hin- und herschalten. Gemäss der Angaben oben wird X versuchen im 32-Bit-Modus (wegen der DefaultColorDepth Direktive, ohne sie würde X im 8-Bit-Modus starten.) Die erste Auflösung, die versucht wird, ist 1280x1024 (es wird einfach der Reihenfolge in der 'Modes'-Zeile gefolgt.) Denke daran, dass "1280x1024" einfach ein Label für die Werte in der ModeLine ist.

Du solltest wissen, dass das ModeLine Generator-Skript Optionen hat, um seine Timings für ältere oder kleinere Monitore etwas zu lockern, und dass es die Möglichkeit hat, ModeLines für spezielle Monitore anzubieten. Abhängig davon, was für eine Hardware du hast, ist sie vielleicht nur schwer mit den Standard-Optionen zu betreiben. Wenn das Bild zu gross ist, zu breit oder zu klein, oder nicht genügend horizontal oder vertikal gekippt ist, und die Monitor-Kontrollen zur Kompensierung nicht ausreichen, kann man mittels xvidtune(1) die ModeLine besser dem Monitor anpassen.

In den meisten modernen Monitoren gibt es kein fixes Limit der Bandbreite, daher ist sie auch oftmals nicht in den Spezifikationen aufgeführt. Aber je mehr du in der Bandbreite nach oben gehst, desto verschwommener wird das Bild. Du könntest also zum Testen die Bandbreite deiner Grafikkarte (auch "dotclock" genannt) eingeben (so kannst du deinen Monitor nicht beschädigen) und Schritt-für-Schritt in BW heruntergehen, bis du ein schönes, klares Bild hast.

Wenn dir das unnötig kompliziert erscheint, liegt das daran, dass es genau das ist. XFree86 4.0 kümmert sich darum, und macht diesen Prozess bedeutend einfacher, da es viele eingebaute Modi hat, und ausserdem in der Lage ist, Angaben aus vielen "plug and play" DDC und DDC2 Monitoren auszulesen.

Du kannst das "Colas XFree86 ModeLine Generator script" hier herunterladen: http://koala.ilog.fr/ftp/pub/Klone/. Du brauchst den Klone Interpreter, und musst ihn kompilieren. Er ist in als `lang/klone` in den ports. Die Skripte existieren im "scripts" Verzeichnis der Klone Distribution. (Der port installiert sie nach /usr/local/lib/klone/scripts.)

Es sind zwei Versionen des Skriptes dabei, die erste ist eine CGI Version die identisch zu der obigen Webseite ist. Die zweite ist eine nicht-CGI-Version die deine komplette XF86Config-Datei nimmt, dekodiere die Monitor-Spezifikationen, die du in xf86config/XF86Setup eingegeben hast (Hast du eigentlich die echten Spezifikationen für deinen Monitor eingegeben, oder die generischen benutzt?) und passe die existierenden ModeLines an.

[Back to Main Index] [To Section 10.0 - System Administration] [To Section 12.0 - Für fortgeschrittene Users]

---

www@openbsd.org

Originally [OpenBSD: faq11.html,v 1.29]
$Translation: faq11.html,v 1.5 2003/01/17 14:02:59 jufi Exp $
$OpenBSD: faq11.html,v 1.5 2003/01/17 18:47:48 jufi Exp $

12 - Für fortgeschrittene User

Inhaltsverzeichnis

Mit *fortgeschrittenen Usern* meinen wir Leute, die ein Unix-System administrieren können und wissen, wie es funktioniert. Wenn du den Anweisungen in dieser Sektion folgst, ohne zu wissen, was du tust, kannst du deinem System Schaden zufügen!

12.1 - DMA Zugriff für IDE Festplatten erzwingen

Dem PCI IDE Code ist dein Chipset vielleicht nicht bekannt. Wenn das so ist, wirst du eine Meldung beim Booten bekommen, die etwa so aussieht:

```
pciide0: DMA, (unused)
```

Wenn du so eine Meldung erhältst, kannst du versuchen, den DMA Modus mittels 'flags 0x0001' in deinem pciide Eintrag in deiner Kernel-Konfigurationsdatei zu erzwingen. Das würde ungefähr so aussehen:

```
pciide* at pci ? dev ? function ? flags 0x0001
```

Wenn man das gemacht hat, wird der pciide Code versuchen, den DMA Modus zu benutzen, unabhängig davon, ob er weiss, wie er das mit deinem Chipsatz anstellen soll. Wenn das funktioniert, und es dein System durch 'fsck' und die restliche Startsequenz schafft, ist es wahrscheinlich, dass das Ganze weiter so funktioniert. Wenn es nicht klappt, und das System hängt oder in 'panic' verfällt, kannst du den DMA-Modus schlicht und einfach nicht benutzten (solange keine Unterstützung für deinen Chipsatz hinzugefügt wurde, natürlich). Wenn du die volle Dokumentation für deinen Chipsatz findest, ist das ein guter Anfang für volle Unterstützung im PCI-IDE Code des Kernels. Du kannst auf der Website des Herstellers nachsehen, oder sie auch anrufen. Wenn dein PCI-IDE Controller Teil deines Motherboards ist, finde heraus, wer der Hersteller des Chipsatzes ist, und wende dich an sie!

Wenn du diese Meldung beim Booten bekommst, weisst du dass dein DMA aktiviert wurde:

```
cd0(pciide0:1:0): using PIO mode 3, DMA mode 1
```

Das bedeutet, dass pciide0, channel 1, drive 0 (was hier ein ATAPI CD-ROM ist) DMA Datentransfers benutzt.


www@openbsd.org

Originally [OpenBSD: faq12.html,v 1.34]
$Translation: faq12.html,v 1.14 2003/04/09 12:21:32 jufi Exp $
$OpenBSD: faq12.html,v 1.14 2003/04/09 16:15:41 jufi Exp $

13 - Using IPsec (IP Security Protocol)

Table of Contents

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

13.1 - 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 (or here) 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 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

Ensure it is hard for anyone but the receiver to understand what data has been communicated. For example, ensuring the secrecy of 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 datagram is processed only once, regardless of how many times it is received. I.e. it should not be possible for an attacker to record a transaction (such as a bank account withdrawal), and then by replaying it verbatim cause the peer to think a new message (withdrawal request) had been received. *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 Authentication Header (AH), and Encapsulating Security Payload (ESP).

AH provides authentication, integrity, and replay protection (but not confidentiality). The main difference between the authentication features of AH and ESP is that AH also authenticates portions of the IP header of the packet (such as the source/destination addresses). ESP authenticates only the packet payload.

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.

In practice, it is recommended that ESP be used for most applications.

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:

IPhdr	**AH**	IPhdr2	TCPhdr	data

An example of a transport mode AH packet is:

IPhdr	**AH**	TCPhdr	data

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:

IPhdr	**AH**	**ESP**	*TCPhdr*	*data*

This is called **Transport Adjacency**. The tunneling version would look like:

IPhdr	**AH**	**ESP**	*IPhdr2*	*TCPhdr*	*data*

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 (AH 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 ISAKMP automatic key exchange (RFC2407, RFC2408, and RFC2409) in the isakmpd(8) daemons.

13.6 - How do I set up IPsec with manual keying?

Manual keying is the easiest way to get started with IPsec. You can set up 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 ensure that the IP ESP protocol is enabled in the OpenBSD kernel. Though it is enabled by default, you should verify this fact by doing the following:

```
# sysctl net.inet.esp.enable
net.inet.esp.enable = 1
```

Similarly, if AH is required, verify that it has been enabled:

```
# sysctl net.inet.ah.enable
net.inet.ah.enable = 1
```

If necessary, these protocols may be enabled using the *-w* option to sysctl(8).

You can edit `/etc/sysctl.conf` to turn these on (or off) 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.

Note that in most cases, such as with the *rc.vpn* script or with the example below, only ESP is required. In these cases, *you do not need to enable AH*. In fact, there may be security concerns related to enabling AH if you are not using it.

Next, you will 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 (20 bytes) of randomness, for example, do:

```
# openssl rand 20 | hexdump -e '20/1 "%02x"'
```

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 (40-128 bits recommended)
SKIPJACK  80 bits
```

Now, you need to set up 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, set up 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, set up 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, set up 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, set up 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, set up 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, set up 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 set up 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... Set up 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
```

Set up 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
```

Set up 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, set up 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 set up 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.

13.7.1 - What is isakmpd?

ISAKMP is the Internet Security Association and Key Management Protocol. It is sometimes referred to as IKE, or Internet Key Exchange. It is the standard key exchange mechanism for IPsec. It addresses 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, as it can expose identity information to an eavesdropper.

Phase 2 - Security Associations are negotiated on behalf of IPsec. Phase 2 establishes tunnels or endpoint SAs between IPsec hosts. **Quick Mode** is used in Phase 2 because there is no need to repeat a full authentication; 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 set up the actual tunnels. In Phase 1, your IPsec nodes establish a connection where they exchange authentication (Either a X.509 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.

13.7.2 - How do I get started with isakmpd?

By default, OpenBSD comes with the all the necessary binaries for ISAKMP and the IPsec stack. Example configuration files may be found in `/usr/share/ipsec/isakmpd`. For the purposes of this example, copy *policy* to `/etc/isakmpd/isakmpd.policy`. Copy *VPN-east.conf* to `/etc/isakmpd/isakmpd.conf`. Here we attempt to show you how to set up 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).

Although both the ESP and AH protocols are enabled in the kernel by default, you should verify the protocols you need are available by the procedure outlined in the FAQ section on Manual Keying. Next, you need to edit `/etc/isakmpd/isakmpd.policy`. This file tells ISAKMP who can access IPsec. In this scenario, the policy file states that anybody who sends data using Encapsulating 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 set up the variables that will affect the main behavior 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 fulfill 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 set up 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.

13.7.3 - Starting isakmpd

You may wish to use

```
# isakmpd -d -DA=90
```

the first time you decide to run this daemon. The daemon will not be running in daemon mode but as a regular (foreground) 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**.

13.8 - 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, makes key management much more flexible.

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.

In order to be usable by isakmpd, the X.509 certificates need to have a Subject Alternative Name (subjectAltName) extension added to them. This extension describes the certificate holder's identity, and may be added by using either certpatch(8), or a custom openssl configuration file such as `/etc/ssl/x509v3.cnf`. The examples in README.PKI describe this process using IP addresses as subjectAltName identifiers.

Though IP addresses are the default mechanism for subjectAltName extensions, certpatch also supports using either a FQDN (Fully Qualified Domain Name) or a UFQDN (User FQDN). These can be very useful for mobile users. An example of an FQDN might be `www.openbsd.org`. An example of an UFQDN could be an email address, such as `Jorgen.Granstam@abc.se`. It should be noted that unlike IP-based identifiers, isakmpd performs no verification of the data presented in FQDN or UFQDN-based identification; the identifiers are treated simply as strings.

The following examples will use FQDNs as subjectAltNames.

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.

These keys and certificates should then be moved to their appropriate directories. The default locations for the keys and certificates are as follows:

`/etc/isakmpd/ca`	public-key (PEM-format) certificates of trusted CAs
`/etc/isakmpd/certs`	trusted public-key (PEM-format) certificates (with subjectAltName extensions)
`/etc/isakmpd/private`	Private Keys. These should have corresponding public keys in the `cert` directory, above

These locations may be overridden by values in the [X509-certificates] section of isakmpd.conf(5).

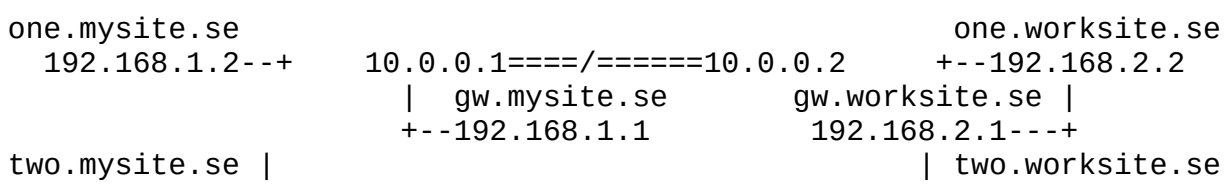
Note that if the CA infrastructure is to be trusted, the CA private key (`/etc/isakmpd/ca/ca.key`) should be kept in a safe place, should be of an appropriate bit length (i.e. 2048-bit), and should be encrypted with a strong passphrase.

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 has been heavily modified. I will also use a much shorter file here than the example file in the man page to make it easier to understand. 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 exist as far as I know.

Actually since everyone who reads this already has a working configuration for the pre-shared 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



```
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/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 section names 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 pre-shared key case.
[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]
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 of course).
# 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/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 do not 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 are 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 snake oil today, it is left as an exercise for the reader to change this policy into not allowing DES either). Note that anyone who does encrypts 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 pre-shared key case. In that case, anyone who knows the pre-shared 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:

```
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 sub-licensed 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:\
        MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\
        BAMTCUlLRUxBQiBDQTAeFw0wMDAxMjgxNzQyMTZaFw0wMTAxMjcxNzQyMTZaMCEx\
        CzA           This is would be a user certificate           AQEB\
        BQA                                                         IUuz\
        eOW8P5UGJUH2JVkiA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
        ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjpUi3AgMB\
        AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMP1ad+\
        iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaai\
        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:\
        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\
        eOW8P5UGJUH2JVkiA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
        ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjpUi3AgMB\
        AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMP1ad+\
        iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaai\
        NuzgmQ9hh3GifEgN9VEiC3j4kTytonKr0Q+vTLM7xYzheOxvrtUErRwZ9Xs1KzHe\
        yiXHSU8="
licensees:  "x509-base64:\
        MIIBsTCCARoCAQAwDQYJKoZIhvcNAQEEBQAwITELMAkGA1UEBhMCc2UxEjAQBgNV\
        BAMTCUlLRUxBQiBDQTAeFw0wMDAxMjgxNzQyMTZaFw0wMTAxMjcxNzQyMTZaMCEx\
        CzA      This is would be the key of the subject of the     AQEB\
        BQA                          certificate                    IUuz\
        eOW8P5UGJUH2JVkiA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
        ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjpUi3AgMB\
        AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMP1ad+\
        iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaai\
        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 sub-licensing 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\
        BAMTCUlLRUxBQiBDQTAeFw0wMDAxMjgxNzQyMTZaFw0wMTAxMjcxNzQyMTZaMCEx\
        CzA              This would be the CA certificate          AQEB\
        BQA                                                            IUuz\
        eOW8P5UGJUH2JVkiA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
        ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjpUi3AgMB\
        AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMP1ad+\
        iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaai\
        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 were 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 (e.g. 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 an 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\
        eOW8P5UGJUH2JVkiA2CTDryFf0CHYwd2P003dtVYw5RvET7XLMpRZiCcWtBdxneW\
        ct+016zUBP/cQMMl+KownxAUq9ezA8GvTyUWC97SOMOgoVj/QR3FHmEjpUi3AgMB\
        AAEwDQYJKoZIhvcNAQEEBQADgYEALGShaAxHvGncev0iFnKrJI4x5T4vlaMP1ad+\
        iWLV5q9H3wickVGN0NPerq0YLwx/VA9WaecYN8V+ALtNKYPuDiT11zwvE8GQeaai\
        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, e.g. 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 is 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 authorization.

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 "authorization 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 e.g. 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 organization 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).

13.9 - 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 is actually based on the OpenBSD isakmp daemon.

The following MS-Windows clients have been reported to be compatible:

- Ashley Laurent VPN software
- PGP VPN software
- Cisco IRE client
- Microsoft Windows 2000, XP

The following gateways/routers have been reported to be compatible:

- Cisco IOS
- Cisco PIX
- Intel LanRover
- Cendio Fuego
- KAME for FreeBSD
- FreeS/WAN for Linux
- Symantec Raptor
- Ericsson eBox
- F-Secure VPN+
- Teamware TWISS
- 3com Pathbuilder
- Nortel Contivity
- CheckPoint FW-1
- Watchguard Firebox III
- Lucent Access Point

13.10 - 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 set up 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:

  ```
  # 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:

  ```
  # 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
```

- To turn on all useful debugging in isakmpd, start it as

```
# /sbin/isakmpd -d -DA=90
```

  or (to skip the most detailed timer debug info)

```
# /sbin/isakmpd -d -DA=90 -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. Tcpdump 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.

```
$ 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.

13.11 - 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), 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
- 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
- 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 2631 - Diffie-Hellman Key Agreement Method
- RFC 2709 - Security Model with Tunnel-mode IPsec for NAT Domains

[FAQ Index] [To Section 12 - For Advanced Users] [To Section 14 - Using disks in OpenBSD]


www@openbsd.org

$OpenBSD: faq13.html,v 1.75 2003/05/02 15:41:59 nick Exp $

OpenBSD

[FAQ Index] [To Section 13 - IPsec]

14 - Disk Setup

Table of Contents

Using OpenBSD's disklabel

Table of Contents

What is disklabel?

First be sure to read the disklabel(8) man page.

Disklabels 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 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 FAQ 4, Space Needed. 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. separating /,  /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 users 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 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 man 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, 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)
        setpid          Set the identifier of a given table entry
        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.
- **setpid** Change the partition identifier of the given partition table entry. This command is particularly useful for reassigning an existing partition to OpenBSD.
- **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. In the below example we're adding a third SCSI drive to the system.

```
# 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).

```
# newfs sd2a
```

Or whatever your disk was named as per OpenBSD's disk numbering scheme. (Look at the output from dmesg(8) 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/sd2a /u
```

Finally, add it to /etc/fstab(5).

```
/dev/sd2a /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 -pdum` to copy /usr/local to the /mnt directory. Edit the /etc/fstab(5) file to show that the /usr/local partition is now /dev/sd2a (your freshly formatted partition.) Example:

```
/dev/sd2a /usr/local ffs rw 1 1
```

Reboot into single user mode with **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 voila, 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).)

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 space setup. The file must not reside on a filesystem which has SoftUpdates enabled (they are disabled by default). To start out, you can see how much swap you currently have and how much you are using with the swapctl(8) utility. You can do this by using the command:

```
$ 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.

```
$ 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

```
$ sudo chmod 600 /var/swap
$ 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.

```
$ 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.

```
$ cat /etc/fstab
/dev/wd0a / ffs rw 1 1
/var/swap /var/swap 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.

```
$ 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.

```
$ 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(8).

```
$ 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.

```
$ sudo swapctl -a /dev/vnd0c
$ 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.

As Soft Updates are still in development as a whole, an fsck(8) is still needed after the computer is abruptly turned off without a clean shutdown sequence, but this will be remedied in future versions.

More internals and details about Soft Updates can be found in the papers of Ganger and Patt and from McKusick.

To use Soft Updates, your kernel must have

option FFS_SOFTUPDATES

compiled in, this is already in place on GENERIC.

Enabling soft updates must be done with a mount-time option. When mounting a partition with the mount(8) utility, you can specify that you wish to have soft updates enabled on that partition. Below is a sample */etc/fstab(5)* entry that has one partition *sd0a* that we wish to have mounted with soft updates.

```
/dev/sd0a / ffs rw,softdep 1 1
```

Note to sparc users: Do not enable soft updates on sun4 or sun4c machines. These architectures support only a very limited amount of kernel memory and cannot use this feature. However, sun4m machines are fine.

14.6 - When I boot after installation of OpenBSD/i386, it stops at "Using Drive: 0 Partition: 3" - i386 specific.

This isn't actually an error message by itself, it is the boot loader in the MBR telling you which drive and partition it is about to boot from. The problem is, the boot process stopped here.

There are two common reasons why this may happen: an incompatibility between the BIOS and the OpenBSD MBR, or a drive geometry problem. An example of a drive geometry problem would be if you were to move a drive from one computer to another, update a BIOS or change a BIOS setting on a computer, though it is also reported that it can happen for unknown reasons during install.

Note: As of OpenBSD 3.1, the BIOS compatibility problem should be resolved on virtually all computer systems. However, it is still possible to have a drive geometry problem, so it is still possible to have the system hang after this message.

To fix the BIOS compatibility issue, you must replace the boot loader with one that is compatible with your system. Fortunately, boot loaders are easy to come by.

Installing the BootEasy boot loader:

This requires booting your system. As you can't boot off the hard disk directly, we have to use a boot floppy or CD-ROM to start the boot process. When the system gets to the 'boot>' prompt, redirect it to boot from the hard disk:

```
reading boot.......
probing pc0 com0 com1 pci mem [639k 79m a20=on]
disk hd0 fd0
>> OpenBSD BOOT 1.28
boot> boot hd0a:/bsd
```

This command redirects the boot process to the file /bsd on the 'a' partition of the first hard drive, allowing your system to

boot.

Once the system is booted, you need to install the BootEasy boot loader. The file itself can be found on both the CD-ROM and the FTP sites, in the directory `3.3/tools/booteasy/Boot.bin`, and can be installed using the following fdisk(8) command:

```
# fdisk -i -f /mnt/3.3/tools/booteasy/Boot.bin wd0
```

This assumes you have the 3.3 CD mounted on /mnt and that you are using an IDE drive. You may need to change this depending upon where Boot.bin is and what kind of hard disk you have (a SCSI drive would typically be 'sd0'). **Note: do NOT do this if your OpenBSD partition is not the entire disk!** Initializing the MBR this way creates one OpenBSD partition spanning the entire disk and deletes all other partitions, which is rarely good.

BootEasy has another feature that might interest you even if you have no problem with the default loader: It is capable of selecting a boot partition at startup -- it will prompt you for which partition to boot from, and will boot the active partition if no other choice is made. This might be very useful should you have multiple OS's on one disk. Installing BootEasy this way is done from MS-DOS with the BOOTINST.EXE program found in the BootEasy directory on the CD-ROM and FTP servers.

Installing the MS-DOS boot loader:

Boot from a Windows 9x or DOS v6 boot disk with FDISK.EXE on it. Once the system is booted to an MS-DOS prompt, enter the following:

```
A:\>fdisk /mbr
```

You should see a brief disk access, and then a command prompt should return, with NO message of any kind. "Bad command or file name" means the disk you used did not have FDISK.EXE on it. If you do this properly, the "Using ..." message will be removed, as it replaces the very code that produces that message. If you had a BIOS compatibility issue, it will now be gone, reboot, your OpenBSD install should come right up.

It has been reported that this also works with FreeDOS.

The OS-BS Boot Loader:

Another boot loader, OS-BS, is included with the OpenBSD CD-ROMs and available on the FTP sites in 2.9/tools/osbs135.exe. The OS-BS web page is at http://www.prz.tu-berlin.de/~wolf/os-bs.html.

LILO:

The Linux LILO program can also be used. For details, see INSTALL.linux

Avoiding the problem:

Relatively few machines have the BIOS compatibility problem, but if you are setting up a machine that you know has this problem, it is fairly easy to avoid it. The only time the OpenBSD loader is installed on your system during a normal install is when you say 'Y' to the 'Use entire disk for OpenBSD' question. If you answer 'N' and manually create the OpenBSD disk partition, it will not replace the boot loader unless you use the 'reinit' or 'update' commands of fdisk. This, of course, assumes that your drive STARTED with some kind of boot loader -- if it doesn't (new, blank drive, drive pulled from another platform), you will have to install one before the system will boot.

Fixing a drive geometry problem:

Ideally, you would want to avoid this problem by maintaining the same drive geometry, not by fixing it, however sometimes you can't avoid it. An example would be moving a large drive from an old computer which didn't support LBA geometry to a new machine which insists upon using LBA.

Start your machine using a boot disk or CD-ROM, as indicated above. Log in as root, and execute the following commands:

```
# cp /usr/mdec/boot /boot
```

```
# /usr/mdec/installboot -v /boot /usr/mdec/biosboot wd0
```

Reboot, your system should come right up.

installboot(8) installs and configures the partition boot loader, biosboot(8), which loads boot(8). boot(8) is the module which loads the kernel into RAM. biosboot(8) has a table within it that points to the physical location (according to the system's BIOS) of boot(8). If you do anything which changes the BIOS's perception of the location of boot(8), you must re-run installboot(8) as above to reinitialize the table pointing to boot(8).

See Install Boot for more info.

14.7 - What are the issues regarding large drives with OpenBSD?

OpenBSD has support for file systems of sizes much larger than any currently or soon to be available hard disks, however there are limitations on some interfaces which are smaller than the theoretical maximum of OpenBSD. In the case of IDE drives, the limit is 128GB, the limit of the currently popular ATA interface. Note the next generation of ATA drives, those with capacities greater than 128G (1G=2^30 here, not 1,000,000,000, so drive manufacturers will often call this 137G), are not supported by OpenBSD 3.1 and earlier.

Unfortunately, the full ability of the OS isn't available until AFTER the OS has been loaded into memory, and the booting process introduces limits of its own. The boot process has to utilize (and is thus limited by) the system's boot ROM. The OpenBSD i386 boot loaders (biosboot(8) and boot(8)) also have their own internal 8G limitation, from an older BIOS limit.

For this reason, the entire /bsd file (the kernel) must be located on the disk within the boot ROM addressable area, or within the first 8G of the disk, whichever is smaller. This means that on some older i386 systems, the root partition must be completely within the first 504M, but for most newer computers, the root partition may be anywhere within the first 8G.

Note that it is possible to install a 40G drive on an old 486 and load OpenBSD on it as one huge partition, and think you have successfully violated the above rule. However, it might come back to haunt you in a most unpleasant way:

- You install on the 40G / partition. It works, because the base OS and all its files (including /bsd) are within the first 504M.
- You use the system, and end up with more than 504M of files on it.
- You upgrade, build your own kernel, whatever, and copy your new /bsd over the old one.
- You reboot.
- You get a message such as "bad magic"

Why? Because when you copied "over" the new /bsd file, it didn't overwrite the old one, it got relocated to a new location on the disk, probably outside the 504M range the BIOS supported. The boot loader was unable to fetch the file /bsd, and the system hung.

To get OpenBSD to boot, /bsd must be within the boot ROM's supported range. To play it safe, the rule is simple:

The entire root partition must be within the computer's BIOS (or boot ROM) addressable space or within the first 8G, whichever is smaller. There are no ways around this at this time. Trust us. You *must* follow this rule.

This is another good reason to partition your hard disk, rather than using one large partition.

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 drive (or use a bootable CDROM) 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(8) command at your shell prompt. You should get output similar to this:

```
# 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:

```
# 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(8) 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(1) 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: <ARCHIVE, Python 28388-XXX, 5.28>
```

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 -tvs 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 -tvs 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 ther fdisk(8)r command, create a primary OpenBSD partition on this newly installed drive. Example:

  ```
  # 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:

  ```
  # 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:

  ```
  # newfs /dev/rsd0a
  # newfs /dev/rsd0h
  ```
- Mount your newly prepared root (/) file system on /mnt. Example:

  ```
  # mount /dev/sd0a /mnt
  ```
- Change into that mounted root file system and start the restore process. Example:

  ```
  # cd /mnt
  # 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:

```
# 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:

```
# cp /usr/mdec/boot /mnt/boot
# /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:

```
# umount /mnt
# 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:

```
# mount -u -w /
```

- Once you have remounted in r/w mode you can continue restoring your other file systems. Example:

```
(simple mount table)
# mount /dev/sd0h /usr; cd /usr; restore -rs 2 -f /dev/rst0

(more advanced mount table)
# mount /dev/sd0d /var; cd /var; restore -rs 2 -f /dev/rst0
# mount /dev/sd0e /home; cd /home; restore -rs 3 -f /dev/rst0
# 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.

14.10 - Mounting disk images in OpenBSD

To mount a disk image (ISO images, disk images created with dd, etc) in OpenBSD you must configure a vnd(4) device. For example, if you have an ISO image located at */tmp/ISO.image*, you would take the following steps to mount the image.

```
# vnconfig svnd0 /tmp/ISO.image
# mount -t cd9660 /dev/svnd0c /mnt
```

Notice that, since this image is a CD image you must specify type of *cd9660* when mounting it. This is true, no matter what type, e.g. you must use type ffs when mounting disk images.

To unmount the image use the following commands.

```
# umount /mnt
# vnconfig -u svnd0
```

For more information, refer to the vnconfig(8) man page.

14.11 - Help! I'm getting errors with PCIIDE!

PCI IDE DMA is unreliable with many combinations of hardware. Until recently, most "mainstream" operating systems that claimed to support DMA transfers with IDE drives did not ship with that feature active by default.

OpenBSD is aggressive and attempts to use the highest DMA Mode it can configure. This will cause corruption of data transfers in some configurations because of buggy motherboard chipsets, buggy drives, and/or noise on the cables. Luckily, Ultra-DMA modes protect data transfers with a CRC to detect corruption. When the Ultra-DMA CRC fails, OpenBSD will print an error message and try the operation again.

```
wd2a:  aborted command, interface CRC error reading fsbn 64 of 64-79 (wd2 bn 127; cn
0 tn 2 sn 1), retrying
```

After failing a couple times, OpenBSD will downgrade to a slower (hopefully more reliable) Ultra-DMA mode. If Ultra-DMA mode 0 is hit, then the drive downgrades to PIO mode.

If OpenBSD does not successfully downgrade, or the process causes your machine to lock hard, please send in a bug report.

14.12 - 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
```

This means that pciide0, channel 1, drive 0 (which appears to be an ATAPI CD-ROM) is using DMA data transfers.

14.13 - RAID options for OpenBSD

RAID (Redundant Array of Inexpensive Disks) gives an opportunity to use multiple drives to give better performance, capacity and/or redundancy than one can get out of a single drive alone. While a full discussion of the benefits and risks of RAID are outside the scope of this article, there are a couple points that are important to make here:

- RAID has nothing to do with backup.
- By itself, RAID will not eliminate down-time.

If this is new information to you, this is not a good starting point for your exploration of RAID.

Software Options

OpenBSD includes RAIDframe, a software RAID solution. Documentation for it can be found in the following places:

- FAQ 11, RAID
- RAIDframe Homepage
- man page for raidctl(8)
- man page for raid(4)

Starting with OpenBSD 3.1, the root partition can now be directly mirrored by OpenBSD using the "Autoconfiguration" option of RAIDframe. Previous releases of OpenBSD could not have the root partition mirrored this way.

Hardware Options

Many OpenBSD platforms include support for various hardware RAID products. The options vary by platform, see the appropriate hardware support page (listed here).

Another option available for many platforms is one of the many products which make multiple drives act as a single IDE or SCSI drive, and are then plugged into a standard IDE or SCSI adapter. These devices can work on virtually any hardware platform that supports either SCSI or IDE.

Some manufacturers of these products:

- Arco
- Maxtronic
- Infortrend

(Note: these are just products that OpenBSD users have reported using -- this is not any kind of endorsement, nor is it an exhaustive list.)

Non-Options

An often asked question on the mail lists is "Is the Promise or HighPoint IDE RAID controller supported?". The answer is "No". These cards and chips are not true hardware RAID controllers, but rather BIOS-assisted boot of a software RAID. As OpenBSD already supports software RAID in a hardware-independent way, there isn't much desire among the OpenBSD developers to implement special support for these cards.

[FAQ Index] [To Section 13 - IPsec]


www@openbsd.org

$OpenBSD: faq14.html,v 1.83 2003/05/01 01:47:41 nick Exp $