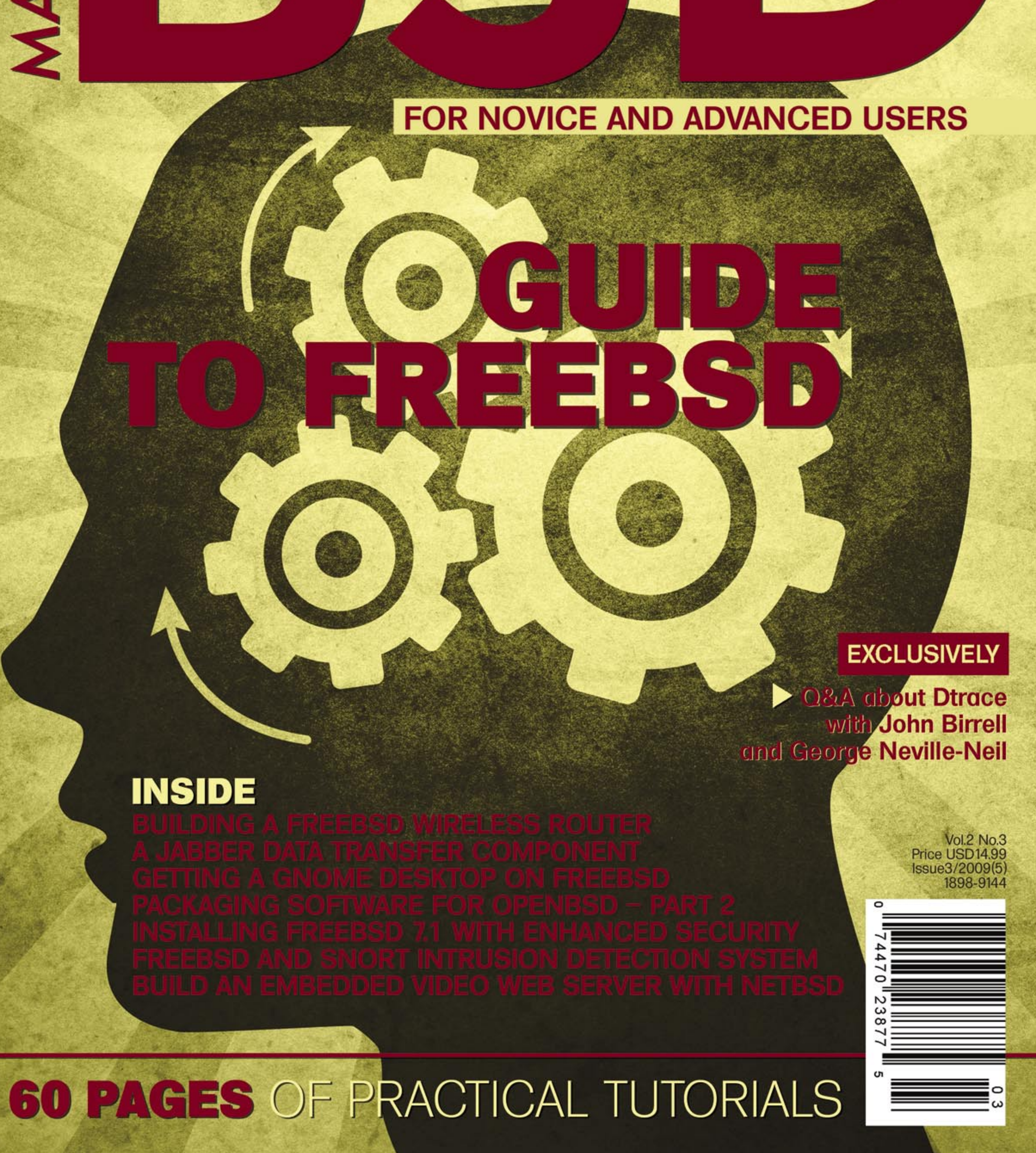
FREE DVD INSIDE FREEBSD 7.1 32BIT

MAGAZINE BSD

FOR NOVICE AND ADVANCED USERS



GUIDE TO FREEBSD

EXCLUSIVELY

▶ Q&A about Dtrace with John Birrell and George Neville-Neil

INSIDE

BUILDING A FREEBSD WIRELESS ROUTER
A JABBER DATA TRANSFER COMPONENT
GETTING A GNOME DESKTOP ON FREEBSD
PACKAGING SOFTWARE FOR OPENBSD – PART 2
INSTALLING FREEBSD 7.1 WITH ENHANCED SECURITY
FREEBSD AND SNORT INTRUSION DETECTION SYSTEM
BUILD AN EMBEDDED VIDEO WEB SERVER WITH NETBSD

Vol.2 No.3
Price USD14.99
Issue3/2009(5)
1898-9144

60 PAGES OF PRACTICAL TUTORIALS

PC-BSD
MARS
FreeBSD®
iX systems
PC-BSD®

PC-BSD GALILEO AND GALACTIC TECH SUPPORT

The people at iXsystems may not do a whole lot in the way of observational astronomy, but they do appreciate Galileo's intrepid work to change the face of science. In that spirit, PC-BSD hopes to change the face of the Operating System world. As Galileo reduced complex problems to a simple set of terms on the basis of everyday experience and common-sense logic, PC-BSD translates computer processes that would be difficult for the casual user into simple, intuitive interfaces. iXsystems also offers technical expertise and unparalleled industry support for the FreeBSD and PC-BSD platforms, which will make your experience using Galileo out of this world.

Built on a FreeBSD 7.1 core, PC-BSD Version 7.1 Galileo Edition brings stability, security and ease of use to the desktop and the server through its use of the KDE 4.2 desktop and self-installing software packages, as well as graphical system administration tools. PC-BSD Version 7.1 Galileo Edition has hundreds of programs available for download on http://www.pbidir.com as well as on the second installation disc, covering a wide array of functions, from graphical editing programs to office oriented software. Galileo Edition also has full proxy support for PBI and system updates, thin client server support, drastically improved nVIDIA performance and speed increases, and a new backup and recovery tool.

Getting PC-BSD up and running is fast and easy, but having expert help on-hand to solve your problems can take your computing experience to new heights. From optimizing your small office set-up to guidance on large-scale deployments, the iXsystems team can ensure you get the most from your PC-BSD and FreeBSD systems.

technical expertise

When you sign on for iXsystems Professional Services you get a team of PC-BSD and FreeBSD experts. iXsystems partners with the most advanced developers and long-time contributors from the BSD communities to offer custom development and advanced level support and consulting services.

large rollouts and migrations

The Professional Services Team provides installation support for large networks. Our technicians will work with you to determine your operational needs and set up any number of desktops and servers. Our experts can also provide specialized support to your system administrators.

custom pbi creation and application installation

PC-BSD uses a graphical utility, known as PBIs (push button installers), to remove and install software. These PBIs are self-contained and contain their own libraries, eliminating the problem of shared dependencies. The experts at iXsystems are well versed in compiling software applications into PBIs for use on PC-BSD. On the occasion where a program needed to run on Galileo is unavailable, the Professional Services Team can develop a push button installer application. If needed, these applications can be deployed over multiple desktops by our technicians.

escalation management

iXsystems is the all-around FreeBSD company that builds FreeBSD-certified servers and storage solutions, runs the FreeBSD Mall, and is the corporate sponsor of the PC-BSD Project. When the iXsystems Service Support Team encounters a confirmed bug, we can escalate the bug to the FreeBSD engineering team. We can also work with The FreeBSD Project to create and submit patches to the FreeBSD community for possible inclusion in the latest release.

Be a part of our efforts to change the face of the Operating System world. Contact iXsystems at (408)943-4100 or visit our website at http://ixsystems.com/support/professional-bsd-support.html and fill out the Inquiry form for more information. We will pair you up with an Account Management Service Professional that can assess your needs and create a custom FreeBSD or PC-BSD support plan for your organization!

Editor's Note

Dear All,

It is already the fifth issue of BSD magazine and we all hope there will be another 50...and more! I hope you will like the refreshed layout of the magazine – short news and a little reorganization of the content. Hopefully, you will find it useful and helpful in your journey to BSD world.

This issue is devoted to FreeBSD distribution. We did our best to cover the most interesting and useful topics in form of a step-by-step tutorials, so that everybody can take the chance do it.

For beginners we prepared the article describing the process of FreeBSD 7.1 installation and configuration. In the how-to's section we covered topics like OpenSMTPD, GNOME desktop on FreeBSD, packaging software, Jabber server, building wireless router, CPU scaping and much more.

In security corner you will find articles devoted to LDAP authentication and Snort Intrusion Detection Scanner. For those of you, who are interested in multimedia on BSD systems, Donald T. Hayford wrote a great article on building an embedded video web server. We also included lots of tips&trick by Dru Lavigne and Mikel King.

As always, we are waiting for your comments, replies, ideas and suggestions. If you would like to become BSD author or betatester, don't hesitate- keep the mails coming in!

Enjoy!

[illegible]

Karolina Lesińska
Editor in Chief

MAGAZINE BSD

Editor in Chief: Karolina Lesińska
karolina.lesinska@bsdmag.org

Contributing: Remko Lodder, Gilles Chehade, Jan Stedehouder, Edd Barrett, Eric Schnoebelen, Eric Vintimilla, Vermaden, Nicolas Greneche, Svetoslav P. Chukov, Donald T. Hayford, Mikel King, Dru Lavigne, Federico Biancuzzi

Art Director: Agnieszka Marchocka
DTP Technician: Ireneusz Pogroszewski
Przemysław Banasiewicz

Senior Consultant/Publisher:
Paweł Marciniak *pawel@software.com.pl*

National Sales Manager: Ewa Dudzic *ewa.dudzic@bsdmag.org*
Marketing Director: Ewa Dudzic *ewa.dudzic@bsdmag.org*
Executive Ad Consultant:
Karolina Lesińska
karolina.lesinska@bsdmag.org
Advertising Sales: Karolina Lesińska
karolina.lesinska@bsdmag.org

Production Director: Marta Kurpiewska

Publisher :
Software Wydawnictwo Sp.z.o.o
02-682 Warszawa, Bokserska 1
Poland
worldwide publishing

Postal addres:
Software Media LLC
1521 Concord Pike, Suite 301
Brandywine Executive Center
Wilmington, DE 19803
USA
tel: 1 917 338 36 31
www.bsdmag.org

Software-Wydawnictwo Sp z o.o. is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: *editors@bsdmag.org*
Print: 101 Studio, Printed in Poland

Distributed in the USA by: Source Interlink Fulfillment Division, 27500 Riverview Centre Boulevard, Suite 400, Bonita Springs, FL 34134 Tel: 239-949-4450.

All trade marks presented in the magazine were used only for informative purposes. All rights to trade marks presented in the magazine are reserved by the companies which own them.

The editors use automatic DTP system AURUS

Mathematical formulas created by Design Science MathType™.
DVDs tested by AntiVirenKit GDATA Software Sp. z o.o.

Subscription Agency:
EMD The Netherlands - Belgium
P.O. Box 30157
1303 AC Almere
The Netherlands

Phone + 31 (0) 36 5307118
Fax + 31 (0) 36 5407252

Installing FreeBSD 7.1 with Enhanced Security (Jails)

Remko Lodder

This article will guide people that are new to FreeBSD on installing the software and enhancing it's security by setting up FreeBSD jails that will give service to for example an webserver.

We will begin by fetching the installation media, then using the media to do the installation, we will upgrade our system and place the foundation for our jails, and finally setup the required infrastructure around it. Advanced users that are already familiar with how FreeBSD works, might benefit from the Jails paragraph which you can find below.

Obtaining the installation media

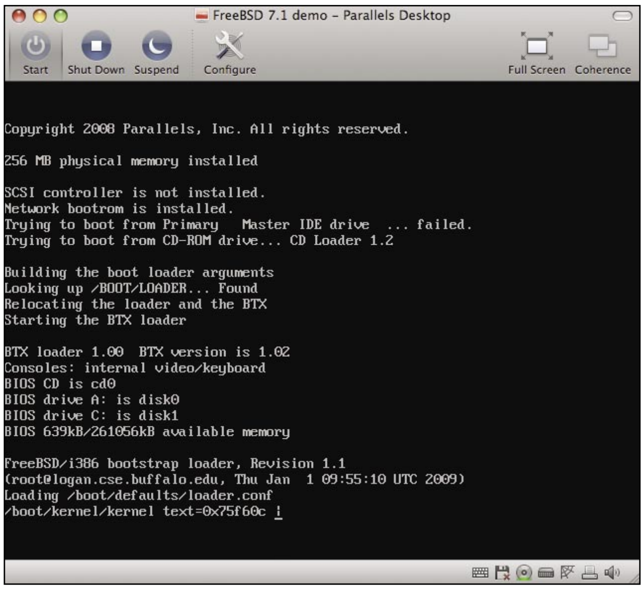
So we decided to give FreeBSD a try, good! But how do we get it running? Lets visit the FreeBSD website at *http://www.freebsd.org/*. We will see the index page of the website, showing a big yellow *Get FreeBSD now* button. If we click on this we will navigate to a new page, showing the available downloads. For example the *7.1-i386 ISO* images are available on: *ftp://ftp.FreeBSD.org/pub/FreeBSD/releases/i386/ISO-IMAGES/7.1/*, but it might be inefficient to download the files using the main FreeBSD FTP server. There are localized FTP Servers probably also near you which can be used to get the media. The localized FTP Servers are available through `ftp://ftp.<countrycode>.FreeBSD.org/pub/FreeBSD/releases/i386/ISO-IMAGES/7.1` which in my case would be: *ftp://ftp.nl.FreeBSD.org/pub/FreeBSD/releases/i386/ISO-IMAGES/7.1/*. On the FTP Server there is a list of downloadable files like: (see Listing 1). Personally I always download the disc1.iso file, this file delivers me the standard installation and does not require that I have internet-access (which is the case for the bootonly ISO). If you download the disc1 ISO file you can get a rapid installation, which takes less then 15 minutes in my case.

Preparing the installation media

We have fetched the required ISO (either CD or DVD, depending on your wishes) and need to put it on CD. If you have a burner you mostly get software with it. Given that we do the FreeBSD installation for the first time, I'll assume that there is some kind of operating system already running on it, which should support the burning of ISO files. Within Windows, if the software is correctly installed, you can double click on the ISO file after which it will burn the contents of the ISO file in a pre-determined format to the CD or DVD. Please check whether the CD or is readable before restarting the machine. You can do that by inserting the CD or DVD and navigating through it's directory structure.

Starting the installation

Now that the CD had been burned we can boot from it. The installation CD is configured to startup the installation application immediatly so no further actions are required to get the installer going.



```
Copyright 2008 Parallels, Inc. All rights reserved.

256 MB physical memory installed

SCSI controller is not installed.
Network bootrom is installed.
Trying to boot from Primary   Master IDE drive   ... failed.
Trying to boot from CD-ROM drive... CD Loader 1.2

Building the boot loader arguments
Looking up /BOOT/LOADER... Found
Relocating the loader and the BTX
Starting the BTX loader

BTX loader 1.00  BTX version is 1.02
Consoles: internal video/keyboard
BIOS CD is cd0
BIOS drive A: is disk0
BIOS drive C: is disk1
BIOS 639kB/261056kB available memory

FreeBSD/i386 bootstrap loader, Revision 1.1
(root@logan.cse.buffalo.edu, Thu Jan  1 09:55:10 UTC 2009)
Loading /boot/defaults/loader.conf
/boot/kernel/kernel text=0x75f60c |
```

Figure 1. Boot initial

Configuring during the installation

The installation CD brings you to the installer, which is the heart of the installation that we will be doing. The first window that we will be seeing is the main screen. Here *sysinstall* (in case you want to return to the application sometime later) gives you the option to do various configurations and select an installation type.

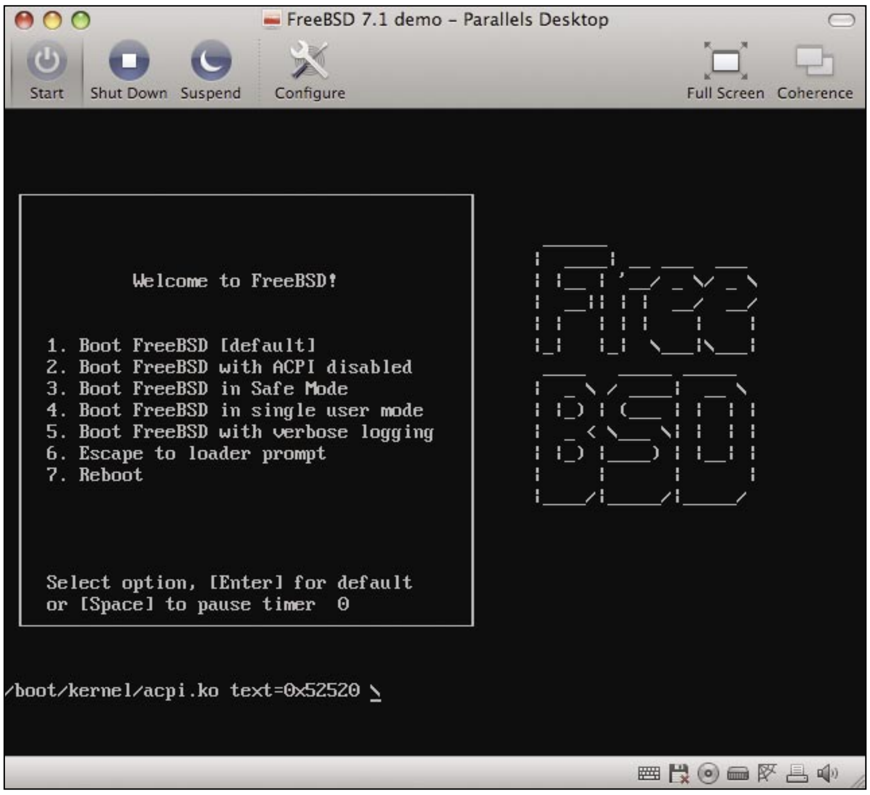
We will start by picking the *Standard* option, which is the most convient for new users. The installer will tell us that the upcoming screen will help us setting up a partition scheme for the disk that we will be using. By clicking [*OK*] we will proceed to the next window, sometimes this window is an alert that the geometry is not the same as being advertised, but I never had problems even with this warning.

We click on [*OK*] again and the *fdisk* screen will popup. Do not be afraid, you do not have to understand this part. Since we will be installing FreeBSD we will use the entire disk (if you are not sure whether you want to do that, you can always install this in a virtual machine, use a seperated disk, or read the handbook for detailed instructions on how to create multi-boot instances) which makes it easy to get going. Press the [*A*] button and a disk layout will be automatically created. Navigate to the middle line (Which is the entire disk that we just created) and Press the [*S*] button. This will mark the partition as active and makes sure we can boot from the disk.

Then the installer will ask us where we want to install the boot manager, do we want to place it on the partition root, or do we want to place it on the MBR (which is the super root of the disk). We will select the BootMgr and continue to the next screen. The installer continues by loading the disklabel editor. The disklabel editor is used to *configure* the partition, it enables you to select what space you want to assign where, so that you can limit the resources. Since we do not want to think much about this, we press the [*A*] button and the space is being distributed via a standard scheme (a formula is behind that, so that you will always get the best possible solution). Continue by pressing the [*Q*] button.

So we know how the disk layout is going to be, we have made up a scheme to see where we are going to put our space, but we didn't select what to install yet. Mostly I pick the *Minimal* installation. This is the one that is being done the quickest and has maximum flexibility in the future. Of course you are free to select the option you prefer instead.

After selecting the distribution type, the installer will ask us from which part we want to install the distribution. If you burned the CD1 image or the DVD image, you can select CD/DVD here, else select the network installation (this is not covered in this article). The installer will dump the contents of the CD on the disk layout as we defined it, and will continue with question whether we want to setup an ethernet or SLIP device. The latter



```
Welcome to FreeBSD!

1. Boot FreeBSD [default]
2. Boot FreeBSD with ACPI disabled
3. Boot FreeBSD in Safe Mode
4. Boot FreeBSD in single user mode
5. Boot FreeBSD with verbose logging
6. Escape to loader prompt
7. Reboot

Select option, [Enter] for default
or [Space] to pause timer  0

/boot/kernel/acpi.ko text=0x52520 \
```

Figure 2. Bootup install

Listing 1. Files on the FTP

```
ftp> ls
229 Entering Extended Passive Mode (|||64964|)
150 Here comes the directory listing.
-rw-r--r--    1 500      450      37826560 Jan 02 23:22 7.1-RELEASE-i386-
bootonly.iso
-rw-r--r--    1 500      450      578529280 Jan 02 23:22 7.1-RELEASE-i386-
disc1.iso
-rw-r--r--    1 500      450      556648448 Jan 02 23:21 7.1-RELEASE-i386-
disc2.iso
-rw-r--r--    1 500      450      611702784 Jan 02 23:20 7.1-RELEASE-i386-
disc3.iso
-rw-r--r--    1 500      450      301914112 Jan 02 23:19 7.1-RELEASE-i386-
docs.iso
-rw-r--r--    1 500      450      1895702133 Jan 02 23:19 7.1-RELEASE-i386-
dvd1.iso.gz
-rw-r--r--    1 500      450      231577600 Jan 02 23:15 7.1-RELEASE-i386-
livefs.iso
-rw-r--r--    1 500      450           478 Jan 02 23:15 CHECKSUM.MD5
-rw-r--r--    1 500      450           723 Jan 02 23:15 CHECKSUM.SHA256
```

probably doesn't make sense to you but perhaps Ethernet does. Ethernet is the *standarized* cable that most people plug in their machines and have internet access. Select [*YES*] to configure the Ethernet device.

Now that we have selected yes, a new window appears with various interfaces, select the one that sounds like your internet-facing card. This sounds a bit vague, but in case you have one controller there are only four options, three of them being: PLIP, SLIP and PPP, which are most likely not the ones we need to configure here.

Select the driver that remained, and we will be asked to do autoconfiguration of the interface by using `IPv6`. Since we do not understand this we select [*NO*]. Most networks in home environments are setup to use automatic configuration using DHCP. This is our next option, so we will select [*YES*] here. Now that we have selected this, a new window appears with advanced information about an address that we obtained.

The window expects us to give a name to the machine, select a name that you favor and navigate to the OK button and continue. The hostname is setup as `yourname.yourdomainname.extention`. My machines are all named after elves from the *Elvandar* series so their name is:

```
<name of elf>.elvandar.org
```

Which makes me able to easily spot which machine is which. After doing this configuration, we will return to the main screen, where we can select *Exit Installation*. This will reboot our machine with the fresh installation on it.

Rebooting the machine for the first time

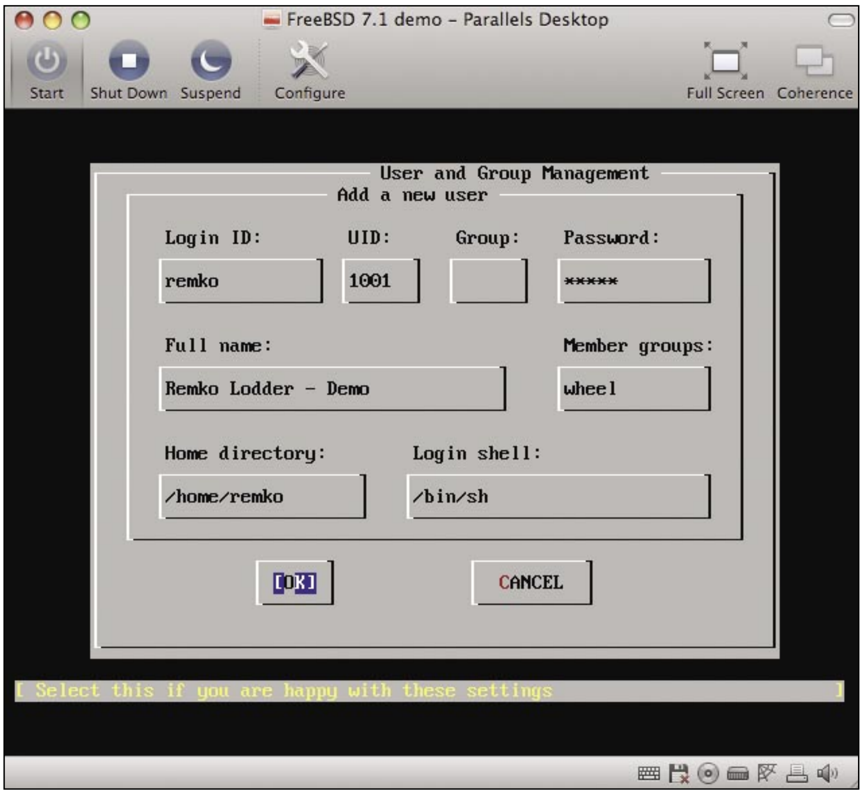
After the configuration had been completed and we exit from the *sysinstall* application, the machine will reboot so that it will be prepared for it's first use.

Retrieving the latest source code tree for FreeBSD 7.1

So, now that we have installed the default installation of FreeBSD and restarted it so that all required services are started, it's time to make sure our system is as up to date as possible. While there are multiple ways to do this, I will use the *csup* method to retrieve the latest source code for the 7.1-RELEASE branch. We will compile this into a new version for our machine (including the latest security patches) and it will finally form the foundation for our Jail Infrastructure.

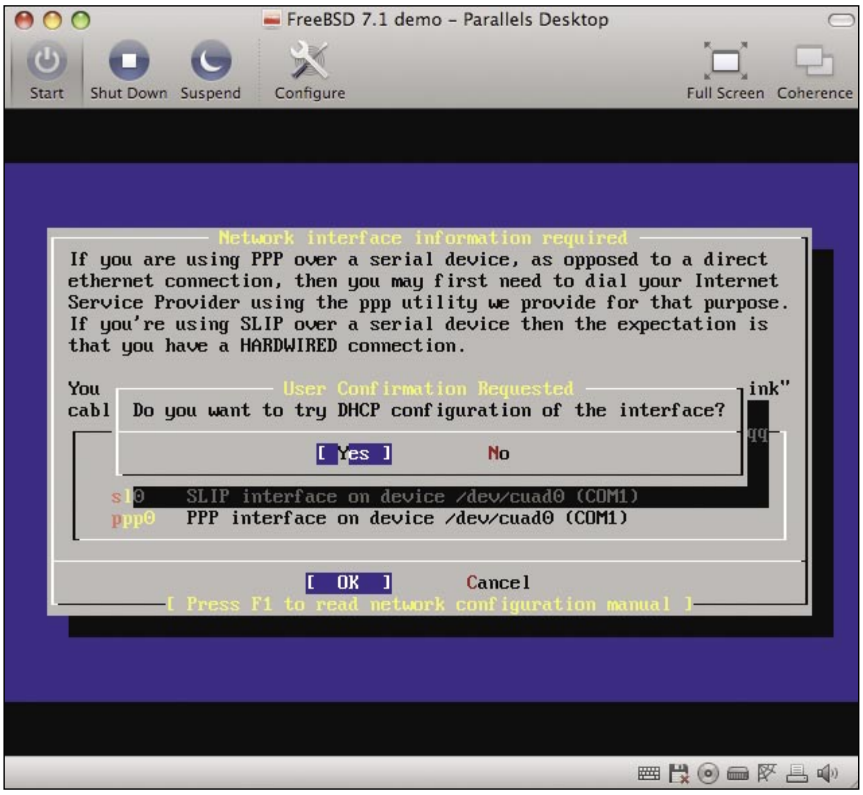
Logon to the system and switch user to `root`.

```
% su -
Password:
#
```



```
                 User and Group Management
                     Add a new user

   Login ID:      UID:     Group:     Password:
   remko          1001                *****

   Full name:                         Member groups:
   Remko Lodder - Demo                wheel

   Home directory:        Login shell:
   /home/remko            /bin/sh

          [OK]              CANCEL

[ Select this if you are happy with these settings ]
```

Figure 3. Install account creation



```
         Network interface information required
 If you are using PPP over a serial device, as opposed to a direct
 ethernet connection, then you may first need to dial your Internet
 Service Provider using the ppp utility we provide for that purpose.
 If you're using SLIP over a serial device then the expectation is
 that you have a HARDWIRED connection.

 You         User Confirmation Requested                      ink"
 cabl  Do you want to try DHCP configuration of the interface?

               [ Yes ]              No

    sl0    SLIP interface on device /dev/cuad0 (COM1)
    ppp0   PPP interface on device /dev/cuad0 (COM1)

              [  OK  ]          Cancel
     [ Press F1 to read network configuration manual ]
```

Figure 4. Install dhcp

You are now the root user and you will be able to retrieve the source code and eventually compile it. Now let us copy the template CVSup file, which will be used by csup to retrieve our source code. We will place the copy in our `root` directory.

```
# cp /usr/share/examples/cvsup/stable-
supfile /root/freebsd-71-csup
```

After that we need to edit the file so that it does what we expect it to do:

```
# vi /root/freebsd-71-csup
```

A new window will be drawn in which the contents of the file will be displayed. We need to modify two different variables, which can be easily found by searching for the text. Scroll down by using the arrow keys and place the cursor on the `CHANGE_THIS` part of the text below:

```
*default host=CHANGE_THIS.FreeBSD.org
```

Issue `cw` and type the following: cvsup.countrycode where countrycode should be replaced by the country you are in. I would type cvsup.nl. After that press [esc] and navigate to the line stating the default release, which looks like the following:

```
*default release=cvs tag=RELENG_7
```

Place the cursor on the `RELENG_7` part and again issue `cw` and type `RELENG_7_1` and hit [esc]. Issue `<quote>:wq</quote>` after which the file had been saved. We now set the server from which we will be fetching the sources, and the distribution that we have choosen. 7.1-RELEASE in our case (including security patches).

You will return to a root prompt (`#`) where we can update the source code:

```
# csup /root/freebsd-71-csup
```

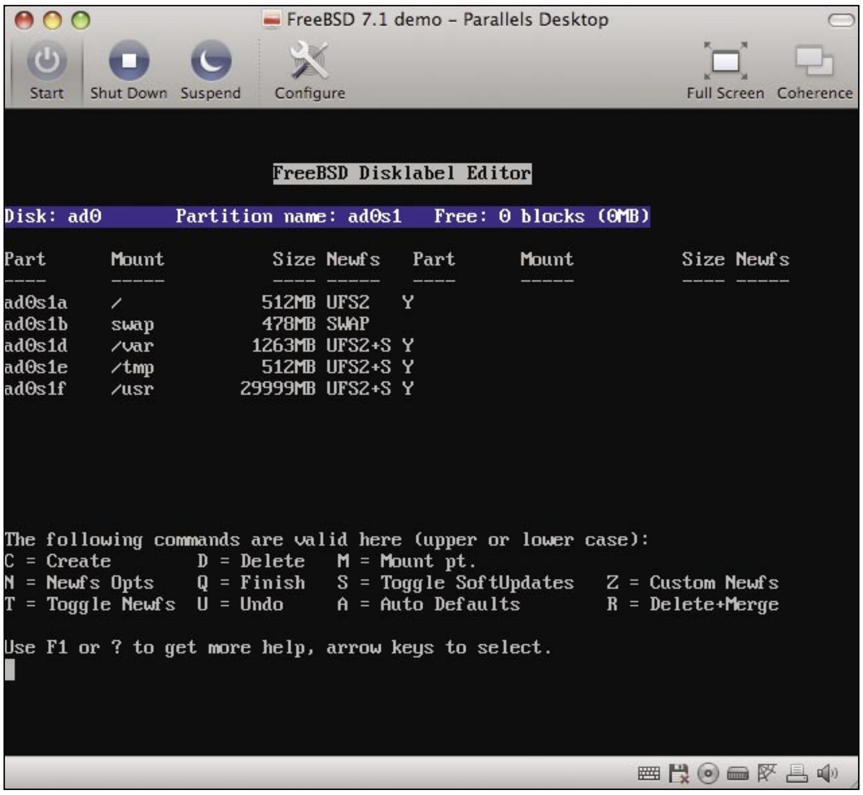
If everything had been configured correctly this will take a little and will show information about files that are being added / checked out and things like that. After the run had been completed you will return to a root (`#`) prompt.

Compiling the latest source code for your system

Now that we have the source code, we can find it under `/usr/src`. Navigate to it by doing:

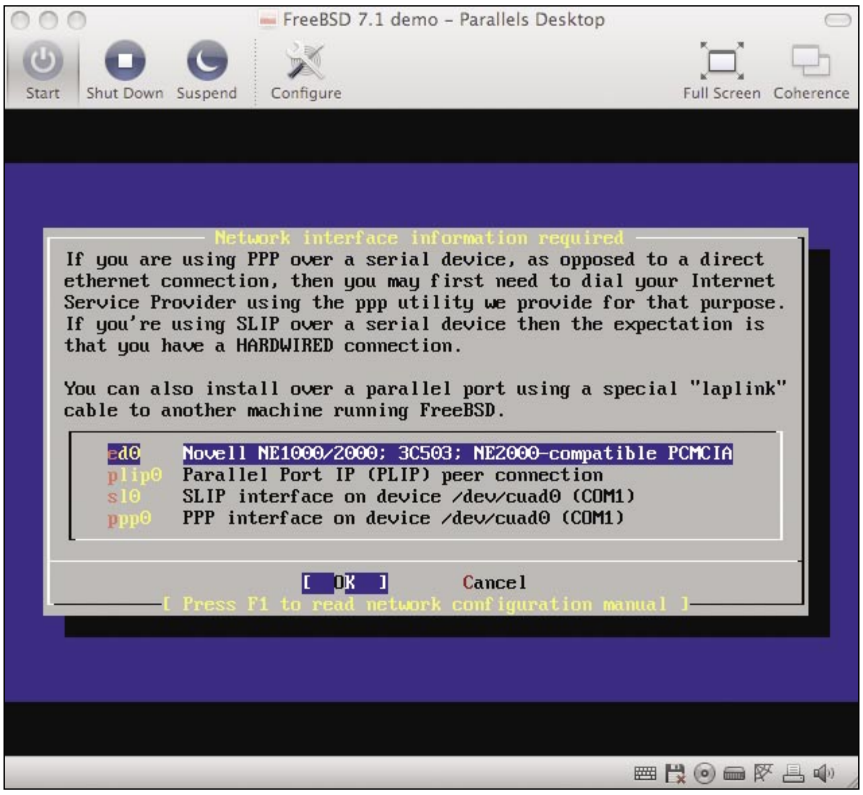
```
# cd /usr/src
```

Now we can start the upgrade procedure. Note that I am expecting you to use the `GENERIC` kernel and that we do not modify things upfront. You can find more information about the procedure here [1], on how to adjust your kernel configuration to your specific need, note well that you should be able to support yourself in case you go for this option because only



```
                          FreeBSD Disklabel Editor

Disk: ad0       Partition name: ad0s1   Free: 0 blocks (0MB)

Part      Mount          Size Newfs   Part      Mount          Size Newfs
----      -----          ---- -----   ----      -----          ---- -----
ad0s1a    /             512MB UFS2    Y
ad0s1b    swap          478MB SWAP
ad0s1d    /var         1263MB UFS2+S Y
ad0s1e    /tmp          512MB UFS2+S Y
ad0s1f    /usr        29999MB UFS2+S Y


The following commands are valid here (upper or lower case):
C = Create        D = Delete    M = Mount pt.
N = Newfs Opts    Q = Finish    S = Toggle SoftUpdates    Z = Custom Newfs
T = Toggle Newfs  U = Undo      A = Auto Defaults         R = Delete+Merge

Use F1 or ? to get more help, arrow keys to select.
```

Figure 5. Install disklayout

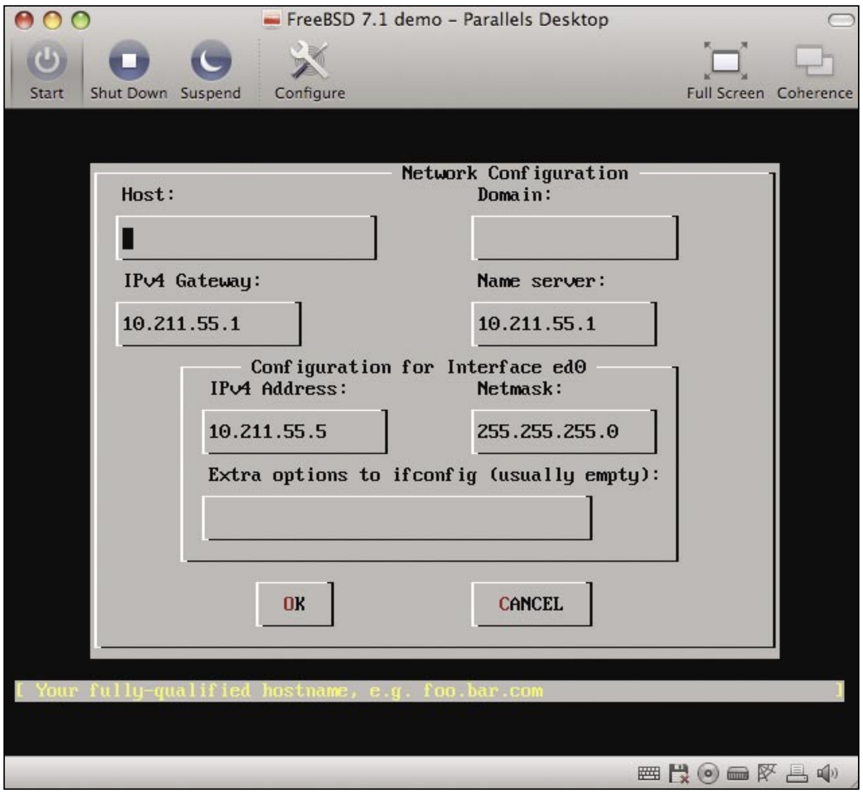


```
 ------ Network interface information required -----------------
| If you are using PPP over a serial device, as opposed to a direct
| ethernet connection, then you may first need to dial your Internet
| Service Provider using the ppp utility we provide for that purpose.
| If you're using SLIP over a serial device then the expectation is
| that you have a HARDWIRED connection.
|
| You can also install over a parallel port using a special "laplink"
| cable to another machine running FreeBSD.
|  ------------------------------------------------------------------
| |  ed0     Novell NE1000/2000; 3C503; NE2000-compatible PCMCIA
| |  plip0   Parallel Port IP (PLIP) peer connection
| |  sl0     SLIP interface on device /dev/cuad0 (COM1)
| |  ppp0    PPP interface on device /dev/cuad0 (COM1)
|  ------------------------------------------------------------------
|
|                  [  OK  ]          Cancel
 ---------[ Press F1 to read network configuration manual ]---------
```

Figure 6. Install ethernet

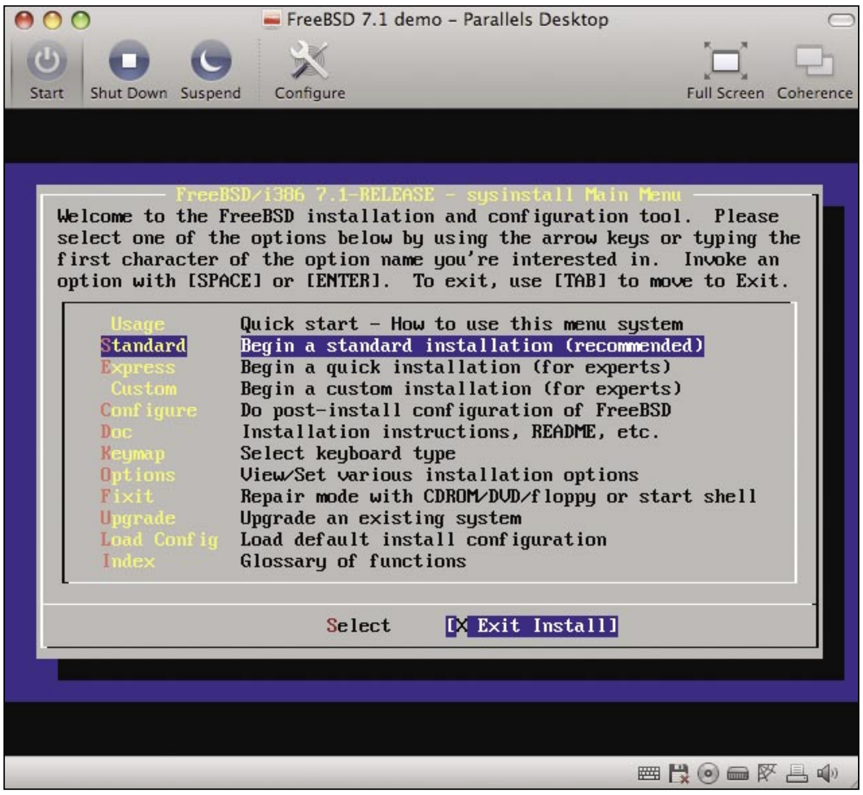
the GENERIC kernel is supported by the FreeBSD team. Of course the various teams will do their best to get you up to speed in case you do have problems, and one of them will be the question whether or not the GENERIC kernel works or not.

Assuming we will do a normal run without any modifications, we will issue the following:



```
Network Configuration
Host:                          Domain:

IPv4 Gateway:                  Name server:
10.211.55.1                    10.211.55.1

Configuration for Interface ed0
IPv4 Address:                  Netmask:
10.211.55.5                    255.255.255.0

Extra options to ifconfig (usually empty):

        OK                 CANCEL

[ Your fully-qualified hostname, e.g. foo.bar.com ]
```

Figure 7. Install ethernet dhcp



```
FreeBSD/i386 7.1-RELEASE - sysinstall Main Menu
Welcome to the FreeBSD installation and configuration tool.  Please
select one of the options below by using the arrow keys or typing the
first character of the option name you're interested in.  Invoke an
option with [SPACE] or [ENTER].  To exit, use [TAB] to move to Exit.

   Usage        Quick start - How to use this menu system
  Standard      Begin a standard installation (recommended)
  Express       Begin a quick installation (for experts)
   Custom       Begin a custom installation (for experts)
  Configure     Do post-install configuration of FreeBSD
  Doc           Installation instructions, README, etc.
  Keymap        Select keyboard type
  Options       View/Set various installation options
  Fixit         Repair mode with CDROM/DVD/floppy or start shell
  Upgrade       Upgrade an existing system
  Load Config   Load default install configuration
  Index         Glossary of functions

             Select          [X Exit Install]
```

Figure 8. Install mainoverview afterinstall

```
# make buildworld && make buildkernel
```

on the commandline which will result in a new world and kernel in it's holding place. If all went well you will have a message stating that the kernel build was succesfull (it will not tell you that the world build was succesfull because it scrolled out of view, but by using this command, the kernel will only be build when the world had been succesfully completed).

```
<screen>>>>> Kernel build for GENERIC
completed on ``date''</screen>
```

We can now issue `make installkernel` and reboot into single user mode. After rebooting logon again and again navigate to `/usr/src`:

```
# cd /usr/src
```

Start the mergemaster tool in order to install the kernel:

```
# mergemaster -p
```

This will install required support files and configuration files that where needed, so that the new world will be able to succesfully install and start. If this completes we can issue the following:

```
# make installworld
```

followed by:

```
# mergemaster -U
```

The latest mergemaster command will automatically upgrade files that had not been user modified.

When this completes we can reboot the machine and everything will start working again.

Using the latest source code as a foundation for the Jails

With the latest sourcecode that we have prepared and installed on the host system it is possible to finally start working on our jails. The idea of jails that we will be presenting is heavily borrowed from Simon Nielsen's guide of installing FreeBSD Service Jails, which is a form of jails that is being used for service specific goals, like webservers and things like that. Before we can do the setup though, it's important to understand a few bits and pieces of the upcoming installation.

The installation for the distribution will be done in one so-called `master-jail`. This master jail is the template for each and every jail and will be accessed read-only. Each jail will get it's own space in which it can write to. Following Simon's style we will do a few definitions;

- Each jail will be mounted under the `/home/j` directory. This will be our jail root.
- `/home/j/mroot` will be the template for all our jails and will be read only for us.
- All jails will get a seperated directory under `/home/j` and will be named with something descriptive, like www for the webserver.
- Each jail will have a /s directory which will be linked to the read-write system, this enables us to seperate read-write access and protect our default binaries and things. You are free to make everything as read only as possible, but be aware that some directory's like `/var` and `/tmp` need write access as well to write away important state information and logging.
- Each jail will have a read-write system that is based upon `/home/j/skel`
- Each jailspace (the read write portion of each jail) will be created in `/home/js`

For the sake of the installation we will keep track of these definitions, but of course you are free to modify that to your needs. In case you do want to use a seperated partition or anything, I would suggest either `/jails` or `/usr/local/jails` as the name for the directory structure.

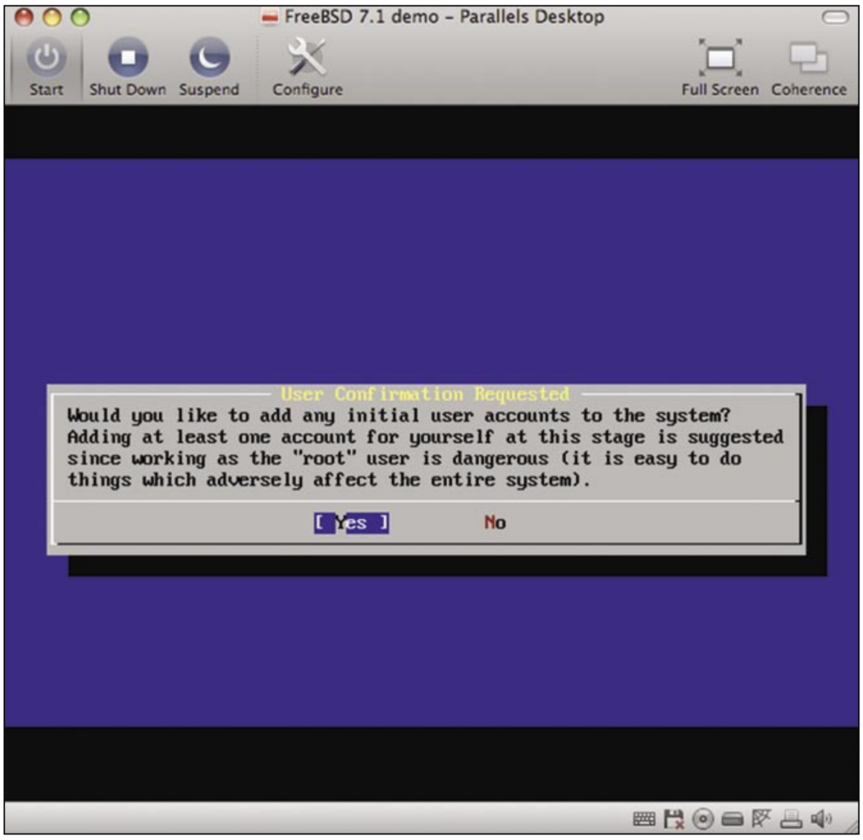
The directories to do this are not setup automatically, so we need to create them upfront:

```
# mkdir /home/js /home/j
```

Because we already have up to date sources, which are already compiled, we can easily start the installation. Before we can do that we need to create the appropriate directores:

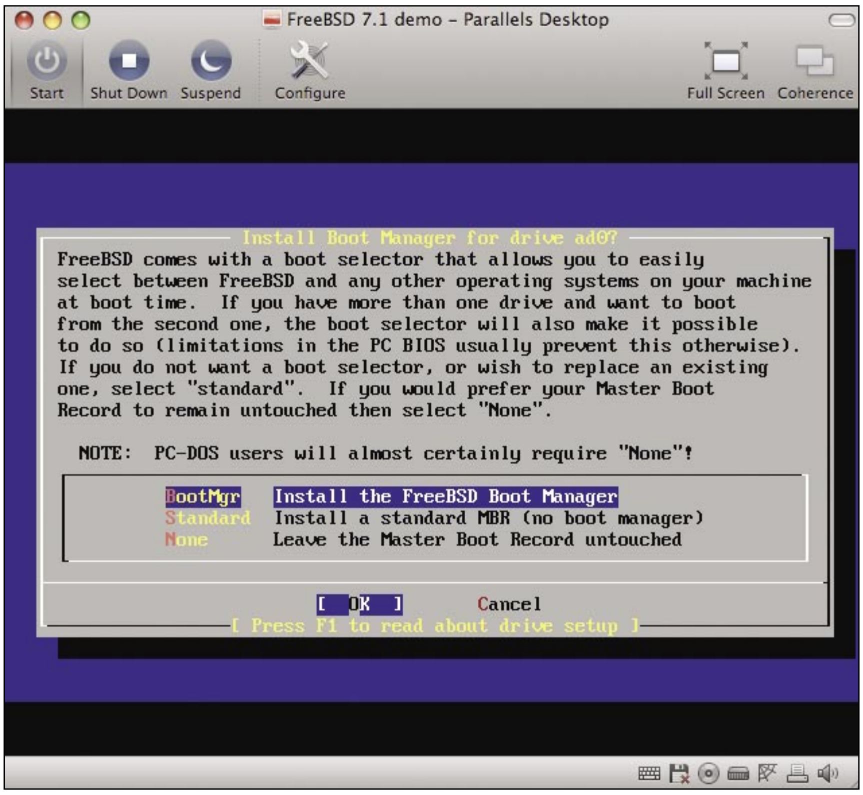
```
# mkdir -p /home/j/mroot
# cd /usr/src
# make installworld DESTDIR=/home/j/mroot
```

After that we need to navigate into the `/home/j/mroot` directory and create the skeleton for the ports:



```
User Confirmation Requested
Would you like to add any initial user accounts to the system?
Adding at least one account for yourself at this stage is suggested
since working as the "root" user is dangerous (it is easy to do
things which adversely affect the entire system).

                 [ Yes ]          No
```

Figure 9. Install user creation question



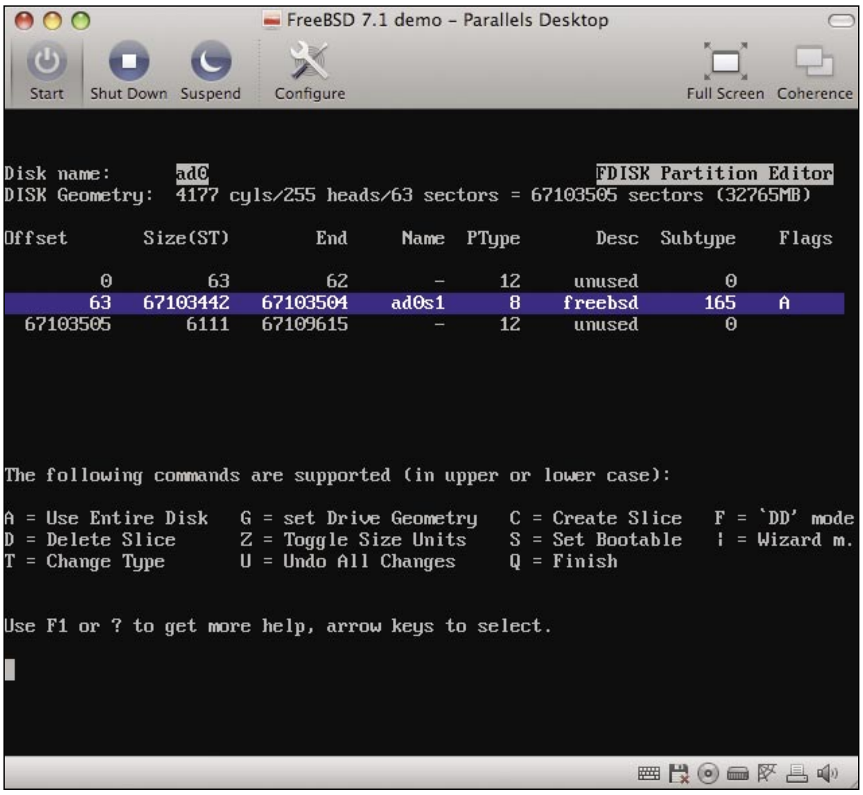
```
Install Boot Manager for drive ad0?
FreeBSD comes with a boot selector that allows you to easily
select between FreeBSD and any other operating systems on your machine
at boot time.  If you have more than one drive and want to boot
from the second one, the boot selector will also make it possible
to do so (limitations in the PC BIOS usually prevent this otherwise).
If you do not want a boot selector, or wish to replace an existing
one, select "standard".  If you would prefer your Master Boot
Record to remain untouched then select "None".

  NOTE:  PC-DOS users will almost certainly require "None"!

        BootMgr   Install the FreeBSD Boot Manager
        Standard  Install a standard MBR (no boot manager)
        None      Leave the Master Boot Record untouched

                 [  OK  ]          Cancel
          [ Press F1 to read about drive setup ]
```

Figure 10. Installation bootmgr

```
# cd /home/j/mroot
# mkdir usr/ports
# portsnap -p /home/j/mroot/usr/ports
fetch extract
```

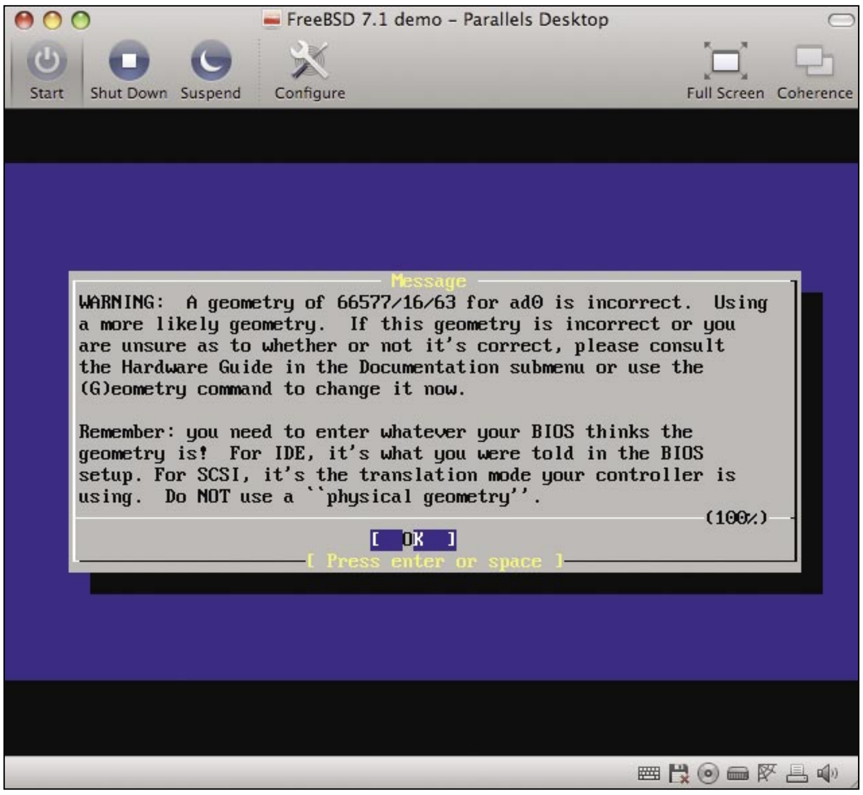
Now create the skeleton for the read-write portion of the system

```
# mkdir home/j/skel /home/j/skel/home
/home/j/skel/usr-X11R6 /home/j/skel/
distfiles
# mv etc /home/j/skel
# mv usr/local /home/j/skel/usr-local
# mv tmp /home/j/skel
# mv var /home/j/skel
# mv root /home/j/skel
```



```
Disk name:      ad0                                FDISK Partition Editor
DISK Geometry:  4177 cyls/255 heads/63 sectors = 67103505 sectors (32765MB)

Offset       Size(ST)        End     Name  PType       Desc  Subtype    Flags

         0         63         62        -     12     unused        0
        63   67103442   67103504    ad0s1      8    freebsd      165     A
  67103505       6111   67109615        -     12     unused        0




The following commands are supported (in upper or lower case):

A = Use Entire Disk    G = set Drive Geometry   C = Create Slice     F = `DD' mode
D = Delete Slice       Z = Toggle Size Units    S = Set Bootable     | = Wizard m.
T = Change Type        U = Undo All Changes     Q = Finish

Use F1 or ? to get more help, arrow keys to select.
```

Figure 11. Installation fdisk



```
-------------------------- Message ---------------------------
WARNING:  A geometry of 66577/16/63 for ad0 is incorrect.  Using
a more likely geometry.  If this geometry is incorrect or you
are unsure as to whether or not it's correct, please consult
the Hardware Guide in the Documentation submenu or use the
(G)eometry command to change it now.

Remember: you need to enter whatever your BIOS thinks the
geometry is!  For IDE, it's what you were told in the BIOS
setup. For SCSI, it's the translation mode your controller is
using.  Do NOT use a ``physical geometry''.
                                                          (100%)
                            [  OK  ]
------------------- [ Press enter or space ] -----------------
```

Figure 12. Installation fdiskwarning

The mergemaster tool can assist us with the installation (and later on updating) the configuration files. Since this will create additional directories that are not needed, we need to remove them afterwards.

```
# mergemaster -t /home/j/skel/var/
tmp/temproot -D /home/j/skel -i
# cd /home/j/skel
# rm -R bin boot lib libexec mnt proc
rescue sbin sys usr dev
```

We are almost done, we need to setup the read write file system sot hat we have a place to store files etc.

```
# cd /home/j/mroot
# mkdir s
# ln -s s/etc etc
# ln -s s/home home
# ln -s s/root root
# ln -s ../s/usr-local usr/local
# ln -s ../s/usr-X11R6 usr/X11R6
# ln -s ../../s/distfiles usr/ports/
distfiles
# ln -s s/tmp tmp
# ln -s s/var var
```

Because we have specifically created a read-write space in our jail, we need to make sure that we can build ports in the right directory, add the following to `/home/j/skel/etc/make.conf` by doing the following:

```
echo "WRKDIRPREFIX?= /s/portbuild" >>
/home/j/skel/etc/make.conf
```

Our basic installation and setup had now been done. Lets continue by setting up specific jails, the example I am going to give only handles setting up a webserver, see the FreeBSD Handbook for additional examples and more information.

Assuming everything went fine so far, we will setup the basic things needed to build a webserver. First, we will handle the foundation for it and later we will use third party packages that will enable to use of the webserver. We will name the jail `www`, which is the appropriate name for a webserver.

Let us create the directories that are required for this;

```
# mkdir /home/j/www /home/js/www
```

Since we want to use the jails after a restart, we will specify them in the `/etc/fstab` file so that the machine will configure the required directories during startup:

```
echo "/home/j/mroot /home/j/www nullfs
ro 0 0" >> /etc/fstab
echo "/home/js/www /home/j/www/s"
nullfs rw 0 0" >> /etc/fstab
```

The above will only make the directory structure available, we of course need to start the management foundation for the jails as well. We can do that by adding the following lines to `/etc/rc.conf`:

```
# echo '
jail_enable="YES"
jail_set_hostname_allow="NO"
jail_list="www"
jail_www_hostname="www.example.org"
jail_www_ip="192.168.0.1"
jail_www_rootdir="/usr/home/j/www"
jail_www_devfs_enable="YES"' >> /etc/
rc.conf
```

We need to copy over the skeleton directory to the jail, before we can do this we need to install the `sysutils/cpdup` utility:

```
# pkg_add -r cpdup
```

This will remotely add the cpdup utility as hinted by Simon's guide.

```
# cpdup /home/j/skel /home/js/www
```

The jails are now ready to be started, we will need to attach the various required directories (which will happen automatically at boot):

```
# mount -a
```

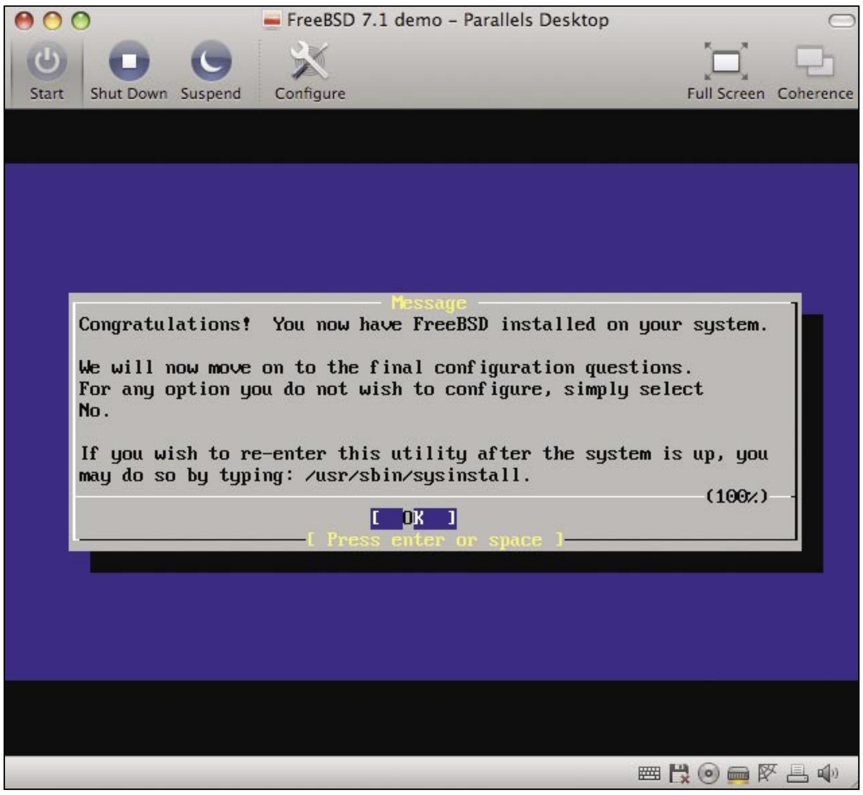
And we need to start the jails

```
# /etc/rc.d/jail start
```

You should be able to view information about the jails that are being started now. In case you didn't see this information, or want to review information about what is currently running, you can use the `jls` command, which will list the active jails. In order to do something fancy with the jail, you need to get access to it. Since we are logged in as root, we can easily hop into the jail:

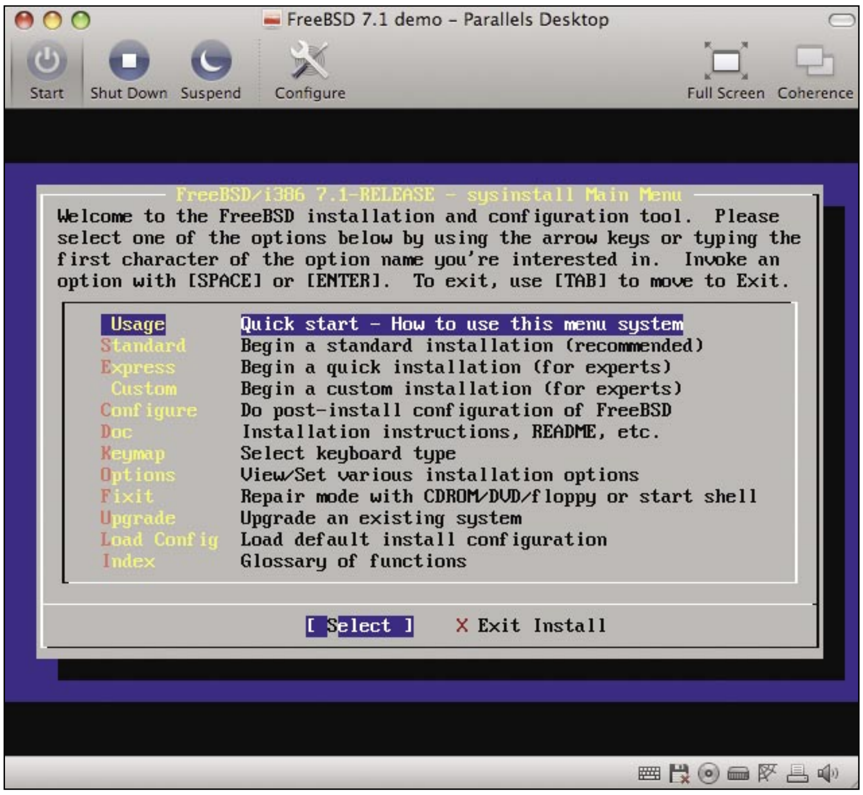
```
# jexec 1 /bin/csh
```

This will give you the CSH shell within the first jail (You can get the JailID from the `jls` command).



```
Message
Congratulations!  You now have FreeBSD installed on your system.

We will now move on to the final configuration questions.
For any option you do not wish to configure, simply select
No.

If you wish to re-enter this utility after the system is up, you
may do so by typing: /usr/sbin/sysinstall.
                                                          (100%)
                            [  OK  ]
                    [ Press enter or space ]
```

Figure 13. installation installed congratz



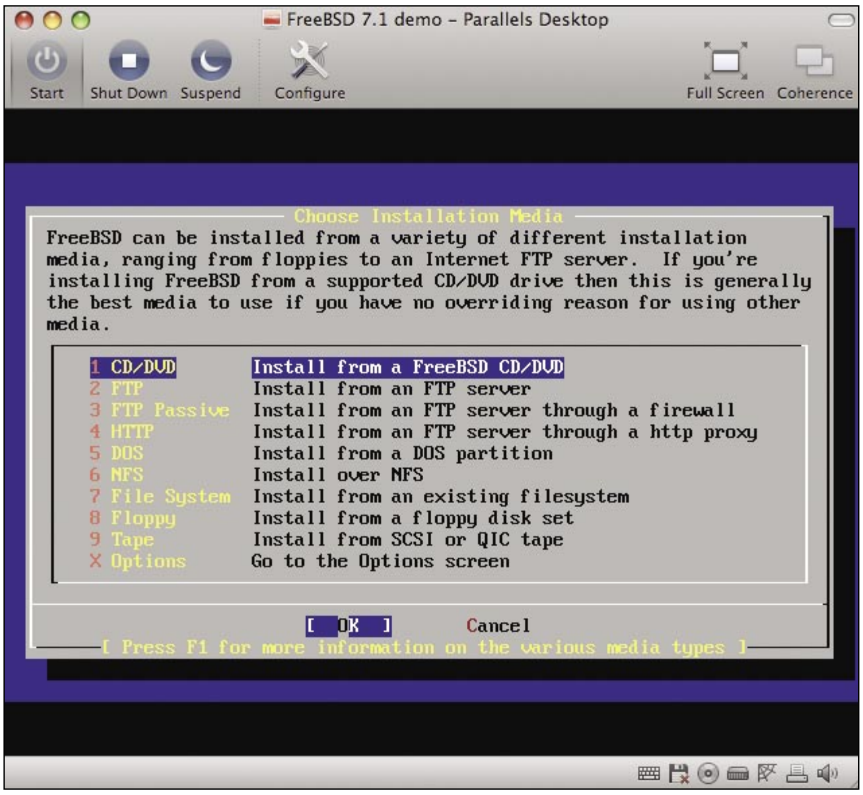
```
FreeBSD/i386 7.1-RELEASE - sysinstall Main Menu
Welcome to the FreeBSD installation and configuration tool.  Please
select one of the options below by using the arrow keys or typing the
first character of the option name you're interested in.  Invoke an
option with [SPACE] or [ENTER].  To exit, use [TAB] to move to Exit.

  Usage          Quick start - How to use this menu system
  Standard       Begin a standard installation (recommended)
  Express        Begin a quick installation (for experts)
   Custom        Begin a custom installation (for experts)
  Configure      Do post-install configuration of FreeBSD
  Doc            Installation instructions, README, etc.
  Keymap         Select keyboard type
  Options        View/Set various installation options
  Fixit          Repair mode with CDROM/DVD/floppy or start shell
  Upgrade        Upgrade an existing system
  Load Config    Load default install configuration
  Index          Glossary of functions

              [ Select ]     X Exit Install
```

Figure 14. Installation mainscreenbeforeinstall

Installing third party packages within your jails

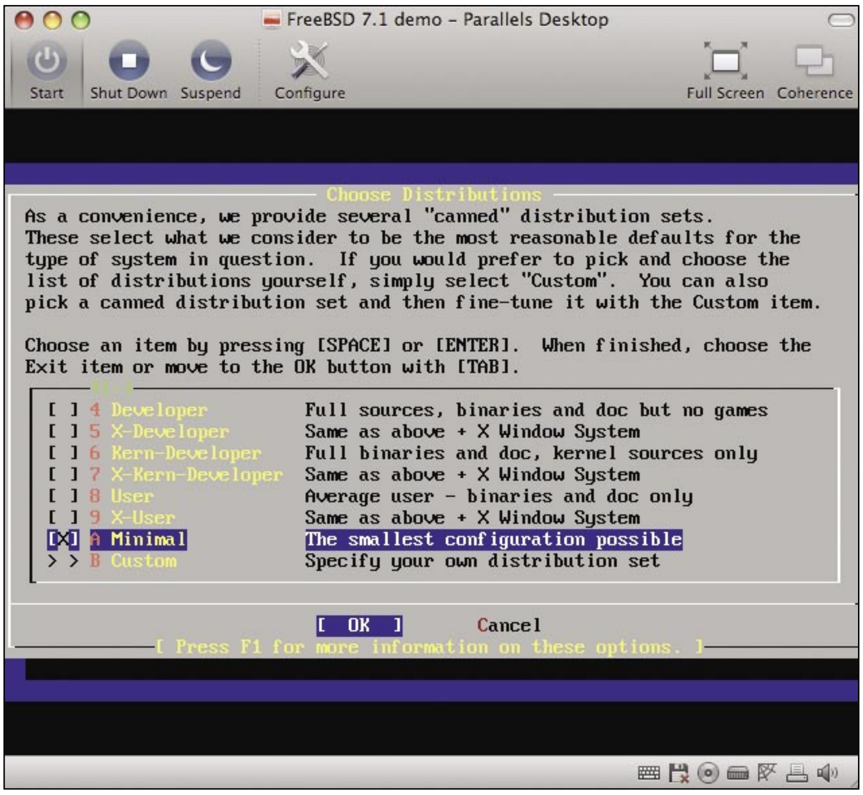
We now have a complete jail running, but no services yet. Before we determined that we will be running a webserver on this. I will give an idea on how to setup a webserver, which you can adjust to your own specific needs.

During the Jail installation we retrieved a fresh copy of the FreeBSD Ports Collection, by using the portsnap utility. This enables us to easily install a webserver. There are various webservers, but the one commonly used by people is the Apache webserver. It's also the best known webserver (in my understanding) and has lots of documentation available online at *http://httpd.apache.org.*



```
Choose Installation Media
FreeBSD can be installed from a variety of different installation
media, ranging from floppies to an Internet FTP server.  If you're
installing FreeBSD from a supported CD/DVD drive then this is generally
the best media to use if you have no overriding reason for using other
media.

  1 CD/DVD          Install from a FreeBSD CD/DVD
  2 FTP             Install from an FTP server
  3 FTP Passive     Install from an FTP server through a firewall
  4 HTTP            Install from an FTP server through a http proxy
  5 DOS             Install from a DOS partition
  6 NFS             Install over NFS
  7 File System     Install from an existing filesystem
  8 Floppy          Install from a floppy disk set
  9 Tape            Install from SCSI or QIC tape
  X Options         Go to the Options screen

                    [  OK  ]          Cancel
      [ Press F1 for more information on the various media types ]
```

Figure 15. Installation mediaselection



```
Choose Distributions
As a convenience, we provide several "canned" distribution sets.
These select what we consider to be the most reasonable defaults for the
type of system in question.  If you would prefer to pick and choose the
list of distributions yourself, simply select "Custom".  You can also
pick a canned distribution set and then fine-tune it with the Custom item.

Choose an item by pressing [SPACE] or [ENTER].  When finished, choose the
Exit item or move to the OK button with [TAB].

 [ ] 4 Developer          Full sources, binaries and doc but no games
 [ ] 5 X-Developer        Same as above + X Window System
 [ ] 6 Kern-Developer     Full binaries and doc, kernel sources only
 [ ] 7 X-Kern-Developer   Same as above + X Window System
 [ ] 8 User               Average user - binaries and doc only
 [ ] 9 X-User             Same as above + X Window System
 [X] A Minimal            The smallest configuration possible
 > > B Custom             Specify your own distribution set

                    [  OK  ]          Cancel
         [ Press F1 for more information on these options. ]
```

Figure 16. Installation packagetoinstall

The FreeBSD Ports Collection has various copies of the Apache webserver, with various tastes. We will install version 2.2 of the webserver and use default installation options to get it going. Remember, we are still in the jail.

Navigate to the Apache Ports directory:

```
# cd /usr/ports/www/apache22
```

To compile and install it use the following:

```
# make install
```

If this is your first time, a popup might show a new window in which you can select certain options that you want to have enabled on your apache installation. We do not care about this, so we navigate to [*OK*] and continue the build. After a while the installation completes, and the webserver had been installed. To start the webserver first add the following line to */etc/rc.conf*:

```
# echo 'apache22_enable="YES"' >>
/etc/rc.conf
```

Followed by the start command:

```
# /usr/local/etc/rc.d/apache22 start
```

The webserver will now be started. Start a browser and navigate to the IP you used to setup the jail, if all went well you will see a page that mentions : "It works!". Additional configuration is left to the reader as an excercise.

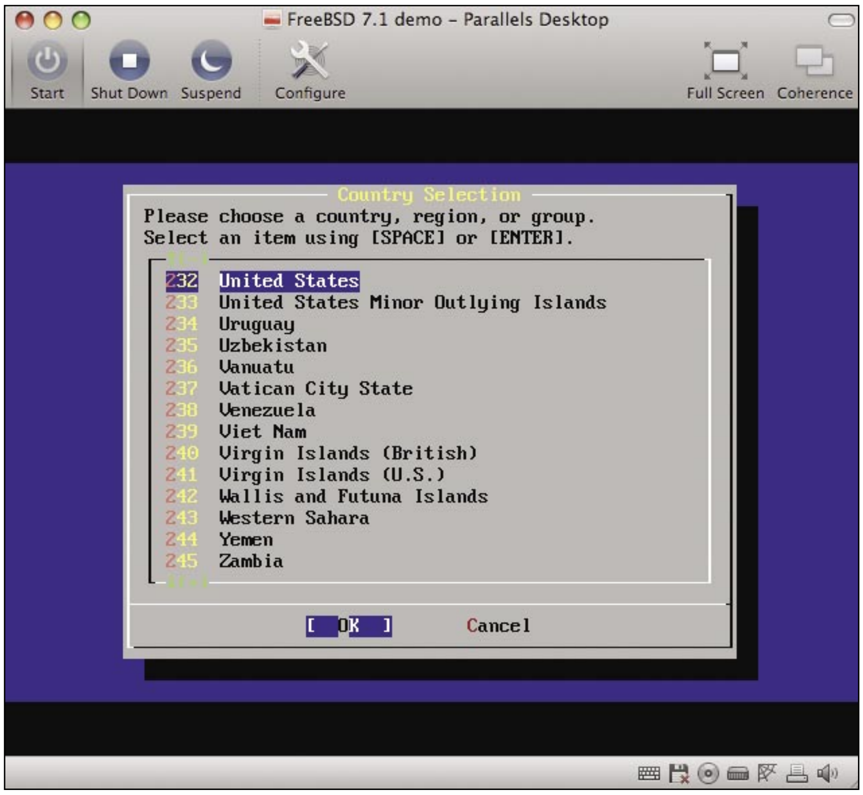
Personally I have setup an hosting webserver, mailserver, internal mailserver, spamfilter + rbl server, playgroundserver by using the same approach. Each and every jail is seperated by eachother, and cannot break out of their scope. Though the current design is limited by having available only a single IP for each jail, it will change in the future and make the FreeBSD Jails an even more robust form of enhancing your FreeBSD box' security.

What will be there in FreeBSD 7.2

First of all, FreeBSD jails will be able to use multiple IP's, both IPv4 and IPv6, as well as IP-less jails. This enhancement makes it easier for webservers for example to do virtualhosting or do SSL virtualhosting.

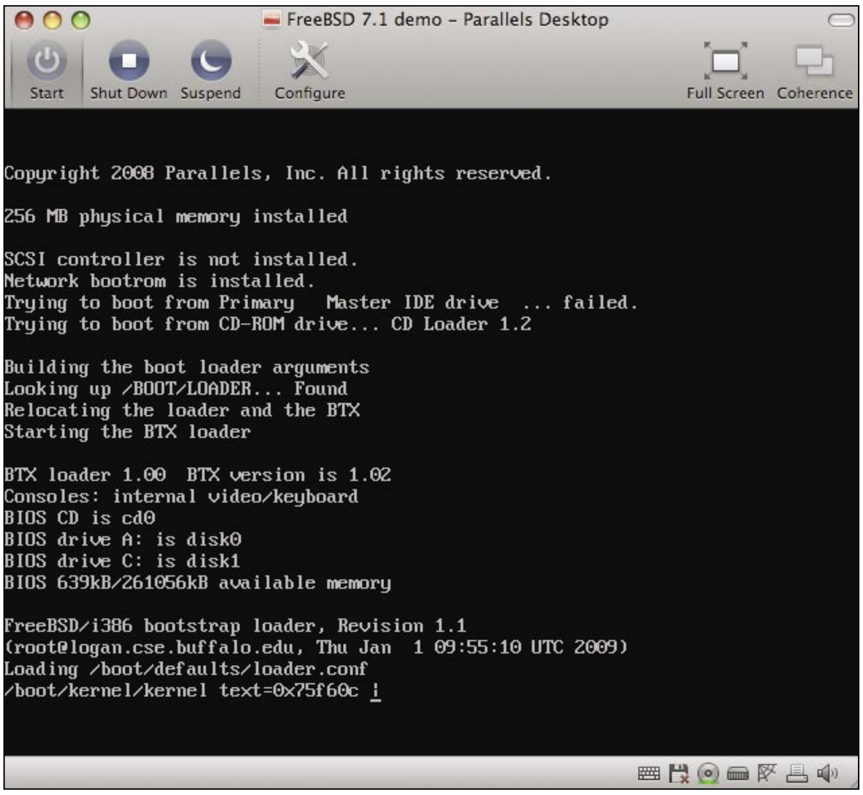
Concluding

After following this article you should be able to install the FreeBSD 7.1 system and setting up FreeBSD jails to do service specific tasks. Furthermore, you know what the upcoming version has to offer in comparison with FreeBSD jails.



```
Country Selection
Please choose a country, region, or group.
Select an item using [SPACE] or [ENTER].

232 United States
233 United States Minor Outlying Islands
234 Uruguay
235 Uzbekistan
236 Vanuatu
237 Vatican City State
238 Venezuela
239 Viet Nam
240 Virgin Islands (British)
241 Virgin Islands (U.S.)
242 Wallis and Futuna Islands
243 Western Sahara
244 Yemen
245 Zambia

[  OK  ]        Cancel
```

Figure 17. Installation selectcountry



```
Copyright 2008 Parallels, Inc. All rights reserved.

256 MB physical memory installed

SCSI controller is not installed.
Network bootrom is installed.
Trying to boot from Primary   Master IDE drive  ... failed.
Trying to boot from CD-ROM drive... CD Loader 1.2

Building the boot loader arguments
Looking up /BOOT/LOADER... Found
Relocating the loader and the BTX
Starting the BTX loader

BTX loader 1.00  BTX version is 1.02
Consoles: internal video/keyboard
BIOS CD is cd0
BIOS drive A: is disk0
BIOS drive C: is disk1
BIOS 639kB/261056kB available memory

FreeBSD/i386 bootstrap loader, Revision 1.1
(root@logan.cse.buffalo.edu, Thu Jan  1 09:55:10 UTC 2009)
Loading /boot/defaults/loader.conf
/boot/kernel/kernel text=0x75f60c |
```

Figure 18. Parallels booting install

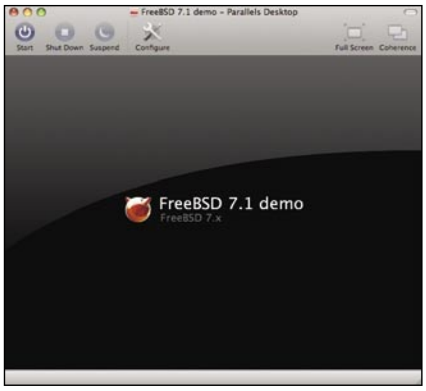


Figure 19. Parallels main screen.

References

I talked a lot about the FreeBSD Handbook, and actually loads of the content of this article found it's history in the handbook. The handbook is one of the best available documents for an opensource Operating System and covers basic things like the installation, Unix Basics and advanced things like Advanced Networking. I think it's advisable for everyone to have a peek at the handbook to get more information about what we did and how to get further with your machine: *http://www.freebsd.org/doc/en/books/handbook/* Or if you rather read a localized version, some people (including me and a lot of other people from the Netherlands) create these versions for you. If it exists you can find it on:

http://www.freebsd.org/doc/‹langcode›/books/handbook/

So for Dutch that would mean:

http://www.freebsd.org/doc/nl/books/handbook/

About the Author

Remko Lodder is a 25 year old FreeBSD enthusiast, in his spare time he likes being with his son and girlfriend, playing with FreeBSD systems, and wearing various FreeBSD hats to help the community. In his professional life Remko is an Unix Engineer for Snow B.V. in the Netherlands mostly focussing on Firewalls and Security (Checkpoint/Juniper etc.). You can contact him by sending an email to: remko@FreeBSD.org

PC-BSD 7.1

The FreeBSD Release Engineering Team is pleased to announce the availability of FreeBSD 7.1-RELEASE. This is the second release from the 7-STABLE branch which improves on the functionality of FreeBSD 7.0 and introduces some new features. Some of the highlights:

- The ULE scheduler is now the default in GENERIC kernels for amd64 and i386 architectures. The ULE scheduler significantly improves performance on multicore systems for many workloads.
- Support for using DTrace inside the kernel has been imported from OpenSolaris. DTrace is a comprehensive dynamic tracing framework.
- A new and much-improved NFS Lock Manager (NLM) client.
- Boot loader changes allow, among other things, booting from USB devices and booting from GPT-labeled devices.
- The cpuset(2) system call and cpuset(1) command have been added, providing an API for thread to CPU binding and CPU resource grouping and assignment.
- KDE updated to 3.5.10, GNOME updated to 2.22.3.
- DVD-sized media for the amd64 and i386 architectures

For a complete list of new features and known problems, please see the online release notes and errata list, available at:

- *http://www.FreeBSD.org/releases/7.1R/relnotes.html*
- *http://www.FreeBSD.org/releases/7.1R/errata.html*

For more information about FreeBSD release engineering activities, please see: http://www.FreeBSD.org/releng/

Availability

FreeBSD 7.1-RELEASE is now available for the amd64, i386, ia64, pc98, powerpc, and sparc64 architectures.

For instructions on installing FreeBSD, please see Chapter 2 of The FreeBSD Handbook. It provides a complete installation walk-through for users new to FreeBSD, and can be found online at: *http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/install.html*

Updating Existing Systems

NOTE: If updating from a 7.0 or earlier system due to a change in the Vendor's drivers certain Intel NICs will now come up as `igb(4)` instead of `em(4)`. We normally try to avoid changes like that in stable branches but the vendor felt it necessary in order to support the new adapters. See the UPDATING entry dated 20080811 for details. There are only 3 PCI ID's that should have their name changed from `em(4)` to `igb(4)`: `0x10A78086`, `0x10A98086`, and `0x10D68086`. You should be able to determine if your card will change names by running the command `pciconf -l`, and for the line representing your NIC (should be named em on older systems, e.g. `em0` or `em1`, etc) check the fourth column. If that says `chip=0x10a78086` (or one of the other two IDs given above) you will have the adapter's name change.

Updates from Source

The procedure for doing a source code based update is described in the FreeBSD Handbook:

- *http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/synching.html*
- *http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/makeworld.html*

The branch tag to use for updating the source is RELENG_7_1.

FreeBSD Update

The freebsd-update(8) utility supports binary upgrades of i386 and amd64 systems running earlier FreeBSD releases. Systems running 7.0-RELEASE, 7.1-BETA, 7.1-BETA2, 7.1-RC1, or 7.1-RC2 can upgrade as follows:

```
# freebsd-update upgrade -r 7.1-
RELEASE
```

During this process, FreeBSD Update may ask the user to help by merging some configuration files or by confirming that the automatically performed merging was done correctly.

```
# freebsd-update install
```

The system must be rebooted with the newly installed kernel before continuing.

```
# shutdown -r now
```

After rebooting, freebsd-update needs to be run again to install the new userland components, and the system needs to be rebooted again:

```
# freebsd-update install
# shutdown -r now
```

Users of Intel network interfaces which are changing their name from `em` to `igb` should make necessary changes to configuration files BEFORE running freebsd-update, since otherwise the network interface will not be configured appropriately after rebooting for the first time.

Users of earlier FreeBSD releases (FreeBSD 6.x) can also use freebsd-update to upgrade to FreeBSD 7.1, but will be prompted to rebuild all third-party applications (e.g., anything installed from the ports tree) after the second invocation of "freebsd-update install", in order to handle differences in the system libraries between FreeBSD 6.x and FreeBSD 7.x.

For more information, see: http://*www.daemonology.net/blog/2007-11-11-freebsd-major-version-upgrade.html*

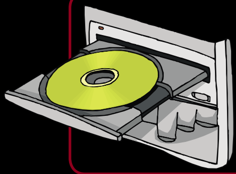
Support

The FreeBSD Security Team currently plans to support FreeBSD 7.1 until January 31st 2011. For more information on the Security Team and their support of the various FreeBSD branches see:

http://www.freebsd.org/security/

Trademark

FreeBSD is a registered trademark of The FreeBSD Foundation.



If the DVD content cannot be accessed and the disc is not damaged, try to run it on at least two DVD-ROMs.

MAGAZINE BSD 3/2009

If you have encountered any problems with the DVD, please write to: cd@software.com.pl

OpenSMTPD

Gilles Chehade

In this issue I will take the opportunity to write about the SMTP server that was imported into the OpenBSD source tree. It isn't enabled yet-it isn't even linked to the build, but it is in progress and this article will describe what it currently does.

When SMTPd was imported, several people asked why we needed a new project and why we did not import their favorite mta (mail transfer agent) application. There's actually more than just one reason. Currently, OpenBSD ships with the well-known and rather unpopular Sendmail, which has a very bad reputation because of its past history of security issues, but I will get to that soon. Long story short, many of us want to replace it with another mta application, so let's see what the alternatives are.

If I look at my mail headers for the last few months, Sendmail, Postfix, Exim, and Qmail account for most of the traffic. Exchange is not going to be useful, so I guess we can skip it.

This leaves us with Postfix, Qmail, and Exim.

Exim is licensed under the GPL, so we'll not be considering it here: OpenBSD no longer imports GPL licensed code into the base, thus it isn't a possible alternative. I don't know what Exim is worth; I can't honestly say I ever even ran it.

Postfix is licensed under the IPL (IBM Public License), so it's also a non-option: as this cannot go in either, the IPL contains a clause which goes against the very goals of the OpenBSD project.

At the time of this writing, Qmail is supposedly released to the Public Domain, so it is a viable alternative from the license point of view. Unfortunately, the developers do not agree that it is a better choice than Sendmail and the fact is that this point is clearly highly debatable. For every Qmail fan I find, I can find someone who strongly opposes it. Not even to mention that the author has been hard to deal with in the past and even if Qmail is public domain, it is still likely that we would have to work with the author at some point.

Sendmail turns out to be the best choice out of these. It has a license which doesn't go against our goals, it is mature and works great, and it isn't going to get your server hacked. Sendmail is used by the largest corporations, with the most complex setups, and it is actively followed and fixed as issues arise.

By now you should be asking yourself *well if Sendmail is so nice, why change to something else ?*.

I recently had to make changes to a setup that had been running for months. I assumed I would deal with it in a few seconds because I knew what I wanted to do and it was a trivial task. I ended up spending another half an hour jumping from a book to a search engine, and spending half an hour testing the new setup just to make sure I did not break anything with my two-line change.

As an OpenBSD user, I am used to getting things working by reading manuals and comments in sample configuration files provided with the system. Sendmail doesn't work that way: if you try a setup with no book and no internet access then you are very likely to fail if you are not very experienced with it.

This is why some developers, including myself, think we need to provide a new SMTP server that is developed with OpenBSD's goals of security and simplicity in mind.

Design and processes description

OpenSMTPD follows the same design as various recent daemons in OpenBSD. It is a multi-process application which uses the imsg framework to let processes do IPC, while making use of several techniques to mitigate risks. The daemon has a fully asynchronous design and, in theory, does not block on IO. In practice we lack an asynchronous DNS resolver (for now) and as a result we have all of our resolutions serialized and blocking.

Except for one process, used for privilege separation, all processes run with no privileges at all and are chrooted to either /var/empty or the mail queue. The processes that need to open files outside of their chroot jails will rely on imsg to do fd passing from a process which can access these files.

OpenSMTPD has several processes which looks a bit scary at first look (see Listing 1).

They all have very specific tasks and while we attempted to reduce the number of processes, it always turned out to be a bad idea from either a security or performances point of view. It doesn't seem over-engineered either; other mta applications have about the same dispatch of tasks.

Let's review what they do:

The two most exposed processes are (1) the SMTP server that handles SMTP sessions from untrusted clients over the network and (2) the control process which handles the enqueuing from system users. They do essentially the same job, turning a set of recipients into a structure that processes can play with, however one does it by parsing a command line, while the other does it by parsing a session it has received over the network. As the ps output above shows, both processes run as `user _SMTPd`, but they are also chrooted to `/var/empty`.

Each time an envelope is created, it is sent to the mail filter agent that is in charge of checking the rule set and deciding if a recipient is rejected or not. It acts as a *firewall* to the other processes, rejecting envelopes that we do not want to process at an early stage. Later, this is where we will get our mail filters plugged. The process runs as `user _SMTPd` and chrooted to the `/var/empty` directory. Envelopes which aren't rejected are handed over to the lookup agent. They are expanded and resolved into a recipient usable by the queue process. Expansion is done iteratively so that aliases to aliases to accounts that have forwards that contain aliases work correctly, but with a hard limit to detect loops in case some users play with self referencing forwards. The lookup agent is also in charge of doing all kind of lookups other processes need, such as looking up a group of MX records or resolving a hostname. The process also runs unprivileged as `user _SMTPd`, but unlike other processes it can't run chrooted, as we want it to access various resources such as the aliases database, resolv.conf, and the passwd database `/etc/pwd.db`, amongst other things.

The queue process is in charge of recording envelopes to a disk-based queue. It was initially also in charge of scheduling deliveries and updating envelopes, but this proved to be the wrong idea as it made the code considerably trickier. The fact is, this process is queried by most of the other processes and we do not want it to perform any time-consuming operations, as it could ultimately stop handling imsg from other processes and cause sessions to timeout. Thus it runs unprivileged, and chrooted to the mail queue root.

The runner process was introduced as a solution to prevent queue process from ever being too busy to handle incoming imsg. The runner process walks through the queue, detects if envelopes are expired or if they can be scheduled. When it finds envelopes that are for an identical sessions and which should be sent to the same remote MX, it merges them into a batch. This allows SMTPd to do a delivery to multiple recipients in a single remote SMTP session. Unprivileged and chrooted to the mail queue root.

The mail transfert agent is an SMTP client which establishes an SMTP session with a remote MX and hands it over one or more envelopes. Process then keeps track of delivery status for each envelope and notifies queue so that a decision is made to try remove envelope from queue, generate a mailer daemon, or try the same delivery later. It runs unprivileged and chrooted to `/var/empty`.

The mail delivery agent is a very simple process which takes care of delivery by simply writing to a file descriptor. The file descriptor points to an mbox, a Maildir, or to a pipe we have to another external mail delivery application, such as procmail for example. Process runs with no privileges and is chrooted to `/var/empty`.

Finally, the parent process is in charge of starting SMTPd and doing all kind of privileged tasks on behalf of other processes. It opens an mbox, Maildir, or even a pipe to a process it just created to start an external delivery agent. It is currently used for authentication too as we need privileges to read the secure passwd database.

Programs

OpenSMTPD ships with SMTPd, the SMTP server daemon, but also with a small set of tools to help the administrator in his daily tasks.There are currently two tools:

```
* makemap
* SMTPctl
```

The makemap utility is used to generate mappings of `key/values` which are used for various purposes inside SMTPd. The

Listing 1. Processes list

```
mx1.poolp.org:gilles {109} ps auxwww | grep smtp
root      23533  0.0  0.2  1020  1948 p2  I+     7:22PM    0:00.02 smtpd:
parent (smtpd)
_smtpd     7376  0.0  0.1   984  1508 p2  S+     7:22PM    0:00.14 smtpd:
mail delivery agent (smtpd)
_smtpd    12218  0.0  0.2  1196  1588 p2  I+     7:22PM    0:00.03 smtpd:
lookup agent (smtpd)
_smtpd    11378  0.0  0.1   984  1424 p2  S+     7:22PM    0:00.13 smtpd:
mail filter agent (smtpd)
_smtpd    10534  0.0  0.2  1056  1596 p2  I+     7:22PM    0:00.05 smtpd:
queue handler (smtpd)
_smtpd     2339  0.0  0.2   964  2032 p2  S+     7:22PM    0:00.13 smtpd:
mail transfer agent (smtpd)
_smtpd    21042  0.0  0.2  1224  2464 p2  S+     7:22PM    0:00.67 smtpd:
smtp server (smtpd)
_smtpd    21956  0.0  0.1  1020  1384 p2  I+     7:22PM    0:00.00 smtpd:
control process (smtpd)
_smtpd    31724  0.0  0.2  1068  1604 p2  S+     7:22PM    0:30.42 smtpd:
runner (smtpd)
```

`newaliases' command is a hard link to the makemap utility which operates in a mode able to check correctness of the aliases database. At the time of this writing, makemap is also used to handle the virtual users database, but a small redesign of the makemap utility is in the works to let us use maps for various other features.

The SMTPctl utility is used to control and interact with the SMTP daemon. The utility currently allows the following:

- Pausing and resuming processes
- The administrator can temporarily stop local deliveries, remote deliveries, or incoming sessions. They can be paused and resumed independentely so that it is possible to stop relaying outgoing messages while still accepting the incoming sessions
- Live statistics display
- The administrator can request the display of various runtime counters which can be useful for troubleshooting and understanding how the server is being used. Statistics look as follows (see Listing 2).

The administrator can also request the display of queue-related information such as a list of messages currently in queue or currently scheduled. There is still work being done on this area, but the output is not likely to go through heavy changes (see Listing 3).

The fields being: delivery method, unique id, sender, recipient, timestamp, and the number of times we attempt delivery for this message. Each time the messages are scheduled for delivery and an attempt is made, the timestamp gets updated so we have a precise idea of when the last time we dealt with it was.

The runqueue, which can be inspected with "show runqueue" contains only the messages which have been marked ready for delivery and will be processed. A "show runqueue" output looks identical to "show queue".

Enqueuer

SMTPctl can operate in a mode where it emulates sendmail in reading a message from its standart input and registering it to the queue, without establishing a network connection to the server. In this mode, mail user agents like the `mail' utility, or `mutt' from ports, can transparently rely on SMTPctl via the mailer.conf(5) mechanism.

Other tools may appear too but so far all of our basic needs are covered with these two utilities and the various hard links to them.

Configuration

From the beginning, we decided to provide a very simple configuration which even a new user could understand upon his or her first read. OpenSMTPD does not roll a custom configuration parser, but uses a pf-like syntax to describe what is to be accepted and what is to be rejected. Describing the configuration file would consist of reading the man page, so I will simply walk you through the various steps of an imaginary setup. This is how I like to get familiar with tools and will allow us to see the different kind of setups that can already be achieved.

Overview of sample configuration file

If we remove aliases, for the sake of simplicity, the default config file has the following rules:

```
   listen on lo0
   accept for domain "localhost"
deliver to mbox "/var/mail/%u"
   accept for all relay
```

This means that SMTPd will listen on the loopback interface, accept mails for users of the "localhost" domain, and accept local users to relay mail. listen could take an address instead of an interface name and a port if we did not want to use the default one:

```
   listen on 127.0.0.1 port 2526
```

Providing the interface name will listen on all INET and INET6 addresses that this interface knows about.

The configuration above is barely usable, it will simply allow local users to relay mail wherever they want, and will only accept mails for recipients that are local.

What if we wanted to do something simple like allowing all of our local users to send mail anywhere, and also accepting mail from the outside from the domain "grazou.poolp.org" ?

Accepting mail for other destinations than localhost

Assuming that my interface is bge0, the configuration file could be changed a bit to also listen `on bge0`:

```
   listen on lo0
   listen on bge0
   accept for domain "localhost"
deliver to mbox "/var/mail/%u"
   accept from all for domain
"grazou.poolp.org" deliver to mbox
"/var/mail/%u"
   accept for all relay
```

Listing 2. smtpctl statistics

```
   % sudo smtpctl show stats|grep '^smtp.'
   smtp.sessions = 4732
   smtp.sessions.aborted = 13
   smtp.sessions.active = 24
   smtp.sessions.ssmtp = 5
   smtp.sessions.ssmtp.active = 0
   smtp.sessions.starttls = 3231
   smtp.sessions.starttls.active = 11
- Queue display
```

Listing 3. smtpctl queue inspection

```
   % sudo smtpctl show queue | grep gilles@openbsd.org
   MTA|1233868410.kCcGFQoOEUq30259.3081509717|gilles@poolp.org|gilles@open
bsd.org|1233868419|0
   MTA|1233868631.cgBppzEFZFE10552.3573212294|gilles@poolp.org|gilles@open
bsd.org|1233868670|0
   MDA|1233868707.XNWYsjRbKbM15852.3848182339|gilles@poolp.org|gilles@graz
ou.poolp.org|1233868715|0
```

At this point, SMTPd listens on both lo0 and bge0 for connections. You may be scared by the "accept for all relay" rule as you'd assume it to apply to bge0 and cause SMTPd to become an open relay, but SMTPd has sane defaults and assumes an implicit "from localhost" rule if there aren't any allowed sources specified. This is why we need to "accept from all" in our second rule: if we didn't specify it then sessions on bge0 would assume relaying is denied. With that in mind, NEVER EVER EVER "accept from all for all relay".

The second rule here allows anyone to send mail to *choupette@grazou.poolp.org* from any address on any interface, and tells SMTPd that it has to deliver the message to `/var/mail/%u`, where `%u` is expanded to the system user the mail has had its envelope resolved to.

The "all" part in "from all" is a keyword that is more explicit than having a netmask, but it is almost (we'll see later why) equivalent to :

```
accept from 0.0.0.0/0 for domain
"grazou.poolp.org" deliver to mbox
"/var/mail/%u"
```

Obviously this means you can use any netmask or address in place of "all", so if I wanted to allow only my local network to use this SMTPd as the final node to cvs.poolp.org, I could use the following rule:

```
accept from 192.168.0.0/16 for
domain "cvs.poolp.org" deliver to mbox
"/var/mail/%u"
```

The "relay" rule that we've seen earlier tells SMTPd that it has to relay the message to another MX host. This is done using a DNS MX records lookup, which SMTPd will use to find which nodes it should try to send the message to. This is how the SMTP protocol works, not a specificity of OpenSMTPD.

Sometimes, however, you want to bypass the MX lookup and force a route to the next node. For example to have your laptop always use your gateway instead of trying to deliver mail itself.

This is done very easily through a "route via" rule:

```
accept for all relay via
"gw.poolp.org"
```

Listing 4. A new makemap utility

```
$ makemap
usage: makemap [-t type] [-o dbfile] file
```

Listing 5. Sendmail-compatible newaliases utility

```
$ sudo
newaliases
/etc/mail/aliases: 48 aliases
```

Listing 6. Enqueuer

```
$ ./send-mail gilles@poolp.org
Subject: foobar
This is a test
^D
$
```

Listing 7. Enqueuer used through the mailer.conf mechanism

```
$ cat /etc/mailer.conf |grep
send-mail
send-mail /usr/libexec/smtpd/send-mail
$ mail gilles
Subject: test
test
```

When a "relay via" directive is declared, SMTPd will only attempt to deliver to the target host, bypassing MX records lookup. At the moment we limit this to one destination, but work will be done to extend this support.

It may seem obvious, but just in case, the gw.poolp.org needs to be aware of this and should have a rule to accept relaying from the internal network:

```
        accept from 192.168.0.0/16
for all relay
```

Adding IPv6 support to our mail server

If you look at our examples so far, none use an explicit address. We have no "listen" directive or "from" rules with an address or a netmask. This is for a simple reason: IPv6 works out of the box. Wherever you can put an address, a netmask, or an interface, you can stick an IPv6 address or netmask. We did not provide any address so SMTPd assumes we want to support inet AND inet6.

IPv6 support works in both incoming and outgoing ways, and is even given the preference when it is applicable.

Adding SSL/TLS support to our mail server

OpenSMTPD knows of two ways to deal with SSL sessions: sSMTP which is just a regular SMTP session over SSL on a dedicated port; and starttls which is the same SMTP session over SSL but negotiated through an ESMTP extension after a regular SMTP session has been initiated.

I will not go through the details of creating certificates as there is a man page already for this. I could challenge you to read OpenSSL documentations, however you'd surely fall into depression, so I'll encourage you to read starttls(8) on OpenBSD's man pages instead.

When SMTPd starts, it sets up its listening interfaces and looks for matching certificates in /etc/mail/certs. If it finds one, then it assumes that there is SSL support on that interface and starts advertising STARTTLS when the client sends EHLO.

Setting up sSMTP is just a tiny bit trickier:

```
   sSMTP listen on bge0
```

Voila! Prepending "sSMTP" to a listen statement will tell SMTPd that we will use sSMTP instead of STARTTLS.

At this point we can already ensure that incoming sessions are handled via a secure channel; however, we also need SSL for outgoing mails. The "relay via" rule we saw earlier can be instructed to use SSL when it has to relay messages:

```
   accept for all relay via sSMTP
"gw.poolp.org"
   accept for all relay via tls
"gw.poolp.org"
   accept for all relay via SSL
"gw.poolp.org"
```

The first rule will only accept relaying if it can establish a sSMTP session to the remote host. The second will only accept relaying if it can establish a regular session and remote host supports STARTTLS. The third only cares if we establish a secure session, through sSMTP or starttls, whichever works. A message will never be relayed through an insecure channel if we declare that relaying has to go through sSMTP.

Authenticating users

This is still experimental code but it does work to some extent and I am the main user of it so far.

A listening interface can be told that it supports authentication using this very simple rule:

```
   listen on bge0 enable auth
```

When "enable auth" is declared, SMTPd advertises `AUTH` on that interface. The support is currently limited to `AUTH PLAIN` and `AUTH LOGIN`, so SMTPd will not advertise `AUTH` unless the interface has support for SSL and the client could initiate a secure session (`EHLO` in sSMTPm or `EHLO` after a `STARTTLS`).

Authentication currently uses the bsd_auth(3) API which allows us to use any backend for which we have a login script written. Well, this is at least true in theory, but I have only tried using system authentication, and a custom sqlite-based login script I wrote.

At the moment, SMTPd assumes that an authenticated user has rights to relay, which may need to be changed in the future. Outgoing authentication is in my todo list and ranks at a high position, but isn't yet supported.

Current state

OpenSMTPD is NOT production ready and will still need a lot of work before I can honestly say "you can run it safely". I have been running it for months, as my primary MX backed up by sendmail as secondary MX, and I believe it can reach a usable state for non-critical service in a very short timeframe, but it still lacks essential features like a flawless mailer daemons support for instance, and something only time can grant us: maturity. Many features are planned, like a milter-like interface, and the use of some persistent external MDA applications for servers that store mail in some db or have them pass through a dedup utility and cannot afford to fork for each delivery.

However almost all of the very basic features are here, including some which I did not discuss in this article because they are being changed as I write: aliases and virtual users support, use of maps to enable outgoing auth, etc.

Configuration will still evolve and it is likely that the examples are going to change in the near future, but this was just an overview. Changes will be documented to the man pages and things will work out of the box when we have a stable code that is linked to the build.

We are confident OpenSMTP can and will fill the needs of most people and the most complex setups will still be able to run sendmail, qmail, or exim, if we do not support the features they need. Unlike what some people tend to think, having choice and alternatives is a good thing.

About the Author

Gilles Chehade is a research and development engineer at French search engine Exalead, as well as a freelance instructor and consultant. In the past, he held various positions as editor, instructor, developer, administrator, and consultant for different companies and educational organizations. After being an OpenBSD user for nearly a decade, he joined the project where he works mostly on userland and network daemon code. You can visit is website at *http://www.poolp.org/~gilles/*

MidnightBSD

MidnightBSD is a desktop operating system for i386 and AMD64 PCs. It is based on FreeBSD, but contains a heavily modified ports system named mports. The next release, scheduled for late 2009, will feature a new package management system, OS installer, Live CD, and support for ZFS.

MidnightBSD 0.3 will be the first release centered on usability improvements. User feedback suggests that the most common problems with BSD on the desktop are installation of the OS, software installation, and support for hardware. We hope to improve the user experience in these areas as well as provide better documentation.

Chris Reinhardt is working on new tools to manage software installation. mport is a command line utility to install, remove, and manage software packages from the console. It is based on a new library, libmport; this library will facilitate development of additional tools such as a graphical version by Caryn Holt, and integration with the new OS installer.

The new installer, minstall, will be a GTK application. It is currently under development and will feature a Live CD environment to test the system prior to installation. In addition to minstall, Lucas Holt has been bringing in useful functionality from FreeBSD and DragonFly.

The mports system has grown to 2,400 ports; it is tested periodically with the cluster donated by Eastern Michigan University's computer science department.

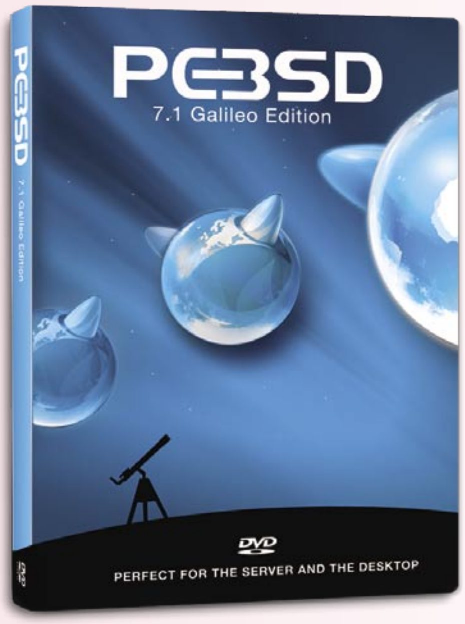
PC-BSD 7.1 Galileo Edition

The latest version of PC-BSD, 7.1 Galileo Edition, was recently released. With faster speeds, better visuals, and more stability, Galileo provides a stellar update for current PC-BSD users. Newcomers to the OS will love the ease of installation that PC-BSD's Push Button Installer (PBI) offers. With KDE's beautifully practical window management tools, and a high level of user-friendliness, Galileo makes it easy to dive into the open source world.

PC-BSD 7.1 is built upon the FreeBSD 7.1-Stable operating system. The Galileo edition includes updated versions of KDE (4.2) and Xorg (7.4). New KDE window effects, screen savers, and better 3D Acceleration make Galileo a visually stunning, yet highly functional, introduction to PC-BSD. The latest version of the Push Button Installer implements PBI Schema 2, which largely improves PBI self-containment to increase reliability. Users may now install FreeBSD Ports without touching the desktop by installing PC-BSD into /PCBSD/local.

If you're looking for the Add / Remove Programs tool, give up. It's not there. Coincidentally, the Update Manager has vanished along with it. Both have been combined under Software & Updates with Galileo. The Updater Tray has been modified as well. It is now merely a tray applet which shows users when updates are available. This is far less taxing on the CPU than its previous functions.

The Galileo edition provides fixes to bugs in the Wi-Fi and Networking tools. It also includes fixes to some previous Linux Emulation problems. The stability of Flash 9 has been greatly improved as well. PC-BSD's System Installer has been enhanced and improved, now with upgrade functionality, for those who wish to install PC-BSD without wiping the disk and losing user data. With these and future updates, the reasons to use PC-BSD continue to increase for new and veteran users alike.



For more information, or to download PC-BSD 7.1 Galileo Edition, visit http://www.pcbsd.org.

Getting a GNOME Desktop on FreeBSD

Jan Stedehouder

Why would you want to install GNOME on FreeBSD? It's a KDE system! This summarizes some remarks I got when checking out how to install the GNOME desktop environment on a FreeBSD box.

There are a few reasons I can think of. For one, I have been using GNOME quite extensively over the last two years and it is a desktop environment I can work with without wondering where function `x` or `y` is. Secondly, the KDE desktop has been undergoing some serious changes since launching KDE 4.0. And while KDE 4.2 is shaping up nicely, it still has some rough edges that stop me from trying it for day to day use, which is especially important since I sometimes need to finish work, instead of playing around with the box to get things working. And finally, there is always the *because it's there* argument. If it can be done, it begs to done.

In this article we will see how the GNOME desktop environment can be installed on a FreeBSD-based box and how the installed desktop compares to some siblings in Linux.

Snags and shortcuts

The first requirement to try out GNOME on a FreeBSD box is a working FreeBSD box. Which I had until I borked it big time. With a rapidly approaching deadline I went for the alternative: getting a prepared virtual machine online.

For this article I used two available virtual machines. On bagvapp.com you can find a few dozen virtual machines, mostly Linux, but also OpenSolaris, FreeBSD 7.1 and, if you are so inclined, Windows 7 beta. The FreeBSD VM is about 900 Mb and takes up 4.5 Gb on your hard drive. It has been tweaked here and there, but it works and it has the looks (Figure 1).

The second virtual machine was already on my box, based on the PC-BSD 7.02 DVD, though you can download a virtual machine directly from the PC-BSD website. It gave an excellent opportunity to test the PBI that delivers GNOME to users (Figure 2).

Method 1: Using the PBI on PC-BSD

The repository for PBI's (*http://www.pbidir.com*) has a package to install GNOME 2.22.3 on your PC-BSD box. PBI's, PC-BSD Installers or push-button installers, are an easy way to install new software. They contain all the needed files and libraries and are self-contained. Installing a PBI doesn't affect the underlying FreeBSD system and the software installed via packages or ports.

The GNOME PBI is 425 Mb and reduces installing the GNOME desktop environment to downloading, double-clicking, entering your root password and following the steps in the wizard. The installation begins with a warning message that this PBI is considered experimental. The next step that requires a user intervention is the question whether the GNOME Display Manager (GDM) should replace the PC-BSD KDE Display

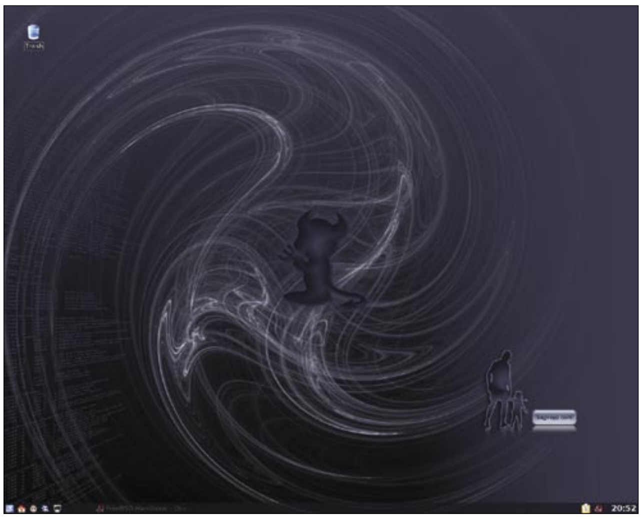


Figure 1. The FreeBSD virtual machine from Bagvapp.com is a pleasant and easy way to try out FreeBSD proper

Manager (KDM). Selecting *no* will add GNOME as option in the Sessions menu of KDM (Figure 3), which is visible after rebooting.

Method 2: Using packages on FreeBSD

Another method, to be used on FreeBSD proper, is outlined on the FreeBSD GNOME page (*http://www.freebsd.org/gnome/*). You can install GNOME either using ports or packages. I can clearly remember taking the ports route to get the GNOME desktop environment and considered it a bit too time consuming for this article. Installing the desktop via packages is simple enough. You need to open a terminal and give yourself root rights with:

```
# su
```

and entering your root password.

To install GNOME you enter:

```
# pkg_add -r gnome2
```

Once this is finished you repeat this for additional collections,

- gnome2-fifth-toe
- gnome2-power-tools
- gnome2-office
- gnome2-hacker-tools

The fifth-toe collection contains programs like Pan (newsgroups), Liferea (RSS feeds), Xchat (IRC), Pidgin (IM), Bluefish (web developement), Galeon (browser), Inkscape (vector graphics) and GIMP (raster graphics). The office collection provides the GNOME office applications like Abiword and Gnumeric. Both power-tools and hacker-tools are geared towards the more adventurous users.

Installing GNOME via packages was hardly a problem. Granted, the need to use the commandline would shy away users coming from Windows, but wouldn't be big deal for somewhat more experienced Linux users. The new desktop environment was added automatically to the existing KDM of the Bagvapp virtual machine.

The solution was to change the PACKAGESITE environment to so-called Tinderbox. For this you need to open a terminal. Then, as user, enter:

```
#export PACKAGESITE=http://
www.marcuscom/tb/packages/7.1-FreeBSD/
Latest
```

After that, give yourself root rights and install the various gnome2 packages.

This isn't the case with less-tweaked FreeBSD installs. There you have to manually edit the `/etc/rc.conf` file and add the following line:

```
gnome-enable="YES"
```

to start up the needed services. There was only one issue. The FreeBSD GNOME page states that GNOME 2.24 is availabe (GNOME 2.24.3 on the Freshports website) and that using `#pkg_add -r gnome2` should install this version. However, it downloaded version 2.22. It's a minor annoyance on which I spend some time trying to fix it.

Method 3: Using packages on PC-BSD

The attempt to install GNOME via packages on the PC-BSD box was halted by the inability to fetch the packages. First, I needed to change the PACKAGESITE environment to the latest 7.1-release.

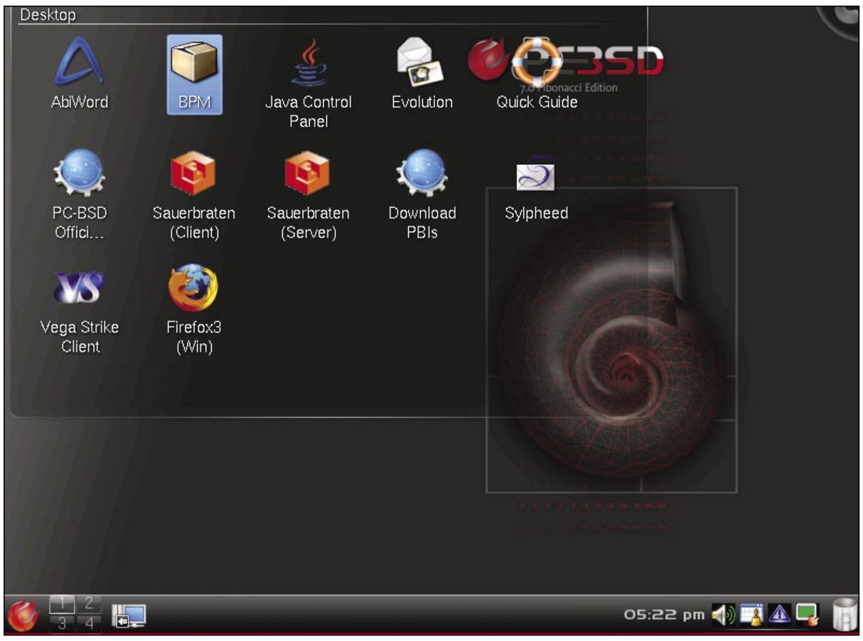


Figure 2. The PC-BSD virtual machine

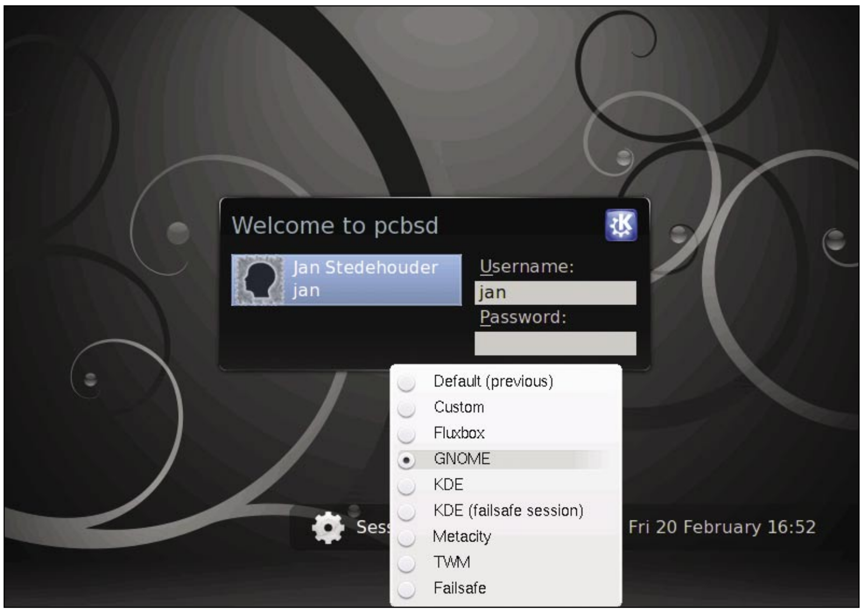


Figure 3. After installing the GNOME PBI package the new desktop environment is available as choice in the Sessions menu

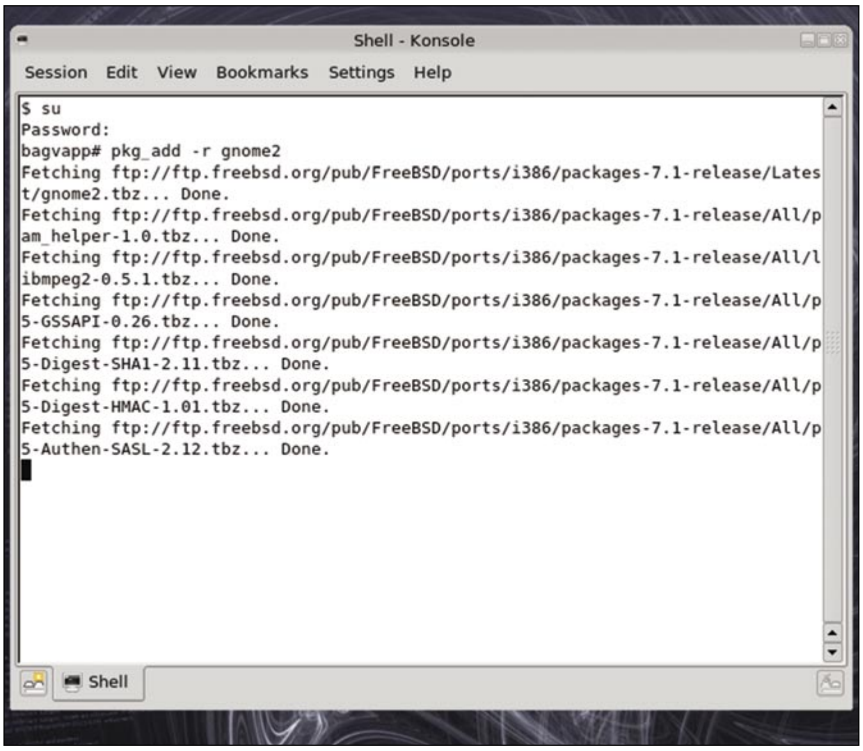
Under PC-BSD you open a terminal (e.g. Konsole), give yourself root rights, and then enter:

```
# setenv PACKAGESITE ftp://
ftp.freebsd.org/pub/FreeBSD/ports/
i386/packages-7.1-release/Latest/
```

After that you use:

```
# pkg_add -r gnome2
```

to install the GNOME desktop. Actually, I tried both 7.0-release and 7.1-release, but each resulted in warnings about dependencies, sometimes resulting in failures to install a package. The dependencies referred to versions that were slightly younger or older than available in the release. Both times I didn't get a working GNOME desktop, which made me feel glad to have used cloned virtual machines for this set of experiments.



```
$ su
Password:
bagvapp# pkg_add -r gnome2
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-7.1-release/Lates
t/gnome2.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-7.1-release/All/p
am_helper-1.0.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-7.1-release/All/l
ibmpeg2-0.5.1.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-7.1-release/All/p
5-GSSAPI-0.26.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-7.1-release/All/p
5-Digest-SHA1-2.11.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-7.1-release/All/p
5-Digest-HMAC-1.01.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-7.1-release/All/p
5-Authen-SASL-2.12.tbz... Done.
```

Figure 4. Installing the GNOME desktop environment via packages is quite painless

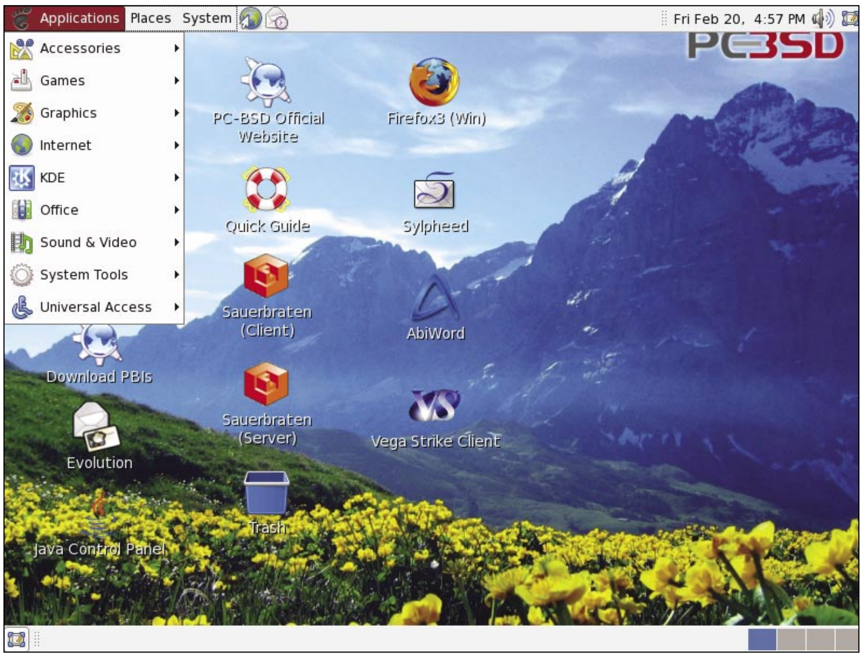


Figure 5. The GNOME desktop on PC-BSD

Looks, feels and comparisons

Installing the desktop environment via PBI (on PC-BSD) or packages (on FreeBSD) both result in vanilla GNOME desktops with their clean panels and menu's. The PC-BSD desktop, which seemed quite organized while using KDE, re-appeared with a cluttered desktop (Figure 5)

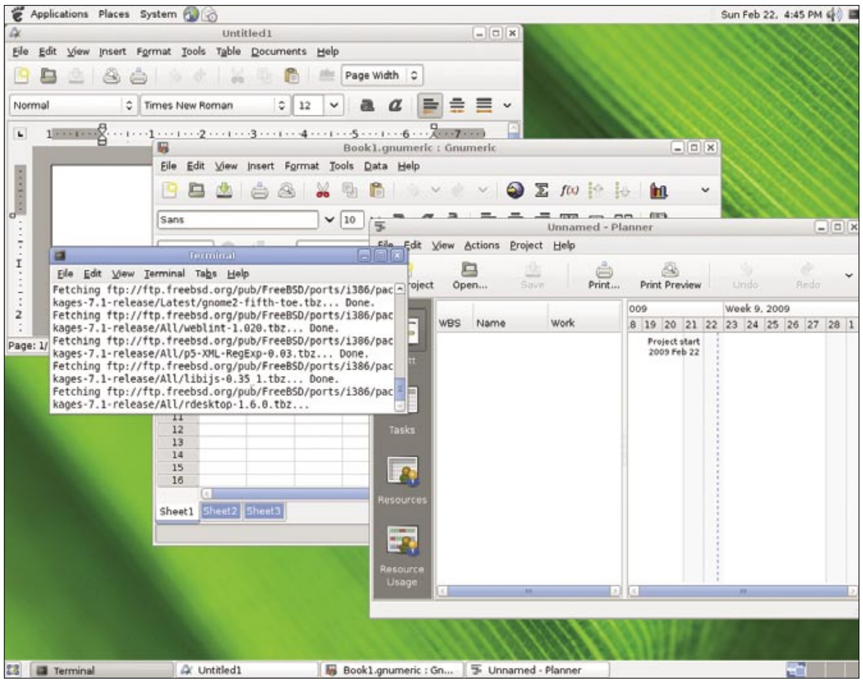
One of the things I immediately liked was the separate KDE entry in the applications menu. On my Ubuntu box, with three desktop environments (GNOME, KDE and Xfce), the KDE applications are mixed with all the other applications. This makes for a very full menu tree, so it was nice to see a separate KDE entry. The top panel contains entries to Applications (where you can find your... well, applications), Places (shortcuts to folders and partitions) and System, which offers access to various tools for settings and management tasks. Anyone who has some experience with GNOME desktops would feel at home. The desktop might look a bit plain, but a trip to *www.gnome-look.org*, where loads of themes and iconsets are available, should solve that.

When you install all five meta-packages (*gnome2*, *gnome2-fifth-toe*, *gnome2-office*, *gnome2-hacker-tools* and *gnome2-power-tools*) you get a complete environment for both mediocre and more advanced tasks. Personally, I don't like all of the choices that were made for GNOME 2.24. For instance, replacing Pidgin as the default IM-client with the Empathy IM-client didn't cut it for me. It wasn't as stable as I want it to be, but as long as I can install Pidgin alongside it I don't mind it's there. Ekiga (formerly GnomeMeeting), the open source alternative to Skype, has reached version 3.0, so videoconferencing is now possible. Abiword and Gnumeric are two light-weight but fully functional programs for wordprocessing and spreadsheets, and Evolution is a powerful program for e-mail and calenders. And, if you do like the GNOME desktop, but not the GNOME-based programs, you can continue working with the KDE-based alternatives.

This doesn't mean all is well. BPM, the graphical front-end to install ports under PC-BSD, wouldn't launch on the GNOME desktop, nor would the System Manager. Both programs require the kcmshell and this appears not to work under GNOME. The Bagvapp virtual machine, FreeBSD 7.1 proper (Figure 6), wouldn't allow me to use the functions under *System›Administration*, functions that would normally result in a request to enter the root password.

The GNOME desktop is used as default by quite a few Linux distributions, like Ubuntu, OpenSUSE, and Fedora (Figure 7). For each of them it isn't a problem to install and use the KDE desktop. What are the major differences between the FreeBSD GNOME desktop and these three others? For starters, each of these Linux distributions has graphical frontends for various management tasks like installing and removing software and managing users.

How far the GNOME desktop can be customized is shown by OpenSUSE (Figure 8). Instead of two panels (one at the top and one at the bottom of the screen), there is one at the bottom. Clicking on *Computer* reveals the *slab*, the default menu panel, with an overview of favorite and recently used applications.



```
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/pac
kages-7.1-release/Latest/gnome2-fifth-toe.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/pac
kages-7.1-release/All/weblint-1.020.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/pac
kages-7.1-release/All/p5-XML-RegExp-0.03.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/pac
kages-7.1-release/All/libijs-0.35_1.tbz... Done.
Fetching ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/pac
kages-7.1-release/All/rdesktop-1.6.0.tbz...
```

Figure 6. The GNOME desktop on FreeBSD 7.1

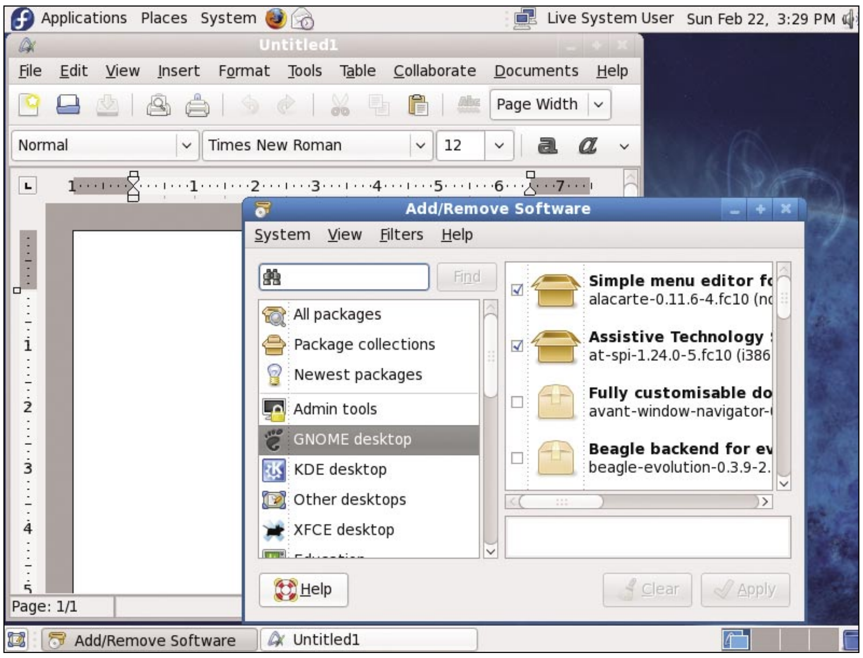


Figure 7. The GNOME desktop on Fedora 10

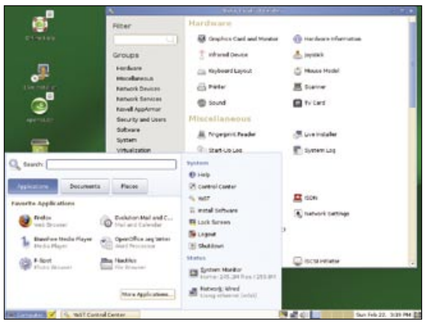


Figure 8. OpenSUSE has a customized GNOME desktop

Conclusions

Getting GNOME up and running on your FreeBSD-based box doesn't require much. When you are using PC-BSD it is enough to get the PBI and on FreeBSD proper the packages are waiting. Granted, installing packages does require some commandline skills, but either way, you have a functional GNOME desktop in less than half an hour. It was a bit disappointing not being able to install GNOME via packages on PC-BSD, after having to change the PACKAGESITE environment in order to get the packages in the first place.

However, both PC-BSD and FreeBSD users can get a vanilla GNOME desktop and almost all tools they need in order to get work done. What is lacking the most for perfect end-user satisfaction is a good default graphical tool for installing and removing software, either for the ports or the packages, (though I do have a slight preference for the packages, since it makes for a faster install).

One thought did come to the fore while working with the GNOME desktops. The KDE desktop is progressing rapidly (and I did look at the KDE 4.2 desktop on PC-BSD 7.1 alpha 1 while playing around for this article) and the GNOME desktop is a mature, solid and complete environment. Both desktops still have issues that need to improve in order to be end-user friendly. Mind you, I define end-user as a non-technical user that works with computers to get tasks done. But, the level of maturity is such that both GNOME and KDE are fine desktop environments regardless of the underlying operating systems, be they Linux or BSD (perhaps even OpenSolaris). This might not suit the evangelists of the various open operating systems, but it does open new avenues for new groups of FreeBSD users.

Packaging Software for OpenBSD – Part 2

Edd Barrett

In the last article in this series, we looked at a simple OpenBSD port. Now we will move on to some more advanced features provided by the ports system in order to package software with more complex needs.

After reviewing the ports I could have used as an example, I decided it would be more effective to introduce the features individually, rather than to introduce a very complex port encompassing all features.

Dependencies

Often a piece of software requires the functionality of another package in the ports tree. You can define a number of dependencies in your port's `Makefile` and the ports system will ensure the necessary software is available. Dependencies manifest themselves in 4 ways:

- `BUILD_DEPENDS` – Programs which are needed at build time of the package, but not at run time. Build dependencies are installed before the port starts building.
- `LIB_DEPENDS` – Shared libraries which the program links. These get installed before the port is built and are needed at runtime too.
- `RUN_DEPENDS` – Programs which are needed at runtime for the software to work. These get installed at package install time.
- `WANTLIB` – Indirectly linked shared libraries and system libraries. By this we mean libraries linked by other library dependencies and libraries in the base system which are not provided by ports.

The format of these variables is well explained in the `bsd.port.mk(5)` manual page, which by now you should have realized is a very valuable resource for porters.

It is very important that you take some time to check that your port will not link any unexpected libraries that the GNU configure script may automatically pick up, as this will lead to your binary package linking different libraries depending upon what libraries the build machine has installed. To be safe you can disable all features you never want enabled using `CONFIGURE_ARGS`. Usually you can use `--without-xxx` or `--disable-xxx` to disable optional features and similarly `--with-xxx` or `--enable-xxx` to ensure certain functionality is built in to the software you are building. In general be as explicit as possible. Dependency examples:

```
LIB_DEPENDS =     mad.>=2::audio/libmad
BUILD_DEPENDS =   ::devel/cmake
RUN_DEPENDS =     ::devel/ectags
```

The command make `port-lib-depends-check` can be used to check for missing/extra library declarations in your `Makefile`. Take a look at the `library-specs(7)` and `packages-specs(7)` manual pages for further information on port dependencies.

Patching

If a port requires modifications to it's source code in order to build on OpenBSD then you will need to use the patching facilities of the ports system.

Adding a patch to a port is simple. Consider we want to make a patch to the `configure` script of a piece of software:

- First extract the port and apply any already existing patches using `make patch` in the port's directory.
- Now `cd` into the sources, which are inside the `WRKDIR`, for example version 2.0.9 of a port named 'nano' would usually extract it's sources under `w-nano-2.0.9/nano-2.0.9`.
- Copy the existing version of the file we wish to patch to a new file with `.orig` appended. Eg. `cp configure configure.orig`.
- Now in the port's directory, run `make update-patches`. Ports will now generate a patch for your changes and inform you that it is about to launch you into an editor. You can

now hit enter and review the patch. You will find the patch in the `patches` directory of the port.

Ports which Install Shared Libraries

Some ports will attempt to install shared libraries, in which case some special handling by ports is required.

First of all you should inform ports which libraries the port is going to install, this is fairly straight forward and will be explained using the *gettext* (internationalization library) port (see Listing 1).

As you can see, gettext installs 5 shared libraries, but what are the numbers all about? When a port with shared libraries is first included in the OpenBSD ports tree, it's shared library version starts at 0.0. Subsequent updates to such a port will then have it library versions *bumped* and the rules for doing so are well documented at *http://www.openbsd.org/porting/libraries.html* . The numbers in the comments are the release versions as per the library author's release, not the OpenBSD library versions. It is considered good practice to comment `SHARED_LIBS` in this way.

The next thing to check is whether the port uses GNU libtool to help generate shared libraries. The tell tale sign of this is that there are scripts called `libtool` dotted around in the build directory after `make configure` is complete. You could use `find . -name 'libtool'` to verify this. You may also wish to log the output of a port build (`make build 2>&1 | tee LOG`) and search inside the log for libtool invocations. If you find your port using libtool, be sure to set `USE_LIBTOOL = Yes` in your port `Makefile`, causing OpenBSD's custom `/usr/local/bin/libtool` to be used instead. If you don't do this, the library versions declared with `SHARED_LIBS` may not be correct.

After the above steps, `make update-plist` should create a file called `PFRAG.shared` in the port's `pkg` directory, listing shared libraries to be installed.

Listing 1. Gettext

```
SHARED_LIBS +=  intl                    4.0      # .8.2
SHARED_LIBS +=  asprintf                1.0      # .0.0
SHARED_LIBS +=  gettextlib              2.0      # .0.0
SHARED_LIBS +=  gettextsrc              2.0      # .0.0
SHARED_LIBS +=  gettextpo               3.0      # .4.0
```

Listing 2. Tex Live Port subdirectories

```
# $OpenBSD: Makefile,v 1.3 2008/10/21 20:57:57 steven Exp $

SUBDIR += base
SUBDIR += texmf
.include <bsd.port.subdir.mk>
```

Listing 3. Package flavors in the mpd port

```
.if ${FLAVOR:L:Mtremor}
CONFIGURE_ARGS +=       --with-tremor \
   --disable-oggflac \
   --disable-shout
LIB_DEPENDS +=          vorbisidec::audio/tremor
.else

WANTLIB +=              theora
LIB_DEPENDS +=          vorbis,vorbisfile,vorbisenc::audio/libvorbis \
   speex::audio/speex \
   shout::net/libshout
.endif
```

Multiple Packages from One Port

Often it makes sense to for one port to generate a number of binary packages. The ports system provides three methods of doing so: subdirectories, multi-packages and flavors.

Subdirectories in Ports

Subdirectories allow a port to have subdirectories, each with it's own `Makefile`, patches and packing list. Subdirectories are typically useful when a piece of software is comprised of many packages, each distributed in separate source tarballs.

Implementing such a port is trivial and is best explained with an example. The TeX Live port uses subdirectories as follows (see Listing 2).

In this example, the folders `base` and `texmf` are processed sequentially when `make` is run in the port's directory. Please note the inclusion of `bsd.port.subdir.mk` is required in order for this to work.

Another feature provided by ports, which can be used along side subdirectories, is the `Makefile.inc` file. This is basically a `Makefile` stub, which gets included in `Makefiles` in subdirectories. How is this useful? Usually the maintainer of each subdirectory is common and can go in to a `Makefile.inc`, for example. You can use any ports system `Makefile` variable in `Makefile.inc`.

Package Flavors

Package *flavors* can be used to make multiple packages from one port when a separate packing list is not required. In other words, flavors enable you to make multiple versions of one package. Most commonly this is to provide packages which are compiled with different features and dependencies enabled. To choose which flavor of a port to build, the `FLAVOR` environment variable is set, for example: `env FLAVOR=no_x11 make install`. If the `FLAVOR` variable is not set, then the default flavor is built.

To make use of package flavors, first a list of possible flavors (other than the default flavor) and the default flavor (`FLAVOR ?=`) must be defined. For example the Music Player Daemon (mpd) port can be built with optional *tremor* support (see Listing 3).

Now you can conditionally execute parts of the port Makefile based upon the flavor the build is requesting, using the `.if ${FLAVOR:L:M<flavor>}` construct. Using the mpd port as an example again (see Listing 3).

In the above example, if the tremor flavor is selected, some `configure` arguments are added to enable tremor support and to disable *oggflac* and *shoutcast* support, then the library specifications are updated accordingly. When using `make port-lib-depends-check`, be sure to run it once for each flavor to avoid library specification errors. See the `bsd.port.mk(5)` manual page for more information on flavors (FLAVORS AND MULTI_PACKAGES section).

Multi-Packages

Multi packages are used when you want to make multiple packages from a single port and each package needs it's own packaging stage. This method is often used to split software from one source tarball into separate counter-pieces, for example server and client packages.

First of all, a list of possible packages is defined. Notice how multi-package names start with a dash, so they are not confused with flavor names. The following example is a snippet from the MySQL port:

```
MULTI_PACKAGES =  -main -server -tests
```

Now multi-package specific variables may defined:

```
COMMENT-main =   multithreaded SQL
database (client)
COMMENT-server = multithreaded SQL
database (server)
COMMENT-tests =  multithreaded SQL
database (regression test suite)
```

The default multi-package name is `-main`, which will generate a package postfixed as such. You may wish to create a more suitable package name (as MySQL does), for example `PKGNAME-main = mysql-client-${VERSION}`.

As mentioned briefly before, each multi-package has its own packaging stage, which implies that each multi-package will have its own packing list and description (optionally also shared library list and (un)install messages). So using the above example we expect at least a `DESCR-main`, `PLIST-main`, `DESCR-server`, `PLIST-server`, `DESCR-tests` and `PLIST-tests` file to exist in the `pkg` directory of the port. As it happens, the server component displays a message using a `MESSAGE-server` file too, but this is purely optional.

Modules

The concept of modules was introduced back in OpenBSD 3.x so that commonly used `Makefile` snippets could be atomically grouped for inclusion elsewhere in the ports tree. The QT4 module is a good example. Ports which build against QT4 always have some common elements such as build and library dependencies, environment variables and configure arguments.

For this reason the common elements were grouped together in `/usr/ports/x11/qt4/qt4.port.mk` and ports wishing to build using QT4 can now simply include this module and not have to worry about duplicating all of the common elements. For example the QCA2 port uses the `x11/qt4` module: `MODULES = x11/qt4`. Further information on port modules is provided in the `port-modules(5)` manual page.

Getting Your Port In-Tree

If you think your port will be useful to other users and has been tested on -current (and if possible, a couple of different CPU architectures), then consider submitting it to the ports mailing list for review. There are a few conventions used here which you should follow:

- The subject line should start with the words NEW or UPDATE then the name of the port, for example `NEW: firefox 3`
- For new ports, which are not already in tree, attach a `.tar.gz` of the port.
- For port updates, mail an inline unified diff.
- Always email in plain text
- Don't top post. See *http://en.wikipedia.org/wiki/Posting_style#Top-posting*

Another thing to note is that if you take maintainership of a port, you will be expected to keep it up to date and fix any issues which may crop up. If you don't want to be held responsible, you can omit `MAINTAINER` from your port `Makefile`. In such a case, the port becomes the responsibility of no-one in particular. Generally ports with maintainers are preffered, as issues can be emailed directly to the maintainer and therefore be addressed quicker.

If your port is of high enough standard, a developer will take it and perform the necessary CVS operations for you, but only once they have been given the *OK* from another developer. Once your port is in tree, binary packages for your port will begin to appear in the snapshot packages directory on the OpenBSD FTP servers.

See the OpenBSD web page on information of how to subscribe to the ports mailing list.

Conclusion

You should now have a fairly good understanding of how to package third party applications for OpenBSD. Of course we have barely scratched the surface. I strongly encourage that developers wishing to get into this seriously, have a good read of `bsd.port.mk(5)` and subscribe to the *ports@openbsd.org* mailing list, to start testing other peoples ports. I hope you learned something or at least found the articles interesting to read. Happy port hacking people!

About the Author

Edd is a BSc Hons Computing student at Bournemouth University in the UK. He is currently working on his final year project, which is an experimental compiler, using the Low Level Machine (LLVM) and Python.

He is also a TeX user and member of the TeX user group (both UK and US). Edd was respobsible for bringing the TeX Live typesetter suite port to OpenBSD.

When away from the keyboard, Edd likes heavy metal and ale with friends. He tells us the Judas Priest, Megadeth, Testament gig last month in London was superb!

MAGAZINE
BSD

A Jabber
Data Transfer component

Eric Schnoebelen

So I can chat, but how do I send a picture to Mom? So, you've got your Jabber server up and running, the family using it, and you're still in contact with your friends on the walled garden networks.

You're having family meetings in using a conference room (never mind that little Sally is off at college, and little Jimmy is doing foreign exchange in Bolivia), and all the family communications are secure.

Now *Little Jimmy* wants to send mom a picture of the wonderful casserole he made. But when trying to do a file transfer directly between the two clients, the transfer bombs out. All the computers at the house are NAT'd behind one router, and much of Bolivia seems to be behind another NAT device. What's the family sysadmin to do?

If you're like most people, your workstations/computers are behind something like a NAT/PAT router mapping all of your client workstations to a single public IP address. While this is wonderful for protecting hosts and conserving IP addresses, it makes point to point file transfer between two client behind such devices nearly impossible.

Enter XEP-0065, The SOCKS5 Byte-streams XMPP extension. It is designed for establishing out-of-band byte-streams between users. The byte-stream can be either direct (peer-to-peer) or mediated (through a proxy server). The mediated model is what is used when both clients are behind NAX/PAT devices.

XEP-0065 is purely for file transfers, and other bulk data transfers. It is supported by a wide variety of clients. For real time audio and video conferencing over XMPP, a different set of protocols is being defined and refined. That protocol suite is called Jingle, and it's development is being supported by Google, as part of Google's GTalk offerings. We'll discuss Jingle further in a future article (as soon as I find a component implementation for Jabberd2). Back to XEP-0065.

Implementing XEP-0065

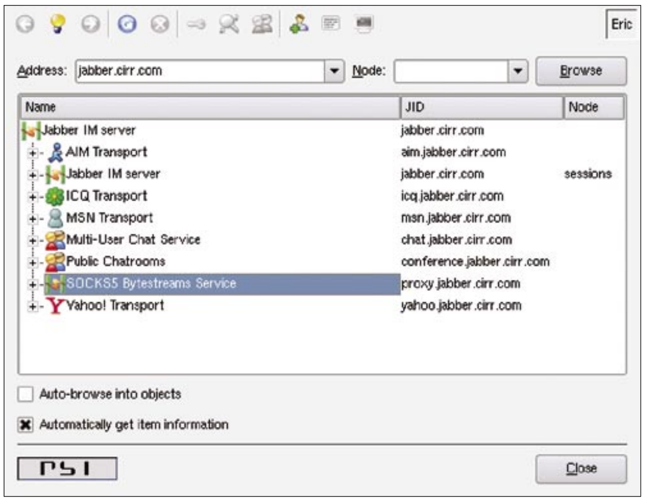
A XEP-0065 proxy server has been implemented in Python, using the Twisted framework. We talked about using Twisted and Twisted's server features when building, installing, and using palaver. Proxy65 (found at *http://proxy65.googlecode.com/files/Proxy65-1.2.0.tgz*) uses the Twisted plug-in/services architecture, just as palaver did.

Obligatory pkgsrc

As you should expect by now, proxy65 can be found in pkgsrc, in the wip category (at the time of writing). It can be found as `py-jabber-proxy65`. Building from pkgsrc is just as with everything else pkgsrc, change to the directory, type `[b]make install`, and you're ready to fly. Skip down to the Configuration section to learn how to configure Proxy65.

The hard way

If you haven't installed any of the previous Twisted applications, then you'll need to download the current edition of Twisted (8.1.0 as of this writing) from *http:*



Name	JID	Node
Jabber IM server	jabber.cirr.com	
AIM Transport	aim.jabber.cirr.com	
Jabber IM server	jabber.cirr.com	sessions
ICQ Transport	icq.jabber.cirr.com	
MSN Transport	msn.jabber.cirr.com	
Multi-User Chat Service	chat.jabber.cirr.com	
Public Chatrooms	conference.jabber.cirr.com	
SOCKS5 Bytestreams Service	proxy.jabber.cirr.com	
Yahoo! Transport	yahoo.jabber.cirr.com	

☐ Auto-browse into objects
☒ Automatically get item information

Figure 1. SOCKS5-service-discovery

//tmrc.mit.edu/mirror/twisted/Twisted/8.1/Twisted-8.1.0.tar.bz2 and install it using the standard Python installation dance of:

```
bunzip2 Twisted-8.1.0.tar.bz2
tar xf Twisted-8.1.0.tar
cd Twisted-8.1.0
python setup.py build
sudo python setup.py install
```

No further configuration of Twisted is needed. Now that Twisted is installed, you need to grab the Proxy65 sources from *http://proxy65.googlecode.com/files/Proxy65-1.2.0.tgz*, and do the same Python installation dance of:

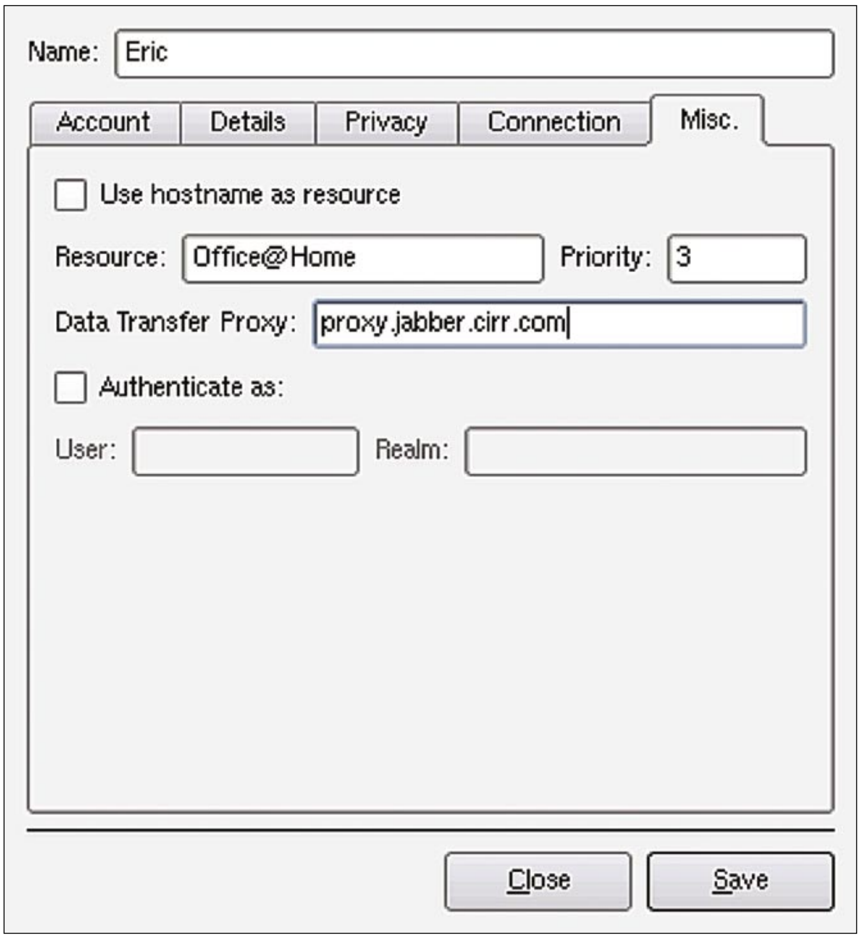
```
gunzip Proxy65-1.2.0.tgz
tar xf Proxy65-1.2.0.tar
cd Proxy65-1.2.0
python setup.py build
sudo python setup.py install
```

Fortunately, for well-written Python modules, build and installation is very straightforward (and both Twisted and Proxy65 are well-written Python modules).

Configuring Proxy65

All the configuration for Proxy65 is done via the Twisted manager command line. If you've built from pkgsrc-wip, a startup script for NetBSD's rc startup system has been installed as `/usr/pkg/share/exampes/rc.d/proxy65`. The installation message describes the variables needed to set it up. To properly configure Proxy65, you need to have the following information:

- The shared secret for talking to your jabberd2 router component. (required)
- A group of address/port pairs to be advertised/used as data transfer addresses/ports (required, must be the public IP address, and the ports must be open)
- The Jabber ID for the proxy server, as a fully qualified domain name. (optional, defaults to Proxy65, unqualified)
- The name of the host where the jabberd2 router component is running (optional, defaults to localhost)
- The port for connecting to the jabberd2 router component (optional,


Name: Eric

Account | Details | Privacy | Connection | Misc.

☐ Use hostname as resource

Resource: Office@Home   Priority: 3

Data Transfer Proxy: proxy.jabber.cirr.com

☐ Authenticate as:

User: Realm:

Figure 2. Psi-Account-Properties-Misc

Listing 1. Begin: Example configuration settings

```
        *   shared secret:   "JabberIsGreat"
        *   address/ports:   192.67.63.14:8160,192.67.63.14:8161
        *   Proxy JID:        proxy.jabber.cirr.com
        *   Jabber host:     jabber.cirr.com
        *   Jabber port:     5347
        *   Log file:         /var/log/jabberd/proxy.log
        *   PID file:         /var/run/jabberd/proxy.pid
        *   user:             jabberd

(Yes, this command line is going to be long!)
```

Listing 2. Begin: twistd command line for Proxy65

```
        sudo twistd \
            --uid=jabberd \
            --logfile=/var/log/jabberd/proxy.log \
            --pidfile=/var/run/jabberd/proxy.pid \
            proxy65 \
            --jid=proxy.jabber.cirr.com \
            --secret='JabberIsGreat' \
            --rport=5347 \
            --rhost=jabber.cirr.com \
            --proxyips=192.67.63.14:8160,192.67.63.14:8161
```

defaults to 6000, jabberd2's router component listens on 5347)

- Path to the file where you want to stash the Proxy65 process id
- Path to the file where you want to have twisted log events related to Proxy65
- Executing user for the Proxy65 component

One addition to your DNS configuration is going to be required. A hostname for the proxy service/server needs to be added to your DNS configuration so outside users can find the proxy server. This needs to be a routable, public IP address.

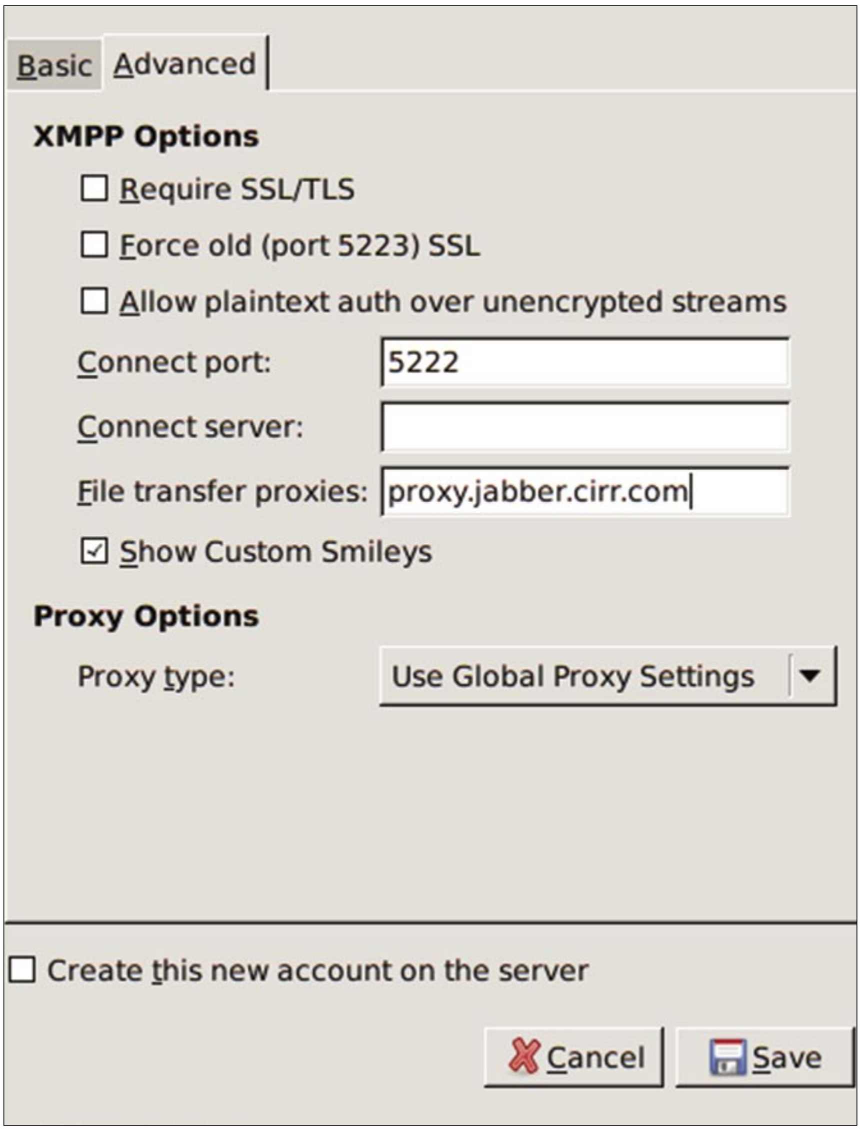
You're also going to need to modify your firewall or NAT device to let the ports listed above be for proxy use. Given the wide variety of devices out there, you'll have to figure that one out on your own.

Assuming the following settings, I'll provide a demonstration command line (see Listing 1).

Twisted breaks the command line up in to two segments. The generic Twisted arguments (user id, log file, pid file) and the Twisted application-specific arguments (in this case, all the Jabber stuff). With that said, here's our command line (see Listing 2).

That's a pretty ugly command line, so you probably want to roll it into a shell script to be used at system boot, or whenever you need to restart the proxy.

With the proxy running, when your favorite Jabber client does service discovery, it should show a new service of *SOCKS5 Bytestreams Service*. See Psi's service discovery screen (Image 1), the new *SOCKS5 Bytestreams Service* is highlighted.



XMPP Options

- ☐ Require SSL/TLS
- ☐ Force old (port 5223) SSL
- ☐ Allow plaintext auth over unencrypted streams

Connect port: 5222

Connect server:

File transfer proxies: proxy.jabber.cirr.com

- ☑ Show Custom Smileys

Proxy Options

Proxy type: Use Global Proxy Settings

- ☐ Create this new account on the server

Figure 3. Pidgin-Advanced-Account

Configuring Clients

The final step to using the proxy service is to configure the clients to use it. That's usually defined on a per server/account basis.

In Psi, the data transfer proxy is defined on a per account basis. On the *Account Properties* window's `Misc.` tab, the `Data Transfe Proxy` setting needs to be filled in with the Jabber ID of the proxy we defined earlier. (in the example, it is proxy.jabber.cirr.com). The screen shot below shows setting the Data Proxy in the `Misc` tab (Figure 2).

In Pidgin, the data transfer proxy is defined on the *Advanced* tab of the *Modify Account* screen. The proxy is defined in the *File Transfer Proxies:* element. The screen shot below shows this tab in Pidgin.

Other IM clients have similar configuration screens and options to set up the data transfer proxy.

To use the proxy, just select *File Transfer* (or *Send File*, or similar) from your chat window, and the client and proxy will do all the work!

Upcoming Topics

In future articles, I'm planning on describing how to implement a web-based Jabber client, implementing a publish-subscribe component for jabberd2, and writing a XMPP bot. If there are Jabber/XMPP topics you'd like to learn more about, let me know, via email as *jabber@cirr.com*, or catch me on XMPP as *eric@jabber.cirr.com*.

About the Author

Eric Schnoebelen is a 25 year veteran of the UNIX wars, using both System V and BSD derived systems.

He's spent more than 20 years working with and contributing to various open source projects, such as NetBSD, sendmail, tcsh, and jabberd2.

He operates a UNIX consultancy, and a small, NetBSD powered ISP. His preferred OS is NetBSD, which he has running on Alpha, UltraSPARC, SPARC, amd64 and i386.

SHORT NEWS

DragonFlyBSD

DragonFlyBSD is a BSD-centric operating system project now in its fifth year of operation. In February DragonFly came out with its 2.2 release, also known as the second HAMMER release. This release contains a large number of stability and performance improvements over 2.0, improved package-source (pkgsrc) compatibility, many new and improved network drivers, and DragonFly's new HAMMER filesystem.

The HAMMER filesystem offers automatic snapshotting, fine-grained history retention, undo, and master-›multi-slave mirroring capability. All functions can be accessed via the live filesystem. The mirroring support includes a non-queued, bandwidth-controlled streaming update capability. It also sports instant boot-time crash recovery. Multi-master clustering and mirroring are still on the drawing board and a year or two away from deployment at the very least, but all other filesystem goals have been met. Unlike most conventional filesystems, HAMMER does not like to delete physical data. Fine-grained historical data retention becomes more coarse-grained during nightly maintainance and deleted data ultimately falls off the disk after its snapshot life is exhausted (typically in the hundreds of days). For this and other reasons HAMMER is designed to operate with large disk partitions. It really only gets comfortable with a few hundred gigabytes and has a design capacity of 1 Exabyte. The filesystem itself utilizes 64 bit dynamically generated inodes, 64 bit file offsets, and a 64 bit byte-offset device API. HAMMER is not a RAID subsystem and does not implement soft-RAID features as would be found in something like ZFS. Ultimately HAMMER is designed to become a cluster filesystem with quorum-based redundancy. Most of HAMMER's on-media data structures revolve around a per-filesystem B-Tree.

Matthew Dillon
www.dragonflybsd.org

FreeBSD Foundation

The FreeBSD Foundation is in our 10th year of supporting the FreeBSD Project and community worldwide! We were founded to fill the need for an outside organization that could support the community's vision and growth. Since then we have been actively involved in supporting three major areas: developer communication, handling legal issues, and funding development projects.

Over the last year we have: Sponsored FreeBSD related conferences like BSDCan, EuroBSDCon, AsiaBSDCon, meetBSD, and NYCBSDCon. We also sponsored FreeBSD developer summits in Ottawa and Cambridge.

Provided 23 travel grants and funding to individuals to attend these conferences.

Provided legal support for the project on issues like understanding the GPLv3 impact on FreeBSD, providing a privacy policy, trademark ownership and permission, and other legal issues that come up.

Provided grants for projects that improve FreeBSD, like Java binaries, Network Stack Virtualization, Improving Hardware Performance Counter Support, making improvements to the TCP stack, Safe Removal of Active Disk Devices, and Improvements to the FreeBSD TCP Stack.

Provided equipment for developers working to improve FreeBSD and projects like the NetPerf cluster.

As a 501(c)3 charity, all of our work is funded by donations. To find out more about what we are doing or to make a donation, please visit *www.freebsdfoundation.org*.

Building a FreeBSD Wireless Router

Eric Vintimilla

Why use a FreeBSD machine as a wireless access point? Don't most Internet Service Providers give you a free modem/router?

While this may be true most of the time, it is not always the case. Besides, building your own is easy, and it gives a great deal of options for both System Administrators and control freaks alike!

Most routers offer some basic functionality, but the possibilities are limitless with a home-built FreeBSD wireless access point. You can set up highly specific packet filtering rules, monitor traffic, email yourself custom reports, and even set up internal bandwidth limits. Plus, being able to SSH into your router is an added bonus!

Requirements

First and foremost, you must have a spare computer with FreeBSD installed. This machine also has to have both a wireless card and a wired NIC. An extra laptop makes a great access point, since it takes up much less space than a desktop computer, especially if you stand them on their side. Software requirements will vary depending on what added features you wish to have, but a basic setup will require `pf`, `bind`, `isc-dhcp40-server`, and `hostapd`.

Installing the necessities

In order to make our wireless access point work properly, we will have to add a couple of packages to our system. First, check to make sure that *bind* is installed:

```
[blendax@moe ~]# named -v
BIND 9.4.2
```

If you get a *command not found* message, then you'll have to add it:

```
   [blendax@moe ~]# sudo pkg_add -r bind9
Once the package is added, check for hostapd (which is
part of the FreeBSD base):
[blendax@moe ~]# hostapd -v
hostapd v0.5.8
User space daemon for IEEE 802.11 AP management,
IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator
Copyright (c) 2002-2007, Jouni Malinen <j@w1.fi> and
contributors
```

If you get an error message, you most likely have a minimal install. You will either have to add it to your system by using *sysinstall* to add some distribution sets or you can ftp into *ftp.freebsd.org*

According to the FreeBSD Handbook, since the release of FreeBSD 5.3, PF has been included in the basic install as a separate run time loadable module. The system will dynamically load the PF kernel module when the `rc.conf(5)` statement `pf_enable='YES'` is present. However, we want *pf* to use the ALTQ framework, which is used for queuing network packets.

In order to enable this, we'll have to customize our kernel to include *pf* support. Luckily, this task is not as bad as it sounds. First, make sure you have the kernel source code.

It can be found in `/usr/src/sys`. If you don't have it, you'll have to get the latest source using your favorite method of source synchronization.

Once you have the latest source, go to kernel configuration directory. Then, make a copy of the GENERIC source, since we're going to use that as our base.

```
[blendax@moe ~]# cd /usr/src/sys/`uname -m`/conf
[blendax@moe ~]# cp GENERIC CUSTOM
```

Now, edit the CUSTOM file and add the following lines to the end of it: see Listing 1.

If you do not want to use ALTQ, you can omit the last seven options.

Now, we're ready to recompile! Start by typing the following:

```
[blendax@moe ~]# cd /usr/src
[blendax@moe ~]# make buildkernel
KERNCONF=CUSTOM
[blendax@moe ~]# make installkernel
KERNCONF=CUSTOM
```

Once it's finished, you will have to reboot:

```
   [blendax@moe ~]# shutdown -r now
```

That is it for the kernel recompilation!

Setting up your wireless card

If you already have your wireless card working properly, you can skip this step. Otherwise, the first thing you will need to know is what kind of wireless card you have. More specifically, you'll need to know what kind of driver your card uses. Usually, wireless cards will use either the *ath* driver for Atheros hardware or the *wi* driver for those based on the Lucent Hermes, Intersil PRISM, and Symbol Spectrum24 chipsets.

Check the man pages for these drivers for a comprehensive list of supported hardware. If your wireless card does not fall into either of these categories, your best bet is to go to the manufacturer's Web site and look for FreeBSD drivers. If they do not offer them, you'll have to download the Windows drivers and use FreeBSD's handy *ndisgen* tool to convert them. For example, I have a really old laptop that uses a Dell TrueMobile 1350 PCMCIA Wireless Adapter (don't laugh). Luckily, ndisgen worked like a charm. Check its man page for more information.

For the rest of this article, I will be using the *wi* driver. If your hardware requires the ath driver, the steps should be similar. The first step is to load the kernel module. To do this without rebooting, type:

```
   [blendax@moe ~]# kldload if_wi
```

However, we want this to automatically turn back on in case we have to reboot our machine, so enter the following line to `/boot/loader.conf`:

Listing 1. Kernel configuration

```
device pf
device pflog
device pfsync
options         ALTQ
options         ALTQ_CBQ
options         ALTQ_RED
options         ALTQ_RIO
options         ALTQ_HFSC
options         ALTQ_PRIQ
options         ALTQ_NOPCC
```

Listing 2. Newly created wireless interface

```
wi0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0      mtu
1500
   ether 00:07:ca:01:e4:9a
   inet 192.168.0.10 netmask 0xffffff00 broadcast 192.168.0.255
   media: IEEE 802.11 Wireless Ethernet DS/11Mbps mode 11b <hostap>   (DS/
2Mbps <hostap>)
   status: associated
   ssid freebsdAP channel 1 (2412 Mhz 11b) bssid 00:07:ca:01:e4:9a
   stationname "FreeBSD WaveLAN/IEEE node"
   authmode OPEN privacy MIXED deftxkey UNDEF wepkey 1:40-bit
   scanvalid 60 dtimperiod 1
```

Listing 3. DHCPD configuration

```
   ### GLOBAL SETTINGS
ddns-update-style none;
always-broadcast on;
default-lease-time 7200;
max-lease-time 7200;
authoritative;
option domain-name-servers 192.168.1.1;
option domain-name "localnet.localdomain";
option netbios-name-servers 192.168.1.1;

### Wired Network
subnet 192.168.0.0 netmask 255.255.255.0 {
        range 192.168.0.100 192.168.0.199;
        option broadcast-address 192.168.0.255;
        option subnet-mask 255.255.255.0;
        option routers 192.168.0.1;
}

### Wireless Network
subnet 192.168.1.0 netmask 255.255.255.0 {

        # NOTE: See: wired->range.notes
       range 192.168.1.100 192.168.1.199;
       option broadcast-address 192.168.1.255;
       option subnet-mask 255.255.255.0;
       option routers 192.168.1.1;
}
```

```
If_wi_load="YES"
```

Another module that is required is *wlan*, which offers generic support for 802.11 drivers.

This driver is automatically loaded when you load *wi*. Unfortunately, there are other drivers that we need, and we will have to reboot our machine. Add the following to `/boot/loader.conf` and then reboot:

```
wlan_scan_ap_load="YES"
wlan_scan_sta_load="YES"
wlan_wep_load="YES"
wlan_ccmp_load="YES"
wlan_tkip_load="YES"
```

`wlan_scan_ap` provides AP mode scanning and `wlan_scan_sta` provides STA mode scanning. The last three modules provide WEP support, AES-CCMP support, and TKIP (WPA) support. If you do not want to set up any type of security (for example, if you actually want your neighbors to have free network access), then you can exclude these. In this article, we are going to use WEP protection (although, WPA is definitely better).

Creating your access point

Assuming your wired connection is already configured, you can set up your access point by typing the following (of course you can change the SSID and WEP key to whatever you wish):

```
[blendax@moe ~]# ifconfig wi0 inet
192.168.0.1 netmask 0xffffff00
ssid freebsdAP wepmode on wepkey
0x1234567890 media DS/11Mbps mediaopt
hostap
```

If it worked, after you type ifconfig, you should something like: see Listing 2.

Next, we're going to enable forwarding between interfaces and turn on the packet filter:

```
[blendax@moe ~]# sysctl -w
net.inet.ip.forwarding=1
[blendax@moe ~]# pfctl -e
```

Everything appears to be in order. Right now we have a basic access point with some security. However, we are not done yet. Now, we are going to add some customizations to our wireless router!

Now, we are going to make some customizations to our wireless access point. The first task is to update our packet filter configuration. Copy the second example `pf.conf` to `/etc/`.

```
  [root@moe ~]# cp /usr/share/
examples/pf/faq-example2 /etc/
```

Next, edit the newly created `pf.conf` file. Change the interfaces to your wired networking connection. Once you make your changes, you can load the configuration file into pf by typing:

```
   [root@moe ~]# pfctl -Fa -f /etc/
pf.conf
```

In order to dynamically assign IP addresses, we have to set up a DHCP server. Install the ISC DHCP server:

```
   [root@moe ~]# cd /usr/ports/net/
isc-dhcp40-server
   [root@moe ~]# portinstall -P
```

Now, make the following changes in `/usr/local/etc/dhcpd.conf`: see Listing 3.

We also have to edit `/etc/hostapd.conf`: see Listing 4.

Finally, add the following to rc.conf so the access point automatically restarts if you reboot: see Listing 5.

That's it! After only a few steps, you now have your own customized FreeBSD access point. There are many more things you can do with your wireless router, such as setting up SSH access and email alerts. The possibilities are endless!

Listing 4. Hostapd configuration

```
interface=wi0
driver=bsd
logger_syslog=-1
logger_syslog_level=0
logger_stdout=-1
logger_stdout_level=0
debug=3
dump_file=/tmp/hostapd.dump
ctrl_interface=/var/run/hostapd
ctrl_interface_group=wheel
         ssid=freebsdAP
macaddr_acl=0
auth_algs=1
ieee8021x=0
```

Listing 5. Start the wireless interface on reboot

```
gateway_enable="YES"
hostname="freebsdAP"
ifconfig_rl0="DHCP"
ifconfig_wi0="inet 192.168.0.1 netmask 0xffffff00 ssid freebsdAP wepmode on
wepkey 0x1234567890 media DS/11Mbps mediaopt hostap"
pf_enable="YES"
pf_rules="/etc/pf.conf"
pf_program="/sbin/pfctl"
pf_flags=""
pflog_enable="YES"
pflog_logfile="/var/log/pflog"
hostapd_enable="YES"
named_enable="YES"
dhcpd_enable="YES"
sshd_enable="YES"
```

Open Source Days

Copenhagen

30-31 October 2009

www.opensourcedays.org

CPU Scaling on FreeBSD UNIX

Slawomir Wojciech Wojtczak
(vermaden)

FreeBSD, as many other today's UNIX systems offer scaling of CPU frequency to save power and emit less heat (which indirectly also leads to less power consumption).

Comparing to other solutions like Solaris or Linux implementations, that directly follow Intel's defined C-states and P-states for CPU, FreeBSD goes a bit further by offering the end user every possible frequency that the CPU can run on, this may sound misleading, but things will be simple after reading the next paragraph.

Lets check what steps are offered by Intel on T7300 2GHz CPU: 800, 1200, 1600, 2000. These steps are supported on mentioned operating systems, but FreeBSD offers these on the same CPU: 150, 300, 450, 600, 750, 900, 1050, 1200, 1400, 1600, 1750, 2000 which means that you can save even more power and you have a lot more flexibility on choosing desired frequency. The same applies to desktop Intel microprocessors, that of course support Intel Speedstep technology, while Solaris or Linux use 1600 or 1866 MHz on Intel E6320 model, FreeBSD stars at 250 ...

Turn It On

FreeBSD's `powerd(8)` daemon that is responsible for frequency scaling is disabled by default, to turn it on with default settings (which are pretty good by the way) you need to do two things, enable `powerd(8)` service by adding `powerd_enable="YES"` to `/etc/rc.conf` file, and then start the daemon itself:

```
box# /etc/rc.d/powerd start
Starting powerd.
box#
```

You may check if the powerd(8) daemon is really running by:

```
box# pgrep powerd
893
box#
```

FreeBSD, to support frequency scaling needs to have `cpufreq(4)` compiled into the kernel (which is default from 7.1-RELEASE) or `cpufreq` kernel module loaded if it is not compiled in.

You can customize the `powerd(8)` daemon to switch to higher or lower state at different load of your CPU then the default, you will have to use `powerd_flags` option at `/etc/rc.conf` file, below is my example, I encourage You to read `man powerd` to get all the details.

```
powerd_flags="-i 85 -r 60 -p 100"
```

Now we have `powerd(8)` up and running scaling our processor frequency, to get current values that `powerd(8)` picks up you need to type this:

```
box# sysctl dev.cpu.0.freq_levels
dev.cpu.0.freq_levels: 2000/31000 1750/27125 1600/22000
```

Listing 1. Setting lowest frequency

```
box# sysctl dev.cpu.0.freq
dev.cpu.0.freq: 150
box# sysctl debug.cpufreq.lowest
debug.cpufreq.lowest: 0
box# sysctl debug.cpufreq.lowest=450
debug.cpufreq.lowest: 0 -> 450
box# /etc/rc.d/powerd restart
Stopping powerd.
Starting powerd.
box# sysctl dev.cpu.0.freq
dev.cpu.0.freq: 450
box#
```

```
1400/19250 1200/13000 1050/11375 900/
9750 750/8125 600/6500 450/4875 300/
3250 150/1625
box#
```

Of course we can disable `powerd(8)` and set the frequency that we want to use manually, like that:

```
box# /etc/rc.d/powerd stop
Stopping powerd.
box# sysctl dev.cpu.0.freq=450
dev.cpu.0.freq: 2000 -> 450
box#
```

To sum up, `cpufreq(4)` kernel module allows us to set other then default steps for our CPU and `powerd(8)` is daemon that automatically sets best step based on the current system load to save maximum power.

Setting Lowest Frequency

We can also set minimal step that we want to use with `powerd(8)` we will have to use `sysctl(8)` MIB called `debug.cpufreq.lowest`. We can also set that up in the `/boot/loader.conf` to make it permanent after reboot, but we can also change it on running system, you will only have to restart `powerd(8)` to make it *know* the new lowest frequency setting: see Listing 1.

Setting Highest Frequency

Some laptops get little too hot when running at maximum avialable speed for their processor, also power consumption grows as we use the top steps, by default FreeBSD does not offer a `sysctl(8)` MIB for that, but patch submited by *Boris Kochergin* allows us to set also the highest step that `powerd(8)` see Listing 2.

To apply this patch you will need to do these simple steps:

```
box# cd /usr/src/sys/kern
box# patch < /path/to/patch
box#
```

Now you will have to recompile your kernel (or just the module if you do not have `cpufreq(4)` compiled in), official FreeBSD Handbook will guide you thru this process efficently. After reboot or reloading `cpufreq(4)` module you can now use new `sysctl(8)` MIB called `debug.cpufreq.highest` that you can use

Listing 2. Patch that enables setting highest frequency

```
--- kern_cpu.c.orig      2008-11-08 13:12:24.000000000 -0500
+++ kern_cpu.c  2008-11-08 10:33:18.000000000 -0500
@@ -131,12 +131,16 @@
 DRIVER_MODULE(cpufreq, cpu, cpufreq_driver, cpufreq_dc, 0, 0);

 static int              cf_lowest_freq;
+static int              cf_highest_freq;
 static int              cf_verbose;
 TUNABLE_INT("debug.cpufreq.lowest", &cf_lowest_freq);
+TUNABLE_INT("debug.cpufreq.highest", &cf_highest_freq);
 TUNABLE_INT("debug.cpufreq.verbose", &cf_verbose);
 SYSCTL_NODE(_debug, OID_AUTO, cpufreq, CTLFLAG_RD, NULL, "cpufreq
debugging");
 SYSCTL_INT(_debug_cpufreq, OID_AUTO, lowest, CTLFLAG_RW, &cf_lowest_freq,
1,
  "Don't provide levels below this frequency.");
+SYSCTL_INT(_debug_cpufreq, OID_AUTO, highest, CTLFLAG_RW, &cf_highest_
freq, 1,
+    "Don't provide levels above this frequency.");
 SYSCTL_INT(_debug_cpufreq, OID_AUTO, verbose, CTLFLAG_RW, &cf_verbose, 1,
  "Print verbose debugging messages");

@@ -295,6 +299,14 @@
   goto out;
  }

+       /* Reject levels that are above our specified threshold. */
+       if (cf_highest_freq > 0 && level->total_set.freq > cf_highest_freq)
{
+               CF_DEBUG("rejecting freq %d, greater than %d limit\n",
+                   level->total_set.freq, cf_highest_freq);
+               error = EINVAL;
+               goto out;
+       }
+
  /* If already at this level, just return. */
  if (CPUFREQ_CMP(sc->curr_level.total_set.freq, level->total_set.freq)) {
   CF_DEBUG("skipping freq %d, same as current level %d\n",
@@ -617,8 +629,13 @@
    continue;
   }

-               /* Skip levels that have a frequency that is too low. */
-               if (lev->total_set.freq < cf_lowest_freq) {
+               /*
+                * Skip levels that have a frequency that is too low or too
+                * high.
+                */
+               if (lev->total_set.freq < cf_lowest_freq ||
+                   (cf_highest_freq > 0 &&
+                    lev->total_set.freq > cf_highest_freq)) {
      sc->all_count--;
    continue;
  }
```

to limit the maximum step for `powerd(8)`. Same as for the lowest setting the best place for making it permanent is the `/boot/loader.conf` file. After you have choosen your lowest and highest settings, avialable frequencies showed by `dev.cpu.0.freq_levels` will be now limited to these settings:

```
box# sysctl dev.cpu.0.freq_levels
dev.cpu.0.freq_levels: 1200/13000
1050/11375 900/9750 750/8125 600/6500
box#
```

Below you can see a table that presents power consumption of avialable CPU frequency steps from Intel T7300 processor, all measured using external wattmeter without battery inside the laptop, just on A/C. CPU was of course loaded to 100% using four `python(1)` precesses calculating this: `999999999999 ** 999999999999;` (see Table 1).

I believe that 1200MHz seems the best maximum frequency to use on that specific CPU, all higher ones consume too much power. I also measured idle power consumption, but even the difference between 150 and 2000 is marginal (3W) so only fully loaded power consumption is important.

Using C-states

We now know how to select frequencies that we want to use on our CPU, now it's time to select C-states, that offer various levels of sleeping our CPU (or exact cores) if they are idle for some period of time. Below is the table that lists C-states avialable for T7300 Intel CPU, more recent versions from *Montevina* platform can have even more C-states with even deeper sleeping (see Table 2).

Table 1. Power consumption according to used CPU frequency

MHz	laptop Power Consumption
150	22W
300	22W
450	23W
600	23W
750	24W
900	25W
1050	26W
1200	27W
1400	33W
1750	42W
2000	47W

Table 2. C-states avialable for T7300 Intel CPU

STATE	EXEC	WAKE UP	POWER	PLATFORM	VOLTAGE	CACHE	LOSS OF CONTEXT
C0	yes	0ns	large	normal	normal	no	no
C1	no	10ns	30,00%	normal	normal	no	no
C2	no	100ns	30,00%	no I/O	normal	no	no
C3	no	50000ns	30,00%	no I/O + no snoop	normal	no	no
C4	no	160000ns	2,00%	no I/O + no snoop	C4_VID	yes	no
C5	no	200000ns	N/A	N/A	C4_VID	L2 = 0KB	no
C6	no	N/A	N/A	N/A	C6_VID	L2 = 0KB	yes

To check which steps are supported and avialable on FreeBSD for your CPU run this command:

```
box# sysctl dev.cpu.0.cx_supported
dev.cpu.0.cx_supported: C1/1 C2/1
C3/57
box#
```

So in that case FreeBSD supports `C0` to `C3` states on `T7300` CPU, you can get/set the lowest C-state for each core, you can set them that way:

```
box#  sysctl dev.cpu.0.cx_lowest
dev.cpu.0.cx_lowest: C1
box# sysctl dev.cpu.0.cx_lowest=C3
dev.cpu.0.cx_lowest: C1 -> C3
box# sysctl dev.cpu.1.cx_lowest
dev.cpu.1.cx_lowest: C1
box# sysctl dev.cpu.1.cx_lowest=C2
dev.cpu.1.cx_lowest: C1 -> C2
box#
```

There is a little catch that you need to know about using C-states, if you set all cores to highest C-states, C3 in mine case, your touchpad will have little lag before you will be able to use it (about 1 second) which can be very annoying in the long term, the solution is to set one core to C-state that offers rather low latency to wake up the core, C2 to be precise and all other cores can be set to use the lowest possible C-state to save as much power as possible. To make these settings permanent use as usual the `/boot/loader.conf` file. You can also display C-states usage statistics per core like that:

```
box# # sysctl dev.cpu.0.cx_usage
dev.cpu.0.cx_usage: 0.00% 0.04% 99.95%
box# sysctl dev.cpu.1.cx_usage
dev.cpu.1.cx_usage: 0.00% 100.00%
0.00%
box#
```

Other Settings

You can also lower the kernel's timer frequency by changing the `sysctl(8)` MIB named `kern.hz` from the default 1000 to 100, this can be done only at boot time, so you need to place `kern.hz=100` line in `/boot/loader.conf` and reboot. It is also planned in future FreeBSD versions to make 100 the default value, but we will not see that until 8.0-RELEASE propably. Other thing you can do, that is not really related to CPU is mounting all your filesystems with `noatime` option.

Now several words to AMD precessors owners, FreeBSD also supports frequency scaling on AMD CPUs with Cool'n'Quiet, the AMD's implementation for frequency scaling. It works the same way as we have used it on Intel CPUs in this article.

… and that is all about scaling frequency on your CPU if you run FreeBSD operating system, hope you will find is useful.

About the Author

Sławomir Wojciech Wojtczak is an UNIX administrator focused on FreeBSD and OpenSolaris techlologies, administrator at biggest BSD forums (daemonforums.org), recently finished University of Lodz, Poland with Master of Information Technology degree.

Get your copy of BSD mag
and save $20 of the shop prices

Why subscribe?

- save $20
- 4 issues delivered directly to you
- never miss an issue

3 EASY WAYS to subscribe:

- *visit: www.bsdmag.org*
- *call: + 31 (0) 36 5307118*
- *fill in the form and post it*

BSD magazine ORDER FORM

□ **Yes**, I'd like to subscribe to BSD magazine starting with issue:

□ 1/2008(1) FreeBSD Ins & Outs
□ 2/2008(2) OpenBSD in the Limelight
□ 1/2009(3) Explore NetBSD
□ 2/2009(4) PC-BSD Uncovered
□ 3/2009(5) Guide to FreeBSD
□ 4/2009(6)
□ 1/2010(7)
□ 2/2010(8)
□ 3/2010(9)

Order information
(□ individual user/ □ company)

Title	
Name and surname	
address	
postcode	
tel no.	
email	
Date	

Company name	
Tax Identification Number	
Office position	
Client's ID*	
Signed**	

Payment details:
□ USA $39.99
□ Europe 29.99€
□ World 39.99€

I understand that I will receive 4 issues over the next 12 months.

Contact Information:

EMD The Netherlands - Belgium
P.O. Box 30157
1303 AC Almere
The Netherlands

Phone	+ 31 (0) 36 5307118
Fax	+ 31 (0) 36 5407252

LDAP Authentication on OpenBSD Boxes

Nicolas Grenèche

This article will focus on a remote user / password database (LDAP) where passwords are stored in a hashed form (SSHA). Only client side aspect will be discussed on our favorite operating system: OpenBSD.

A user directory is a user database distributed on the network. One of the very first directories NIS (Network Information Service), was implemented by SUN. NIS is composed of two programs: ypserv and ypbind (yp prefix stands for *yellow page*, a synonym of NIS). Ypserv is the server part of NIS. It is used to share user information (also called maps) across the network. Ypbind is installed on clients. This program connects ypserv to get user information on clients. It has worked perfectly for years but many drawbacks have arisen:

- maps are transferred in plain text on the network
- ypserv does not require any authentication of ypbind to push maps
- password hashes were exposed on maps. A single ypcat passwd" piped in *john the ripper* and the show would begin. Some (relatively) secure implementation of NIS (i.e., NIS+) hide password hashes
- NIS relies on RPC (Remote Procedure Call). Ypbind asks a program referred to as portmapper to connect ypserv. Portmapper replies to ypbind with a random port number used. Ypbind connects on this port to get maps. This is a nightmare for firewalls
- Information on users are limited by UNIX information. Any extra information must be added in GECOS field which is not very convenient.

Thankfully, LDAP has arrived!

LDAP and authentication basics

LDAP (*Lightweight Directory Access Protocol*) is a massively used protocol to store user's information. This protocol is implemented in OpenLDAP, a directory software available on every operating system's package manager. A LDAP directory may contains objects (users, groups, amd maps, sudoers, DNS zones, etc.) defined as schemas (collection of attributes for an object) and stored in a database backend (i.e., BerkleyDB). A user's LDAP record contains his login, uid, gid, GECOS etc. Each UNIX user account is defined as a posixAccount class. The process running on LDAP server is called slapd and is binded on port 389 (`plain/start_tls`) or 636 (`ssl`). Our configuration will rely on `start_tls` to secure connections (authentication of the server against clients and crypted transactions).

Authentication is the process of proving your identity. OpenLDAP handles authentication in two ways:

- The simple authentication: password is stored in user's LDAP record in a hashed form (MD5, SHA1, SSHA, etc.)
- SASL (*Simple Authentication and Security Layer*): a layer to plug various authentication methods (such as Kerberos) to OpenLDAP

The major benefit of using SASL is that you do not store any authentication information in your directory. Storing passwords in a directory may be a non issue for many sysadmins. The drawback is that every application on your network should know how to deal with your external authentication method. Moreover, LDAP authentication is available in many applications and a very effective overlay is implemented in OpenLDAP (ppolicy) to handle passwords strength and aging.

Prerequisites

I am supposing that you have a running OpenLDAP server using `start_tls` on your network. Anonymous bindings are allowed. Slapd configuration is out of the scope of this article. On the client side, some steps should be achieved before using LDAP authentication on an OpenBSD box. First, let's install the client part of OpenLDAP:

```
# export PKG_PATH=ftp://
ftp.openbsd.org/pub/OpenBSD/4.4/
packages/i386
# pkg_add openldap-client
```

This package provides two useful things:

- Headers to compile ypldap (see below).
- Tools to manage LDAP server (`ldapsearch`, `ldapadd`, `ldapmodify`, etc.)

To configure this package, edit `/etc/openldap/ldap.conf`: see Listing 1 and Table 1.. This file is given as a sample. Feel free to modify it to fit your needs. Configuration should be checked by an ldapsearch:

```
# ldapsearch -x -ZZ
```

This command displays publicly accessible LDAP records of our slapd server using a simple authentication (`-x`) as an anonymous user (no binddn) with `start_tls` enabled and mandatory (`-ZZ`).

Authentication process in OpenBSD

To perform authentication OpenBSD relies on BSD authentication (also known as BSD Auth) framework. BSD Auth performs authentication by executing scripts or programs as separate processes from the one requiring the authentication. These two processes speak trough IPC (*Inter Process Communication*). This provides privilege separation: each process runs as an identity which has only the necessary privileges on the system. This behaviour has significant security benefits, notably improved fail-safeness of software, and robustness against malicious and accidental software bugs.

An alternative is to use PAM (*Pluggable Authentication Module*). This software comes from the Linux world and is available on FreeBSD. Modules providing authentication are dynamically linked into the requesting process. This method is considered to be more flexible than BSD Auth but does not provide privilege separation without additional configuration. The same remark applies to NSS (*Name Switching Service*): that's why it is not implemented in OpenBSD.

The LDAP login component is referred to as login_ldap on OpenBSD. To install it:

```
# pkg_add login_ldap
```

BSD Auth configuration is done trough `/etc/login.conf` file. This file lists the configuration of all available login classes for users. Each user account has a login class. For LDAP authentication, an extra class must be added to `login.conf`.

Listing 1. /etc/openldap/ldap.conf

```
BASE dc=example,dc=com
URI ldap://ldapserver.example.com
TLS_CACERT /etc/openldap/ssl/cacert.pem
TLS_REQCERT demand
ssl start_tls
scope sub
bind_policy soft
```

Listing 2. /etc/login.conf

```
ldap:\
        :auth=-ldap:\
        :x-ldap-server=ldapserver.example.com,,starttls:\
        :shell=/bin/ksh:\
        :x-ldap-basedn=dc=example,dc=com:\
        :x-ldap-filter=(&(objectclass=posixAccount)(uid=%u)):\
        :x-ldap-groupdn=ou=groups,dc=example,dc=com:\
        :x-ldap-groupfilter=(&(objectClass=posixGroup)(memberUid=%u)):\
        :tc=default:
```

Listing 3. List of commands to cvs openbsd sources

```
# export CVSROOT="anoncvs@anoncvs.de.openbsd.org:/cvs"
# cd /usr && cvs checkout -P src
# cd /usr/src/usr.sbin/ypldap
# make depend && make && make install
```

Table 1. /etc/openldap/ldap.conf attributes

Attribute	Description
`BASE`	This is the root of your LDAP directory.
`URI`	Contains the FQDN of your slapd server. It also indicates which method is used to connect to slapd: ldap for plain and start_tls session and ldaps for full SSL.
`TLS_CACERT`	Path to CA (*Certificate Authority*) certificate. This certficate is mandatory to check the signature of the certificate supplied by the server.
`TLS_REQCERT`	Specifies what checks to perform on server certificates in a TLS session.
`ssl`	Method of ciphering transactions between client and slapd (none, ssl or start_tls).
`scope`	Level of recursion of LDAP requests.
`bind_policy`	Policy of client binding to slapd.

Table 2. /etc/login.conf attributes

Attribute	Description
`auth`	Name of login class.
`x-ldap-server`	FQDN of LDAP server.
`shell`	Force shell at login (override current shell). Bash is not available on OpenBSD base system (ksh should be used instead).
`x-ldap-basedn`	This is the root of your LDAP directory.
`x-ldap-filter`	Filter used to retreive posixAccount objects from the LDAP directory.
`x-ldap-groupdn`	Root of posixGroup objects.
`x-ldap-groupfilter`	Filter used to retreive posixGroup objects from the LDAP directory.

Edit `/etc/login.conf` and append: see Listing 2 and Table 2.

Import users

User importation can be accomplished in two ways. OpenBSD does not have a NSS (*Name Switching Service*) mechanism like Linux or FreeBSD. Prior to OpenBSD 4.4, all accounts (and groups) had to be recreated on the OpenBSD box with `ldap` as the login class. Users had to be created this way:

```
# /usr/sbin/useradd -d /home/toto -u
1002 -g 1002 -L ldap toto
```

This way, a user can login to OpenBSD box with his LDAP password. A second method appeared in OpenBSD 4.4 and upcomings ones. This method relies on ypldap: a new program that provides yellow page (yp) maps to OpenBSD using a LDAP backend. The first step is to compile ypldap. Upcomings versions of OpenBSD will include a binary form of ypldap in base system. To compile it, grab the cvs of the latest OpenBSD 4.4 source tree: see Listing 3.

If compilation process complains about missing `ldap.h`, check if openldap-client is installed. The next step is to configure ypldap connection to your LDAP server. This can be done through `/etc/ypldap.conf`: see Listing 4.

This file specifies a name for the domain (to stay compliant with ypbind), maps supplied (provide map), LDAP server's FQDN (directory), and attribute mapping for this directory (i.e., uid on map stands for uidNumber in LDAP).

Like NIS, you must add the following lines to passwd and group files: (see Listing 5).

Listing 4. /etc/ypldap.conf

```
domain localdomain
interval 60

provide map passwd.byname
provide map passwd.byuid
provide map group.byname
provide map group.bygid

directory ldapserver.example.com {

basedn "dc=example=com"

passwd filter "(objectClass=posixAccount)"
attribute name maps to "uid"
fixed attribute passwd "*"
attribute uid maps to "uidNumber"
attribute gid maps to "gidNumber"
attribute gecos maps to "cn"
attribute home maps to "homeDirectory"
fixed attribute shell "/bin/ksh"
fixed attribute change "0"
fixed attribute expire "0"
fixed attribute class "ldap"

group filter "(objectClass=posixGroup)"
attribute groupname maps to "cn"
fixed attribute grouppasswd "*"
attribute groupgid maps to "gidNumber"
list groupmembers maps to "memberUid"
}
```

Listing 5. List of commands to permit users appending to local base (like in NIS). All lines starting with '#' are commands. ypldap -dv is also a command and the trailing lines are its output.

```
#vipw
+:::::::::/bin/ksh
#echo "+:::" >> /etc/group
Then you can check if ypldap works:
# ypldap -dv
configuration starting
applying configuration
connecting to directories
trying directory: X.X.X.X
starting directory update
starting directory update
updates are over, cleaning up trees now
flattening trees
pushing line: user1:*:1000:1000:ldap:12011:0:USER 1:
/home/user1:/bin/ksh
pushing line: user1:*:1000:
[...]
```

Listing 6. /etc/rc

```
if [ X`domainname` != X ]; then
        #if [ -d /var/yp/`domainname` ]; then
        #       # YP server capabilities needed...
        #       echo -n ' ypserv';              ypserv
${ypserv_flags}
        #       #echo -n ' ypxfrd';             ypxfrd
        #fi

        #if [ -d /var/yp/binding ]; then
        #       # YP client capabilities needed...
        #       echo -n ' ypbind';              ypbind
        #fi
```

Listing 7. /etc/rc.local

```
# Add your local startup actions here.

if [ X"${ypldap_flags}" != X"NO" ]; then
        echo -n ' ypldap'
        /usr/sbin/ypldap ${ypldap_flags} 1> /dev/null &
        sleep 10
fi

if [ -d /var/yp/binding ]; then
        # YP client capabilities needed...
        echo -n ' ypbind';              ypbind
fi
```

This example shows how user1 (and his user private group) is pushed from the LDAP directory to maps. The last step is to enable ypbind. You must specify a domainname (the same that the one in ypldap.conf) and launch portmapper before running ypbind:

```
# echo localdomain > /etc/
defaultdomain
# portmap
# ypbind
```

Now you can type in `getent passwd` and `getent group` to check if all your accounts and group are in maps. If not, happy debugging!

Automation at startup

Automation of the whole process is a bit tricky. First you must disable default execution of all yp programs such as ypserv, ypbind and ypxfrd. Ypserv and ypxfrd are of no use here. Ypbind must be disabled because if not, it tries to start before ypldap (or ypbind relies on ypldap). Edit `/etc/rc`: (see Listing 6).

Next, add startup process for ypldap and ypbind. This can be done in `/etc/rc.local`: (see Listing 7). The `sleep 10` after ypldap startup is important because it must be started and up before ypbind. Finally, edit /etc/rc.conf.local to enable services startup:

```
portmap=YES
ypldap_flags=""
```

Ypldap provides a flexible and secure way to handle users from LDAP in OpenBSD. Integration of ypldap will be complete in version 4.5.

Listing 8. /etc/ldap/slapd.conf

```
# include the schema
include /usr/share/openldap/schema/ppolicy.schema
[...]
# load policy module
moduleload ppolicy.la
[...]
# enable ppolicy overlay
overlay ppolicy
# policy location
ppolicy_default cn=hardened,ou=policies,dc=example,dc=com
```

Listing 9. hardened_policy.ldif

```
# basic ldap entry
dn: cn=hardened,ou=policies,dc=example,dc=com
cn: hardened
objectClass: pwdPolicy
objectClass: organizationalRole
# name of the password attribute
pwdAttribute: userPassword
# password aging
pwdMaxAge: 604800
# minimum size of 8 characters
pwdMinLength: 8
# 4 passwords in history
pwdInHistory:
# strict control of password quality
pwdCheckQuality: 2
# password can be locked
pwdLockout: TRUE
# permanent lockout of password after 3 attempts
pwdMaxFailure: 3
pwdLockoutDuration: 0
```

Hardening and aging passwords

There are two ways of hardening passwords in an operating system: server side and client side. On the client side, password renewal procedure relies on local mechanisms like PAM (i.e., cracklib). The password policy must be set on each computer of your network. With OpenLDAP, it became possible to centralize on the server the password policy. The overlay ppolicy has been designed for this purpose. All attributes of this policy criteria are defined in the pwdPolicy object (located in the ppolicy.schema file on my Debian OpenLDAP server). This overlay enables password composition restriction and aging. The configuration is stored in your LDAP database backend. To enable it on server side just add the following to slapd.conf as in Listing 8.

The following LDAP entry can be added to server directory as in Listing 9.

You can specify as many password policies as you need. To bind a user with a non-default policy password, just add a pwdPolicySubentry with the dn of your custom password policy. An external password checker can be plugged into each policy by adding the objectClass `pwdPolicyChecker` to a given policy. This overlay implements an IETF draft *Password Policy for LDAP Directories*. As a consequence, this overlay is very likely to become part of the LDAP standard. This way, ppolicy operates on passwords used for LDAP bindings. Many applications use LDAP binding as their authentication.

Conclusions

This article will be a great help for those who want to start with LDAP on OpenBSD. A proper way of retreiving and authenticating users has been awaited for months (or years!!). Security issues with the classical PAM and NSS couple prevented them to be used in OpenBSD. Now, with ypldap and `login_ldap` package, you can perform authentication securely.

About the Author

Nicolas Grenèche – IT security administrator – University of Orléans – France, Ph.D student at Security and Distributed Systems
nicolas.greneche@univ-orleans.fr
http://blog.garnett.fr (in French)
www.sds-project.fr

FreeBSD
and Snort Intrusion Detection System

Svetoslav P. Chukov

What is an intrusion detection system? An Intrusion Detection System or IDS is a software and/or hardware system designed to detect unauthorized attempts to access computer systems through a network such as the Internet.

These attempts can be part of hacker's attack or just unwanted network activity. An IDS cannot directly detect attacks within encrypted network traffic. However, it can alert the network administrator to potential problems within that traffic.

An intrusion detection system can detect many attacks that can compromise the security and trust of a computer system. These attacks target vulnerable services to take over host computers. In order to achieve that they may try brute force attacks to break passwords, or use viruses, Trojan horses, and worms to trick users into surrendering sensitive information.

An IDS is composed of several components: Sensors which detect security events, a Console to monitor events, send alerts and control the sensors, and a central Engine that records events logged by the sensors. Intrusion Detection Systems can use several output engines like database, log files, pipes or network sockets. The last method is especially useful if you have multiple sites and want to track activity at a central location.

There are several ways to categorize systems depending on the type and location of the sensors and the methodology used by the engine to generate alerts. In this article, we will focus on one of these types – The Network Intrusion Detection System.

A network intrusion detection system (NIDS) is an intrusion detection system that tries to detect malicious activity such as denial of service attacks; port scans or attempts to crack into computers by monitoring network traffic.

The NIDS does this by reading all the incoming packets and trying to match the behavior against a signature. For example, a port scan signature is a large number of TCP connections across many ports on several IP addresses.

A NIDS can be used not only for inspecting the incoming and outgoing network traffic. Often local traffic may indicate an ongoing intrusion as well.

One of the important features of the network intrusion detection systems is that they can communicate with other systems. They can, for example, update blacklists of suspected IP addresses or alter firewall rules to block some specific traffic. One such system is Snort.

Basic overview of Snort and where we can use it.

Snort is a free and open source network intrusion prevention system (NIPS) and network intrusion detection system (NIDS) capable of performing packet logging and real-time traffic analysis on IP networks.

Snort provides uses tools such as protocol analysis and content inspection and matching to analyze and detect hacking activity. Some of these tools also can be used to detect and block attacks and probes, such as buffer overflows, stealth port scans, web application attacks, SMB probes, or OS fingerprinting attempts. The software can also be used for intrusion prevention by dropping attacks as they are taking place.

There are several operating modes that are available in Snort. It can be configured to run in the following modes:

- Sniffer mode. In this mode, Snort simply reads the packets off of the network and displays them for you on the console.
- Packet Logger mode. This mode logs the packets to disk.
- Network Intrusion Detection System (NIDS) mode. This mode analyzes network traffic for matches against a user-defined rule set and performs defensive actions based upon what it detects.

Listing 1a. Output of running Snort

```
Running in IDS mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file /usr/local/etc/snort/snort.conf
PortVar 'HTTP_PORTS' defined :  [ 80 ]
PortVar 'SHELLCODE_PORTS' defined :  [ 0:79 81:65535 ]
PortVar 'ORACLE_PORTS' defined :  [ 1521 ]
Frag3 global config:
    Max frags: 65536
    Fragment memory cap: 4194304 bytes
Frag3 engine config:
    Target-based policy: FIRST
    Fragment timeout: 60 seconds
    Fragment min_ttl:   1
    Fragment ttl_limit (not used): 5
    Fragment Problems: 1
Stream5 global config:
    Track TCP sessions: ACTIVE
    Max TCP sessions: 8192
    Memcap (for reassembly packet storage): 8388608
    Track UDP sessions: INACTIVE
    Track ICMP sessions: INACTIVE
Stream5 TCP Policy config:
    Reassembly Policy: FIRST
    Timeout: 30 seconds
    Min ttl:  1
    Options:
        Static Flushpoint Sizes: YES
    Reassembly Ports:
      21 client (Footprint)
      23 client (Footprint)
      25 client (Footprint)
      42 client (Footprint)
      53 client (Footprint)
      80 client (Footprint)
      110 client (Footprint)
      111 client (Footprint)
      135 client (Footprint)
      136 client (Footprint)
      137 client (Footprint)
      139 client (Footprint)
      143 client (Footprint)
      445 client (Footprint)
      513 client (Footprint)
      514 client (Footprint)
      1433 client (Footprint)
      1521 client (Footprint)
      2401 client (Footprint)
      3306 client (Footprint)
HttpInspect Config:
    GLOBAL CONFIG
      Max Pipeline Requests:    0
      Inspection Type:          STATELESS
      Detect Proxy Usage:       NO
      IIS Unicode Map Filename: /usr/local/etc/snort/
unicode.map
      IIS Unicode Map Codepage: 1252
    DEFAULT SERVER CONFIG:
      Server profile: All
      Ports: 80 8080 8180
      Flow Depth: 300
      Max Chunk Length: 500000
      Max Header Field Length: 0
      Inspect Pipeline Requests: YES
      URI Discovery Strict Mode: NO
      Allow Proxy Usage: NO
      Disable Alerting: NO
      Oversize Dir Length: 500
      Only inspect URI: NO
      Ascii: YES alert: NO
      Double Decoding: YES alert: YES
      %U Encoding: YES alert: YES
      Bare Byte: YES alert: YES
      Base36: OFF
      UTF 8: OFF
      IIS Unicode: YES alert: YES
      Multiple Slash: YES alert: NO
      IIS Backslash: YES alert: NO
      Directory Traversal: YES alert: NO
      Web Root Traversal: YES alert: YES
      Apache WhiteSpace: YES alert: NO
      IIS Delimiter: YES alert: NO
      IIS Unicode Map: GLOBAL IIS UNICODE MAP CONFIG
      Non-RFC Compliant Characters: NONE
      Whitespace Characters: 0x09 0x0b 0x0c 0x0d
rpc_decode arguments:
    Ports to decode RPC on: 111 32771
    alert_fragments: INACTIVE
    alert_large_fragments: ACTIVE
    alert_incomplete: ACTIVE
    alert_multiple_requests: ACTIVE
Portscan Detection Config:
    Detect Protocols:  TCP UDP ICMP IP
    Detect Scan Type:  portscan portsweep decoy_portscan
distributed_portscan
    Sensitivity Level: Low
    Memcap (in bytes): 10000000
    Number of Nodes:   36900

Tagged Packet Limit: 256
Loading dynamic engine /usr/local/lib/snort/
dynamicengine/libsf_engine.so... done
Loading all dynamic preprocessor libs from /usr/local/
lib/snort/dynamicpreprocessor/...
  Loading dynamic preprocessor library /usr/local/lib/
snort/dynamicpreprocessor//lib_sfdynamic_preprocessor_
example.so... done
  Loading dynamic preprocessor library /usr/local/lib/
snort/dynamicpreprocessor//libsf_dcerpc_preproc.so...
```

Listing 1b. Output of running Snort

```
done
  Loading dynamic preprocessor library /usr/local/lib/
snort/dynamicpreprocessor//libsf_dns_preproc.so... done
  Loading dynamic preprocessor library /usr/local/
lib/snort/dynamicpreprocessor//libsf_ftptelnet_
preproc.so... done
  Loading dynamic preprocessor library /usr/local/lib/
snort/dynamicpreprocessor//libsf_smtp_preproc.so...
done
  Loading dynamic preprocessor library /usr/local/lib/
snort/dynamicpreprocessor//libsf_ssh_preproc.so... done
  Loading dynamic preprocessor library /usr/local/lib/
snort/dynamicpreprocessor//libsf_ssl_preproc.so... done
  Finished Loading all dynamic preprocessor libs from
/usr/local/lib/snort/dynamicpreprocessor/
FTPTelnet Config:
    GLOBAL CONFIG
      Inspection Type: stateful
      Check for Encrypted Traffic: YES alert: YES
      Continue to check encrypted data: NO
    TELNET CONFIG:
      Ports: 23
      Are You There Threshold: 200
      Normalize: YES
      Detect Anomalies: NO
    FTP CONFIG:
      FTP Server: default
        Ports: 21
        Check for Telnet Cmds: YES alert: YES
        Identify open data channels: YES
      FTP Client: default
        Check for Bounce Attacks: YES alert: YES
        Check for Telnet Cmds: YES alert: YES
        Max Response Length: 256

SMTP Config:
    Ports: 25 587 691
    Inspection Type: Stateful
    Normalize: EXPN RCPT VRFY
    Ignore Data: No
    Ignore TLS Data: No
    Ignore SMTP Alerts: No
    Max Command Line Length: Unlimited
    Max Specific Command Line Length:
       ETRN:500 EXPN:255 HELO:500 HELP:500 MAIL:260
       RCPT:300 VRFY:255
    Max Header Line Length: Unlimited
    Max Response Line Length: Unlimited
    X-Link2State Alert: Yes
    Drop on X-Link2State Alert: No
    Alert on commands: None

DCE/RPC Decoder config:
    Autodetect ports ENABLED
    SMB fragmentation ENABLED
    DCE/RPC fragmentation ENABLED
    Max Frag Size: 3000 bytes
    Memcap: 100000 KB
    Alert if memcap exceeded DISABLED

DNS config:
    DNS Client rdata txt Overflow Alert: ACTIVE
    Obsolete DNS RR Types Alert: INACTIVE
    Experimental DNS RR Types Alert: INACTIVE
    Ports: 53
SSLPP config:
    Encrypted packets: not inspected
    Ports:
      443      465      563      636      989
      992      993      994      995

+++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
1 Snort rules read
    1 detection rules
    0 decoder rules
    0 preprocessor rules
1 Option Chains linked into 1 Chain Headers
0 Dynamic rules
+++++++++++++++++++++++++++++++++++++++++++++++++++

+-------------------[Rule Port Counts]------------------
---------------------
|             tcp     udp    icmp      ip
|     src       0       0       0       0
|     dst       0       0       0       0
|     any       1       0       0       0
|      nc       1       0       0       0
|     s+d       0       0       0       0
+-------------------------------------------------------
---------------------

+-----------------------[thresholding-config]-----------
----------------------
| memory-cap : 1048576 bytes
+-----------------------[thresholding-global]-----------
-----------------------
| none
+-----------------------[thresholding-local]------------
-----------------------
| none
+-----------------------[suppression]-------------------
-----------------------
| none
--------------------------------------------------------
-----------------------
Rule application order: activation->dynamic->pass->drop-
>alert->log
Log directory = /var/log/snort/
Verifying Preprocessor Configurations!
0 out of 512 flowbits in use.
***
```

Listing 1c. Output of running Snort

```

*** interface device lookup found: em0
***

Initializing Network Interface em0
Decoding Ethernet on interface em0

[ Port Based Pattern Matching Memory ]
+-[AC-BNFA Search Info Summary]-----------------------
-------
| Instances        : 4
| Patterns         : 69
| Pattern Chars    : 297
| Num States       : 225
| Num Match States : 69
| Memory           :   10.83Kbytes
|   Patterns       :   1.63K
|   Match Lists    :   1.72K
|   Transitions    :   6.54K
+-------------------------------------------------


        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.8.2.1 (Build 16)  FreeBSD
   ''''    By Martin Roesch & The Snort Team: http:
//www.snort.org/team.html
           (C) Copyright 1998-2008 Sourcefire Inc., et
al.
           Using PCRE version: 7.7 2008-05-07

           Rules Engine: SF_SNORT_DETECTION_ENGINE
Version 1.8  <Build 14>
           Preprocessor Object: SF_SSLPP  Version 1.0
<Build 1>
           Preprocessor Object: SF_SSH  Version 1.1
<Build 1>
           Preprocessor Object: SF_SMTP  Version 1.1
<Build 7>
           Preprocessor Object: SF_FTPTELNET  Version
1.1  <Build 10>
           Preprocessor Object: SF_DNS  Version 1.1
<Build 2>
           Preprocessor Object: SF_DCERPC  Version 1.1
<Build 4>
           Preprocessor Object: SF_Dynamic_Example_
Preprocessor  Version 1.0  <Build 1>
Not Using PCAP_FRAMES
*** Caught Int-Signal
=======================================================
========================
Packet Wire Totals:
   Received:            0
   Analyzed:            0 (0.000%)
    Dropped:            0 (0.000%)
Outstanding:            0 (0.000%)
=======================================================
========================
Breakdown by protocol (includes rebuilt packets):
      ETH: 0          (0.000%)
  ETHdisc: 0          (0.000%)
     VLAN: 0          (0.000%)
     IPV6: 0          (0.000%)
  IP6 EXT: 0          (0.000%)
  IP6opts: 0          (0.000%)
  IP6disc: 0          (0.000%)
      IP4: 0          (0.000%)
  IP4disc: 0          (0.000%)
    TCP 6: 0          (0.000%)
    UDP 6: 0          (0.000%)
    ICMP6: 0          (0.000%)
  ICMP-IP: 0          (0.000%)
      TCP: 0          (0.000%)
      UDP: 0          (0.000%)
     ICMP: 0          (0.000%)
  TCPdisc: 0          (0.000%)
  UDPdisc: 0          (0.000%)
  ICMPdis: 0          (0.000%)
     FRAG: 0          (0.000%)
   FRAG 6: 0          (0.000%)
      ARP: 0          (0.000%)
    EAPOL: 0          (0.000%)
  ETHLOOP: 0          (0.000%)
      IPX: 0          (0.000%)
    OTHER: 0          (0.000%)
  DISCARD: 0          (0.000%)
InvChkSum: 0          (0.000%)
   S5 G 1: 0          (0.000%)
   S5 G 2: 0          (0.000%)
    Total: 0
=======================================================
========================
Action Stats:
ALERTS: 0
LOGGED: 0
PASSED: 0
=======================================================
========================
Frag3 statistics:
        Total Fragments: 0
      Frags Reassembled: 0
               Discards: 0
          Memory Faults: 0
               Timeouts: 0
               Overlaps: 0
              Anomalies: 0
                 Alerts: 0
     FragTrackers Added: 0
    FragTrackers Dumped: 0
FragTrackers Auto Freed: 0
    Frag Nodes Inserted: 0
     Frag Nodes Deleted: 0
=======================================================
```

Preventive Medicine

As hackers become more and more active, it is extremely important for a server to be up-to-date with appropriate security patches. Network security depends on the strength of the weakest link. When a network has weak security in one segment of the network, regardless of whether just a small office or home LAN, then the machines in that network automatically become vulnerable.

As a network administrator, in order to increase the security and stability of the network, I recommend putting up multiple barriers to reduce the risk of network penetration. An ounce of prevention is worth more than a pound of cure.

So, what barriers do I recommend? First, in order to have a stable and secure operating system,

I will demonstrate how to implement Snort using the FreeBSD operating system. It is easy to find "dry" documentation how to use Snort, what its options are and what that options mean. What is rare to encounter is documentation based on real world experience.

An intrusion detection system like Snort is a good tool for protecting networks when it is setup properly. These systems are especially beneficial when is used in combination with optimized operating system like FreeBSD. FreeBSD is preferred choice for servers with requirement for high reliability, such as firewalls, gateways or border machines accessible by internet. You can also use Snort to protect applications. If you know a particular service is vulnerable, Snort can be used to mask the application from attacks on that service. This is similar to patch on the wire technology used in high end security appliances.

I wrote a server application that receives and sends data through a port to other clients in the network. The application had some known weak points. I setup Snort to sniff the traffic, log a message and drop the packet if there was any attempt to exploit the application. This is just a small area where Snort can be useful.

The combination of FreeBSD, its firewall, and Snort can be used for border machines where security is of high importance. For example with the server mentioned before. 3 days after the server started I analyzed the logs and found multiple attempts to subvert the network. The server was setup with SSH and the application. The server logged dozens of attempts to login with usernames "etc", "jack" and "root" user.

Snort was configured to inspect the incoming packets. Then I checked the log file from time to time to collect new information about the IP addresses that "breached the line".

In another example, I setup pppoe to demonstrate the difficulty of internet service providers (ISP). I would advice any internet service providers to use Snort. Basically, ISP could just provide an internet connection to its customers. Unfortunately, there are a lot of customers that use the internet connection as a springboard for hacking, stealing passwords or some other illegal activity. So, in simple words, the ISP has very difficult job.

The provider has to protect its customers from each other and protect their data. From one side the ISP provides service and from the other side this provider has to protection if it wants to keep its customers.

Snort can be used to detect, stop, and report illegal activity and in that case it can make the ISP's life easier. This is just an example how the intrusion detection system like Snort can be useful.

Listing 1d. Output of running Snort

```
=======================
Stream5 statistics:
            Total sessions: 0
              TCP sessions: 0
              UDP sessions: 0
             ICMP sessions: 0
                TCP Prunes: 0
                UDP Prunes: 0
               ICMP Prunes: 0
TCP StreamTrackers Created: 0
TCP StreamTrackers Deleted: 0
              TCP Timeouts: 0
              TCP Overlaps: 0
       TCP Segments Queued: 0
     TCP Segments Released: 0
       TCP Rebuilt Packets: 0
         TCP Segments Used: 0
              TCP Discards: 0
      UDP Sessions Created: 0
      UDP Sessions Deleted: 0
              UDP Timeouts: 0
              UDP Discards: 0
                    Events: 0
=========================================================================
======
=========================================================================
======
=========================================================================
======
Snort exiting
Run time prior to being shutdown was 3.14117 seconds
```

Installing and Using Snort

We are at the point where I should show you how to get snort on you machine.

To add Snort to your system, type the following command:

```
pkg_add snort-2.8.2.1_1.tbz
```

That installed the snort on my system, you should check if you need some other packages to be installed, and it is different for every system, so if the pkg_add program needs more packages you should install them as well.

Then you can focus on your work with snort. Actually the work with it is very

Listing 2. Logs

```
[**] [1:999369:0] A Test log [**]
[Priority: 0]
02/28-18:53:59.755446 52:54:0:12:35:2 -> 8:0:27:BF:DA:3 type:0x800 len:0x3C
192.168.0.1:3128 -> 10.0.2.15:61247 TCP TTL:64 TOS:0x0 ID:27871 IpLen:20 DgmLen:44
***A**S* Seq: 0x1554F001  Ack: 0xA122F39E  Win: 0x2000  TcpLen: 24
TCP Options (1) => MSS: 1460

[**] [1:999369:0] A Test log [**]
[Priority: 0]
02/28-18:53:59.756180 52:54:0:12:35:2 -> 8:0:27:BF:DA:3 type:0x800 len:0x3C
192.168.0.1:3128 -> 10.0.2.15:61247 TCP TTL:64 TOS:0x0 ID:27872 IpLen:20 DgmLen:40
***A**** Seq: 0x1554F002  Ack: 0xA122F3F1  Win: 0x2238  TcpLen: 20

[**] [1:999369:0] A Test log [**]
[Priority: 0]
02/28-18:53:59.768326 52:54:0:12:35:2 -> 8:0:27:BF:DA:3 type:0x800 len:0x28E
192.168.0.1:3128 -> 10.0.2.15:61247 TCP TTL:64 TOS:0x0 ID:27873 IpLen:20 DgmLen:640
***AP*** Seq: 0x1554F002  Ack: 0xA122F3F1  Win: 0x2238  TcpLen: 20

[**] [1:999369:0] A Test log [**]
[Priority: 0]
02/28-18:53:59.769342 52:54:0:12:35:2 -> 8:0:27:BF:DA:3 type:0x800 len:0x3C
192.168.0.1:3128 -> 10.0.2.15:61247 TCP TTL:64 TOS:0x0 ID:27874 IpLen:20 DgmLen:40
***A***F Seq: 0x1554F25A  Ack: 0xA122F3F1  Win: 0x2238  TcpLen: 20
```

simple. There is a configuration file called snort.conf and several rules files.

I have the configuration file in /usr/local/etc/snort/snort.conf and the rules are there also. So, all the files are available /usr/local/etc/snort directory. You can use them at any location that you want, this is not important.

Let's play with Snort!

Run Snort with the following command:

```
snort -c /path-to-your-config-file -de
-l /path-to-your-log-directory
```

That will run snort with configuration file at your `path-to-your-config-file` and log directory at `/path-to-your-log-directory`.

This is some example output that you should see(see Listing 1).

Summary.

Snort is for you if:

- You have a FreeBSD server which is a border machine that is accessible from internet.
- You are ISP and you want to keep your network safe.
- You have some services that you want to protect against bug exploitation.
- You simply have a server that want to be secure.

Probably many people are wondering for the exact reason to use FreeBSD over the other operating systems. Why FreeBSD, why not GNU/Linux for example? In the beginning I said some things about stability and security. I am not saying that the other operating systems are not secure and not stable but FreeBSD has proven itself as one of the top OSs. FreeBSD has optimal and effective support of TCP/IP network. This is a perfect platform for an IDS. One of the important things for such a system is the performance and the way how the OS handles the network packets. You can see that a slow system can not be very effective with a heavy network load. That is because all the packets have to traverse the rules of the IDS and that takes some time. The speed of the packet process drives the productivity of the IDS. This lowers the chance a “bad” packet will get in undetected. A clever and smart hacker can see that you protect the network with Snort and probably can find a way to overcome the system. So, the performance needs to be at a very high level to reduce the chance that an intruder will take advantage of this by using a brute force attack.

Against the hackers, any sign of weakness will be exactly where they attempt to attack. So, I would advice you to use only the strong and reliable combinations if you wish your machines and network to be safe and secure...

About the Author

Svetoslav Chukov is a developer and a system administrator of FreeBSD, NetBSD and GNU/Linux. He is a big supporter of the open source software and also a big friend of *BSD and GNU/Linux. He always tries to show their advantages over other operating systems.

Svetoslav is interested in topics about the performance, stability and security. Svetoslav Chukov believes in the idea of the Open Source.

Build An Embedded Video Web Server With NetBSD

Donald T. Hayford

While it's safe to say that the recently developed USB video driver was built and tested using only a desktop "i386-compatible" machine, the beauty of NetBSD is that the same driver will work on any NetBSD-supported hardware. So grab your favorite embedded processor and let's try some video.

NetBSD is recognized among the different BSD's for supporting a wide variety of processors and single-board computers. Part of the reason for this is the underlying operating system design that abstracts away the specifics of the hardware interface, allowing high-level device drivers to work equally well for all processor configurations. According to David Chisnall, who wrote in NetBSD: Not Just for Toasters (NetBSD: Not Just for Toasters, David Chisnall):

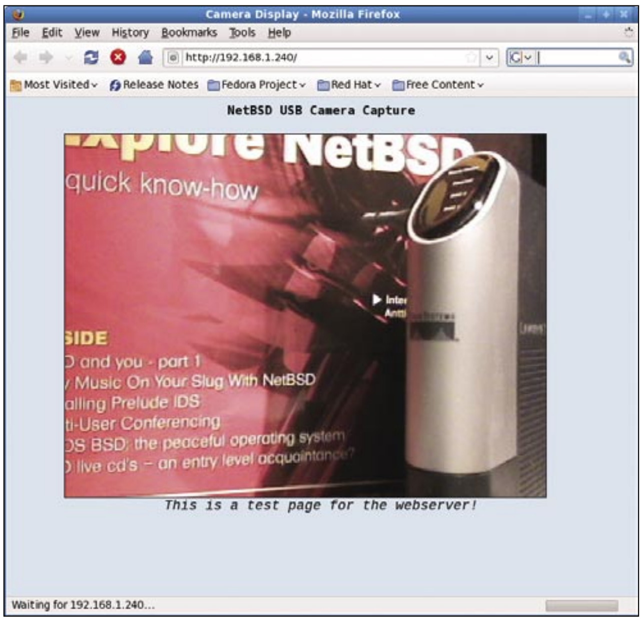
NetBSD has a well-deserved reputation for portability. Part of this reputation comes from the driver layer, which makes use of an abstraction layer known as the Modular Portability Layer (MPL). This layer enables a single driver to be easily used on all architectures by hiding details of exactly how the host talks to the hardware and dramatically reduces the amount of work needed to port it to a new architecture.

In an earlier issue, for example, we added the audio device driver to the Linksys NSLU2 (Slug) ARM-based kernel in order to play music on a Slug (Play Music On Your Slug With NetBSD, Donald T. Hayford, BSD Magazine, Vol. 2, No. 1, 1/2008). I've also successfully added the NetBSD Bluetooth drivers to a NSLU2 kernel. In this article, we'll use the Slug and NetBSD to put together a small, embedded system that can serve video to a web browser or capture stop-action images using a UVC-compliant USB video camera.

USB Video

The USB Video Class (UVC) specification was developed by the USB Implementers Forum (*http://www.usb.org/*) and is available at their website (See the Video Class 1.1 document set available at *http://www.usb.org/developers/devclass_docs/USB_Video_Class_1_1.zip*). Initially released in 2003, the lastest version, 1.1, came out in June, 2005, and a UVC-compatible driver has been in OpenBSD since April, 2008. As part of the 2008 Google Summer of Code (See *http://code.google.com/soc/2008/*) project, Pat Mahoney, under the guidance of Jared McNeill, developed a NetBSD driver (See *http://netbsd-soc.sourceforge.net/projects/uvc/* for more information) that is UVC-compatible.

Not surprisingly, UVC support is available in Linux and has been built in since the 2.6.26 kernel. Video programming support has been available in Linux for a number of years in the form of a library known as video4Linux (v4l). That library has been updated and the standard video interface is now known as video4Linux2 (v4l2) (See *http://www.linuxtv.org/downloads/video4linux/API/V4L2_API/* for a copy of the API). Nearly all open source software that uses video uses one



NetBSD USB Camera Capture

This is a test page for the webserver!

Waiting for 192.168.1.240...

Figure 1. The Slug sends a picture of its favorite magazine to the browser

of these two APIs. The NetBSD driver is v4l2-compatible, so the NetBSD programming interface is the same as the Linux v4l2 API.

I'm sure that when Pat Mahoney was developing the USB video driver, the furthest thought from his mind was whether that driver would also work with a 266 MHz Arm processor with 32 MB of memory. But it should. So, get out your favorite NetBSD-supported embedded processor and let's give it a try.

What you'll need:

- A processor board with Ethernet and USB interfaces that is supported by NetBSD. I've used the Linksys NSLU2 and the Buffalo Kurobox Pro/Linkstation Pro. You can even use a desktop computer if you want, though what fun you'll find in that I couldn't say. The Linksys NSLU2 is used as the example in this article since it is better supported by NetBSD.
- A USB camera that is compatible with the UVC (USB Video Class) standard. For this article, I used a Logitech QuickCam Deluxe for Notebooks. If the camera documentation says *Certified for Windows Vista*, then it is UVC-compatible and will work with the NetBSD UVC driver. Just another of the many nice things that the Redmond crowd has done for the *nix world. (Note: Vista-compatible and Certified for Windows Vista are not the same thing.)
- A desktop computer with Linux or a version of BSD to use for building NetBSD, and to serve files to your embedded system.

What you'll end up with:

- A program that can read images from the camera and store them on the embedded computer's disk.
- A simple webserver that can serve images from the camera to a browser.
- A program that can collect stop-action video from the camera.

Get and Build the System Software

The process of building and installing a NetBSD kernel on the NSLU2 has

Listing 1. Steps to acquire and build NetBSD for the NSLU2

```
mkdir netbsd-20081215
export CVS_RSH="ssh"
export CVSROOT="anoncvs@anoncvs.NetBSD.org:/cvsroot"
cvs checkout -D 20081215-UTC src
mkdir npe
cd npe
(go to Intel website: http://www.intel.com/design/network/products/
npfamily/ixp400_current.htm, download the file IPL_ixp400NpeLibrary-2_3_
2.zip to this directory)
unzip IPL_ixp400NpeLibrary-2_3_2.zip
cd ixp400_xscale_sw/src/npeDl/
echo '#define IX_NPEDL_NPEIMAGE_NPEB_ETH' > IxNpeMicrocode.h
echo '#define IX_NPEDL_NPEIMAGE_NPEC_ETH' >> IxNpeMicrocode.h
cc ixNpeDlImageConverter.c -o foo
./foo
cp IxNpeMicrocode.dat ../../../../src/sys/arch/arm/xscale/
cd ../../../../src/sys/arch/evbarm/conf/
echo 'include "arch/evbarm/conf/NSLU2"'>NSLU2_ALL
echo 'uaudio* at uhub? port ? configuration ?' >> NSLU2_ALL
echo 'audio* at uaudio?' >> NSLU2_ALL
echo 'uvideo* at uhub?' >> NSLU2_ALL
echo 'video* at videobus?' >> NSLU2_ALL
echo 'config netbsd-vid-npe0-20081215 root on npe0 type nfs' >> NSLU2_ALL
echo 'config netbsd-vid-sd0-20081215 root on sd0a type ffs' >> NSLU2_ALL
echo 'config netbsd-vid-sd1-20081215 root on sd1a type ffs' >> NSLU2_ALL
cd ../../../../../src/
./build.sh -O ../obj-armeb -T ../tools-armeb -m evbarm-eb tools
./build.sh -O ../obj-armeb -T ../tools-armeb -D ../distrib-armeb -R ../rel-
armeb -U -u -m evbarm-eb distribution
./build.sh -O ../obj-armeb -T ../tools-armeb -D ../distrib-armeb -R ../rel-
armeb -U -u -m evbarm-eb -V \
    > KERNEL_SETS=NSLU2_ALL release
```

Listing 2. Obtaining and building the necessary packages

```
mkdir ~/pkgsource
cd ~/pkgsource
ftp ftp://ftp.NetBSD.org/pub/pkgsrc/pkgsrc-2008Q3/pkgsrc-2008Q3.tar.gz
su
(enter your root password)
tar -xzf pkgsrc-2008Q3.tar.gz -C /usr
cd /usr/pkgsrc/www/bozohttpd
make install clean
cd ../../net/wget
make install clean
vi /etc/inetd.conf
(change the two lines that start with "#http" to:)
http stream tcp nowait:600_httpd /usr/pkg/libexec/bozohttpd  bozohttpd
/var/www
http stream tcp6 nowait:600_httpd /usr/pkg/libexec/bozohttpd  bozohttpd
/var/www
(save the file)
chmod /var/www 775
exit
(exit superuser mode)
```

been described many times, so I'll just hit the highlights here. If you need more information, see the articles in the NetBSD wiki (See *http://wiki.netbsd.se/How_to_install_NetBSD_on_the_Linksys_NSLU2_(Slug)_without_a_serial_port%2C_using_NFS_and_telnet*) or previous issues of BSD Magazine (NetBSD on the NSLU2, Donald T. Hayford, BSD Magazine, Vol. 1, No. 1, 1/2008). Listing 1 shows all of the steps necessary to build the kernel for the NSLU2. If you want to try out video with a desktop machine, you won't need to change the kernel configuration file since the video driver is already included in the `i386` configuration. If you want to try this for the Kurobox Pro, refer to the 2/2009 issue of BSD Magazine (Building NetBSD for Embedded Systems Using Cygwin, Donald T. Hayford, BSD Magazine, Vol 2, No. 2, 2/2009) for instructions on how to build a kernel for that device. If you want to use some other processor, check it's configuration file to see if the video device driver has already been added. Obviously, you will need to adjust the steps in Listing 1 for the processor type and configuration file names that match your particular processor.

Once you've built the kernel, you'll need to setup the root disk and boot your embedded computer. Instructions for the NSLU2 or Kurobox Pro can be found in the same references as for building the kernel. You'll also want to set up a non-root user that can su to root when necessary (i.e, is a member of the wheel group).

Next, you'll need to get the packages source (For more information on using NetBSD packages, see The pkgsrc guide at *http://www.netbsd.org/docs/pkgsrc/*). While you can cross-build packages, I think it's easier just to do it on the Slug (albeit, a little slower). So, boot up your Slug and follow the steps in Listing 2 to get the package source, install it, and build the web server that we'll use to send video images to browser. There are several web servers that will run on small machines like the Slug; we'll use the Bozotic web server found at `/usr/pkgsrc/www/ bozohttpd`. No, I don't know what Bozotic means, either. Though a goofy name, this tiny web server is surprisingly powerful. Also, build and install wget as shown at the end of Listing 2 to simplify retrieving some files later.

If you look into the available packages that can work with video, you'll find some nice ones, along with several, such as xawtv, that also include a video web server. You can even build xawtv, if you want. But you'll run into problems if you try to run these packages on your embedded processor. The first is that most of these packages expect to find an X windows server, and so won't run (unless, of course, you have an X server running on your embedded processor). Even those that run from the command line will complain about not finding a suitable video driver. What's up, you ask? The problem, as far as I can tell, is that the USB video driver is available in -current (which is the kernel version we built), but not yet in the standard release. Until it is, these packages are looking for a pci-based video card or similar hardware, and don't yet work with the `video4Linux2` interface that the USB driver implements. But don't worry, we'll build our own.

Getting and Building the Video Capture Software

To save you some typing (and typos), the simple programs used in this article to capture video were uploaded by Jared McNeill to the NetBSD ftp server at *http://ftp.netbsd.org/pub/NetBSD/misc/jmcneill/magazine.tar.gz*. Listing 3 shows how to download and build the necessary files. Two of the files, `jpeg_mangle.c` and `.h`, are adapted from files of the same name from Pat Mahoney's source code in the SourceForge CVS repository (The CVS repository can be found at *http://netbsd-soc.cvs.sourceforge.net/netbsd-soc/uvc/*). The other c source files are adaptations of a simple video capture program from Jared.

After building these three programs, run the program called *grabvideo*, as shown in Listing 4. This listing also shows the output that I got. The output lists some of the information available from your camera and is mostly self-explanatory. But notice the line that starts with *pixel format:* and ends with four hex values. If the line you get looks the same, then the rest of the programs provided here will work for you as is; if not, you will have a little more work to do. So what does this all mean?

The system interface to the video driver is captured in the file `src/sys/sys/videoio.h`, a portion of which is shown in Listing 5, while the relevant portions of the code from `videograb.c` are shown in Listing 6 After the video device is opened, the driver is queried through the `ioctl` call for the image format using the structure defined as `v4l2_format` in Listing 5. As of this writing, there are thirty

Listing 3 Acquiring and Building the Video Capture Software

```
mkdir ~/video
cd ~/video
wget http://ftp.netbsd.org/pub/NetBSD/misc/jmcneill/magazine.tar.gz
gunzip magazine.tar.gz
tar -xvf magazine.tar
cd magazine
cp index.html /var/www
gcc grabvideo.c jpeg_mangle.c -o grabvideo
gcc timedavigrab.c jpeg_mangle.c -o timedavigrab
gcc timedvideograb.c jpeg_mangle.c -o timedvideograb
```

Listing 4. Output from the grabvideo program

```
-bash-3.2$ ./grabvideo /dev/video0 testimag.jpg
video format info
height: 240
width: 320
bytes per line: 640
image size: 153600
pixel format: 47504a4d
enum field:1
grabbed 5705 bytes
```

supported formats, ranging from simple RGB one-to-three byte arrays to more complicated YUV arrays or compressed video. The format is specified by a `v4l2_fourcc` macro as shown at the bottom of Listing 5. Only a few of these are shown in Listing 5; refer to the header file and the `v4l2 documentation` for the complete list. My camera returns a format of `0x47504a4d`, which are the four ascii bytes `G`, `P`, `J`, and `M`, or in a mirror, `MJPG`, representing *Motion JPEG*. This is essentially the standard compressed JPEG format except that the table used for the Huffman encoding is fixed and, thus, left out. Motion JPEG is not as good as many of the compression routines that use frame-to-frame predictions like MPEG-2 or -4. However, for surveillance cameras or other applications where there is a signficant time gap between frames, MJPG has the advantage that each frame can be reconstructed without any knowledge of previous frames. If your camera's pixel format is MJPG (and many are), the software as written will work for you. If not, the software will report that it

Listing 5. Portion of the NetBSD videoio.h header file

```
/* $NetBSD: videoio.h,v 1.4 2008/09/25 19:34:49
jmcneill Exp $ */

<snip>
struct v4l2_pix_format {
        uint32_t        width;
        uint32_t        height;
        uint32_t        pixelformat;
        enum v4l2_field field;
        uint32_t        bytesperline;
        uint32_t        sizeimage;
        enum v4l2_colorspace colorspace;
        uint32_t        priv;
};
<snip>
struct v4l2_format {
        enum v4l2_buf_type type;
        union {
                struct v4l2_pix_format pix;
                struct v4l2_window win;
                struct v4l2_vbi_format vbi;
                uint8_t         raw_data[200];
        } fmt;
};
<snip>
/* Pixel formats */
#define V4L2_PIX_FMT_RGB332        v4l2_fourcc('R',
'G', 'B', '1')
#define V4L2_PIX_FMT_RGB555        v4l2_fourcc('R',
'G', 'B', 'O')
#define V4L2_PIX_FMT_RGB565        v4l2_fourcc('R',
'G', 'B', 'P')
#define V4L2_PIX_FMT_RGB555X       v4l2_fourcc('R',
'G', 'B', 'Q')
#define V4L2_PIX_FMT_RGB565X       v4l2_fourcc('R',
'G', 'B', 'R')
<snip>
#define V4L2_PIX_FMT_MJPEG v4l2_fourcc('M', 'J', 'P',
'G')
#define V4L2_PIX_FMT_JPEG  v4l2_fourcc('J', 'P', 'E',
'G')
#define V4L2_PIX_FMT_DV            v4l2_fourcc('d',
'v', 's', 'd')
#define V4L2_PIX_FMT_MPEG  v4l2_fourcc('M', 'P', 'E',
'G')
<snip>
```

Listing 6. Code Fragment from videograb.c

```
<snip>
int
main(int argc, char *argv[])
{
        struct v4l2_format fmt;
        uint8_t *buf;
        int ifd, ofd;
        int error;
        size_t frlen;
        ssize_t rdlen, wrlen;
        size_t huff_offset;

        if (argc != 3)
                usage();
                /* NOTREACHED */

        ifd = open(argv[1], O_RDONLY);
        if (ifd < 0) {
                perror("open camera failed");
                return EXIT_FAILURE;
        }
        ofd = open(argv[2], O_WRONLY|O_CREAT|O_EXCL,
0644);
        if (ofd < 0) {
                perror("open output failed");
                return EXIT_FAILURE;
        }

        error = ioctl(ifd, VIDIOC_G_FMT, &fmt);
        if (error) {
                perror("VIDIOC_G_FMT failed");
                return EXIT_FAILURE;
        }
        printf("video format info\n\theight: %d\n",
fmt.fmt.pix.height);
        printf("\twidth: %d\n",fmt.fmt.pix.width);
        printf("\tbytes per line: %d\n", fmt.fmt.pix.by
tesperline);
        printf("\timage size: %d\n",fmt.fmt.pix.sizeima
ge);
        printf("\tpixel format: %04x\n", fmt.fmt.pix.pi
xelformat);
        printf("\tenum field:%d\n", fmt.fmt.pix.field);
<snip>
```

couldn't figure out where the Huffman table belonged and leave without writing anything. In that case, you'll need to look up your image format and change the supplied software to output that image format. Look around a bit with Google and I'm sure you'll find what you need without too much effort.

Though Motion JPEG is essentially JPEG, most software that will work with JPEG files won't recognize the data if you simply write out the image received from the camera since they expect to find the Huffman table as part of the image data. The routines I borrowed from Pat Mahoney in `jpeg_mangler.c/h` figure out where the Huffman table should go and pass back a pointer to the default UVC-specified table, allowing us to write out a standard JPEG image that can be read by other image processing or display programs. Since all standard browsers display JPEG images, we can use that capability to provide a dynamic display of the video image.

The very simple web page shown in Listing 7, `index.html`, sends two text lines and an image to the browser for display. In Listing 3, this file was put into the proper directory (`/var/www`) after we downloaded and untarred it. The javascript embedded in the web page then waits for two seconds and requests a page reload from the web server. The web browser sends whatever image is stored in the file `/var/www/test.jpg`. To make a dynamic display, then, we'll simply write the captured video image to a drive periodically, and the web browser will automatically send the latest image every time the browser requests a reload. I used a soft link to connect the image saved by the software to the image file that the web server is looking for, as illustrated in Listing 8. Figure 1 shows a captured image in a browser window.

On the other hand, Motion JPEG is very similar to the QuickTime or AVI video formats. By periodically capturing frames and saving these directly to a file, you can generate a stop-action video using the virtually the same software. The only difference is that, instead of rewriting a single image multiple times, you concatenate the same images. The final piece of software, *timedavigrab*, does just that, using command line parameters to determine the time interval between frames and the number of total frames.

Listing 7 Simple javascript file that repetively fetches and displays a video image.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1">
         <title>Camera Display</title>
         <style type="text/css" id="mtmsheet"></style>
  </head>
  <body bgcolor = "#cedfea">
    <h4 align="center">NetBSD USB Camera Capture</h4>
    <tr>
      <IMG src="test.jpg" width=640 height=480 hspace=70 vspace=0 border=1 alt=" ">
      <div align="center">
        <font face="Courier New,Courier,mono" size="4"><I>This is a test page for the webserver!</I></font><br>
      </div>
    </tr>
    <script language="javascript">
       setTimeout("window.location.reload(true)",2000);</script>
  </body>
</html>
```

Listing 8. Using the timedvideograb program to generate continuously updated still images for the `web server`

```
-base-3.2$ ln -s localtest.jpg /var/www/test.jpg
-bash-3.2$ ./timedvideograb /dev/video0 localtest.jpg 2
video format info
        height: 240
        width: 320
        bytes per line: 640
        image size: 153600
        pixel format: 47504a4d
        enum field:1
sleep time: 2
```

Conclusion

The new NetBSD UVC video driver appears to work flawlessly on embedded computers as well as on the larger and more capable desktop machines. The package system needs to catch up to the use of the v4l2 API, but I suspect that will happen as NetBSD moves closer to releasing version 5. Operated with a small embedded device like the NSLU2, a USB video camera can be used as a surveillance camera or for capturing stop-action video of slowly changing objects. And since the NSLU2 is a relatively low-power device, setting up a battery-powered remote system should be straightforward.

You'll note that the kernel we built also incorporated the USB audio device as well as the video device. That's because the Logitech camera I used has both audio and video capability. And though we didn't use it here, you can use the `/dev/audio` device to capture audio as well. If we only had a video output device, we could think about a small, self-contained video conferencing system...hmm.

Acknowledgments

Thanks to Pat Mahoney for developing the USB video driver, and to Jared McNeill for sponsoring Pat and answering my questions.

About the Author

Don Hayford is a Research Leader at Battelle Memorial Institute, where he specializes in the development of data acquisition systems for customers. Don has been involved with microprocessors from the time they doubled in size from four bits to eight, and once knew how to boot up a PDP-11 using the front panel switches. In his career, he has written software for the CP/M, RT-11, MS-DOS, Apple DOS, Windows, Linux, and BSD operating systems using assembler, Basic, C, C++, C#, and Fortran. Married with three children, Don and his wife like to spend their free time cooking and travelling.

FreeBSD Tips

Dru Lavigne

Whether you're new to FreeBSD or have been using it for some time, learning a new trick or two can save you time and increase your user experience. In this Tips & Tricks, we'll show you how to save time at the command line, create a trash directory in your shell, build FreeBSD ports without installing the ports tree, control SSH connections, visualize rc.conf, and create an easy-to-use environment for controlling your FreeBSD system.

When You're Stuck at the Shell

The default shell for the FreeBSD superuser account is `tcsh`. If you have a preference for `bash`, you can always `pkg_add -r bash`, but sometimes you are in an environment where you can't install additional software. No worries, the `tcsh` shell supports many nicities such as autocomplete (by pressing tab) and history (use `h` or your up arrow to review history and !number to select a numbered command).

Note: You don't have to be the superuser to use the `tcsh` shell. If you're unsure what shell you are currently using, ask your shell:

```
echo $0
/bin/tcsh
```

If you get back a different shellname, type tcsh to enter the `tcsh` shell. Here I'll change from the Bourne shell (`sh`) to `tcsh`:

```
echo $0
sh

tcsh
echo $0
/bin/tcsh
```

`tcsh` provides dozens of built-in *hotkeys* and allows you to create your own key mappings for commonly used commands. You can view the current key mappings with:

```
bindkey | more
Standard key bindings
"^@"           ->  set-mark-command
"^A"            ->  beginning-of-line
"^B"            ->  backward-char
"^C"            ->  tty-sigintr
```

The `^` means hold down the control key while you press the character that follows. Note that the *standard key bindings* are case insensitive, meaning `^a` is equivalent to `^A`; however, the "*multi-character bindings* are case sensitive. If you're unsure what a binding does after reading its description, type some text at the command line and see what happens when you try the key binding.

You can create your own key bindings, which can be very useful for commonly repeated actions. While you can overwrite any current key binding, you may prefer to search for an undefined binding:

```
bindkey | grep undefined
"^G"            ->  is undefined
"\300"          ->  is undefined
"\305"          ->  is undefined
```

You can then bind whatever command (string) you wish as long as the

Listing 1. Minimal Ports Tree Installed by porteasy

```
ls /usr/ports/
.cvsignore              INDEX-7           Makefile         UIDs
www/
CHANGES           INDEX-7.bz2       Mk/               UPDATING
COPYRIGHT         KNOBS             README        converters/
CVS/                    LEGAL             Templates/       devel/
GIDs                    MOVED             Tools/            misc/

cd /usr/ports/www
ls
CVS/            Makefile        lynx-current/
cd lynx-current
make install clean
```

Listing 2. Configuring SSH for ls Only

```
ssh localhost
Enter passphrase for key '/home/dru    /.ssh/id_rsa': mypassphrasehere
PTY allocation request failed on channel 0
Desktop
Documents
Images
Music
Videos
file1
file2
Connection to localhost closed.
```

command is enclosed in quotes. Be sure to test your binding after creating it:

```
bindkey -s "^G" "csup -L2 /root/cvs-
supfile"
```

In this example, I've bound *control g* to the command I use to check for system updates. Now, whenever I press *control g* I'll see:

```
csup -L2 /root/cvs-supfile
```

If I don't want to have to press enter after I see the command to start it, I should change the binding to create the newline (press enter) for me:

```
bindkey -s "^G" "csup -L2 /root/cvs-
supfile\n"
```

Now when I press *control g*:

```
csup -L2 /root/cvs-supfile
Parsing supfile "/root/cvs-supfile"
Connecting to freebsd.nycbug.org
Connected to 66.111.2.68....
```

Command Line Trash Directory

If you're used to working in a GUI environment, you may find the ability to periodically restore files from a trash bin quite useful. If so, you've probably noticed that at the command line, once a file it is deleted, it is gone forever. If you've ever been bitten by this, consider creating a simple script. First, `cd` into your home directory and create two directories: one to hold your script (*bin*) and the other to act as a hidden trash (*.trash*) directory to store deleted files:

```
cd
mkdir bin .trash
```

Next, create a script called trash and save it in your newly created bin directory. This command shows the contents of that file:

```
more ~/bin/trash
#!/bin/sh
#script to send deleted files to hidden
trash directory
mv $1 ~/.trash/
```

Don't forget to make the script executable:

```
chmod +x  ~/bin/trash
```

Next, create an alias to replace the `rm` command with the trash script by adding this line to `~/.cshrc`:

```
alias  rm        trash
```

Finally, test that it works:

```
source  ~/.cshrc
echo "some garbage text" > testfile
rm testfile
ls  ~/.trash
testfile
```

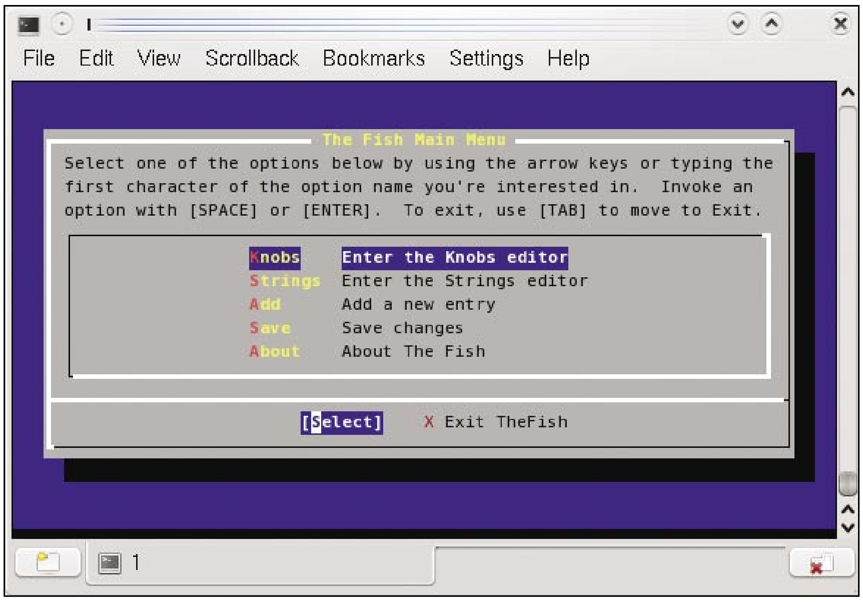
If you ever really do want to delete a file without sending it to the trash directory, you can override your alias like this:

```
\rm filename
```

Keep in mind that your trash directory will only work while you're in the shell and it won't be available to you if the configuration file for your shell does not contain the alias. You may wish to create a trash directory and `rm` alias for both your regular user account and the superuser account.

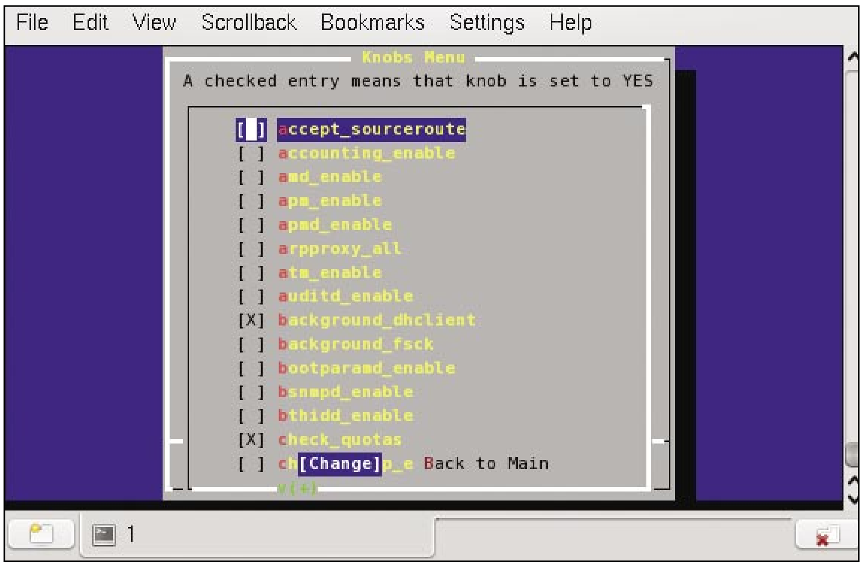
Building a Port Without Installing the Ports Tree

If you have a slow Internet connection or limited disk space, it can be a pain to download and maintain the entire ports tree. The current tarball of the ports tree is over 42MB, and once unzipped it can take up a few hundred MBs of disk space. If you only build a few ports, it makes sense to just download the part



```
The Fish Main Menu
Select one of the options below by using the arrow keys or typing the
first character of the option name you're interested in.  Invoke an
option with [SPACE] or [ENTER].  To exit, use [TAB] to move to Exit.

          Knobs     Enter the Knobs editor
          Strings   Enter the Strings editor
          Add       Add a new entry
          Save      Save changes
          About     About The Fish

              [Select]     X Exit TheFish
```

Figure 1. Main Screen of thefish



```
Knobs Menu
A checked entry means that knob is set to YES

  [ ] accept_sourceroute
  [ ] accounting_enable
  [ ] amd_enable
  [ ] apm_enable
  [ ] apmd_enable
  [ ] arpproxy_all
  [ ] atm_enable
  [ ] auditd_enable
  [X] background_dhclient
  [ ] background_fsck
  [ ] bootparamd_enable
  [ ] bsnmpd_enable
  [ ] bthidd_enable
  [X] check_quotas
  [ ] ch[Change]p_e Back to Main
      v(+)
```

Figure 2. Knobs Editor in thefish

of the ports tree that you need to build the ports you need. This can be easily down with the `porteasy` utility.

To use this utility, become the superuser and install the porteasy package and create an empty ports directory:

```
pkg_add -r porteasy
rehash
mkdir /usr/ports
```

porteasy uses anonymous cvs, so you need to prepare your environment first:

```
touch ~/.cvspass
setenv CVSROOT :pserver:anoncvs@anonc
vs.tw.FreeBSD.org/home/ncvs
```

Now, whenever you want to download (update) the ports skeleton for an application, specify the name of the port you wish to build. In this example, I want to download the skeleton for the `lynx` port:

```
porteasy -u lynx
```

The first time you run `porteasy`, it will download the ports INDEX as well as all the tools, templates, and Mk files that the ports system needs. If your output ends in a message similar to this:

Can't find required port 'lynx', maybe you mean:

```
  lynx-2.8.6.5_5,1
  lynx-2.8.7d13
```

some required ports were not found.

It means that there are multiple versions of the application you requested and that you need to specify which version you want:

```
porteasy -u lynx-2.8.7d13
```

You can get the version information ahead of time by asking for the list:

```
porteasy -l lynx
```

Once the update command succsessfully finishes, you can cd into the ports directory of the application and build the port as usual: see Listing 1.

Controlling SSH

FreeBSD comes with an SSH server which has been pre-configured with some security options. For example, by default, SSH logins by the superuser account are refused. You can further tighten up who is allowed to SSH to your system by modifying the SSH server configuration file, `/etc/ssh/sshd_config`.

For example, to only allow logins from the user "dru", add this line to the bottom of the file:

```
AllowUsers dru
```

Note that the `AllowUsers` keyword is case-sensitive, meaning it won't work if the `A` and the `U` are lowercase. You can add multiple user accounts by placing a space between each user. Don't forget to tell your SSH server that you have made changes to this file:

```
/etc/rc.d/sshd restart
```

You should also test that your changes worked by trying to login as the specified user (which should work) and as several un-specified usernames (which should fail). man `sshd_config` contains many more keywords which can be used to control behaviour such as which IPs are allowed to connect and how users can authenticate once they connect.

If your users are using public key authentication, you can configure your SSH server to allow them to connect in order to run a command. For example, the superuser can configure a user to connect in order to see a listing of the files in their home directory, but to not receive a network terminal (pty) where they can run additional commands. This configuration requires you to `su` to that username and modify the authorized keys file in that user's home directory on the SSH server by inserting this text at the very beginning of `~username/.ssh/authorized_keys`, right before the `ssh-rsa` or `ssh-dsa` keyword:

```
command="ls",no-pty
```

Be sure to have that user test that ssh works as expected: see Listing 2.

If the user instead receives a shell, doublecheck that your inserted text is not on its own line and is just before the key itself.

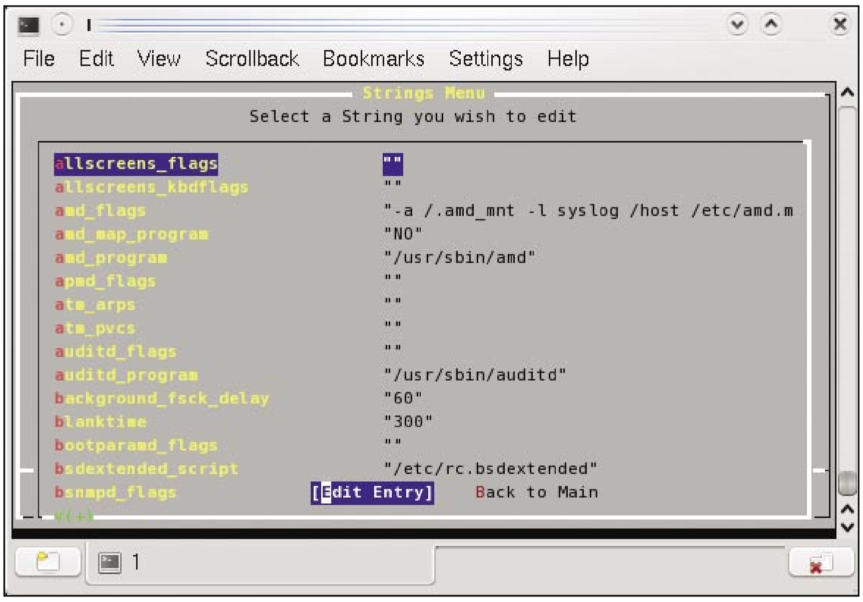
The *Authorized_Keys File Format* section of `man sshd` contains many more ideas for controlling how users connect to your SSH server and what they can do once they get there.

Visual rc Settings

One of the beauties of FreeBSD is that one text file, `/etc/rc.conf`, controls which services start at system boot. This file is easy to edit and `man rc.conf` does a great job of letting you know what services and options are available for this file.

But, sometimes it is nice to have a more visual representation of the possible services to run at system startup. A utility known as `thefish` provides an easy-to-use menu-based program for controlling services. It is easy to install and use (Figure 1):

```
pkg_add -r thefish
rehash
thefish
```



```
                    Strings Menu
           Select a String you wish to edit

  allscreens_flags            ""
  allscreens_kbdflags         ""
  amd_flags                   "-a /.amd_mnt -l syslog /host /etc/amd.m
  amd_map_program             "NO"
  amd_program                 "/usr/sbin/amd"
  apmd_flags                  ""
  atm_arps                    ""
  atm_pvcs                    ""
  auditd_flags                ""
  auditd_program              "/usr/sbin/auditd"
  background_fsck_delay       "60"
  blanktime                   "300"
  bootparamd_flags            ""
  bsdextended_script          "/etc/rc.bsdextended"
  bsnmpd_flags         [Edit Entry]     Back to Main
  v(+)
```

Figure 3. Strings Editor in thefish

By pressing enter with the *Knobs* entry highlighted, you can see which services are available and which will automatically start at boot time: see Figure 2.

You can also insert options (as described in man `rc.conf`) using the Strings editor: see Figure 3.

Note: If you run thefish from a GUI, you may instead see the GTK/QT version which offers the same functionality with a slightly different look and which allows you to select options with your mouse.

Controlling your System with Webmin

I've been a big fan of Webmin (*http://webmin.com/*) for years and continue to be amazed as every version adds yet more functionality and improvements. While designed for remote administration of server systems, it is so handy that even novice users should consider using it to manage their own desktops.

Note: The system you wish to control should have `webmin` installed. You can then access that system from any system containing a web browser.

You can install and configure webmin as follows:

```
pkg_add -r webmin
/usr/local/lib/webmin/setup.sh
```

During the setup.sh script, you can press enter to accept the default path locations. It is a good idea to enter a different port number, login name, password, and to choose y for SSL when prompted. Once the script is finished, open a web browser and type localhost:portnumber, where portnumber is the port number you chose during setup.sh.

Note: Webmin uses a self-signed certificate for SSL connections. If your browser complains, follow its instructions to add an exception to accept the certificate. Also, if for some reason `webmin` did not start, you can start it with `/usr/local/etc/rc.d/webmin` onestart.

Once you've logged in, you'll quickly find that webmin allows you to control most aspects of your system: see Figure 4.

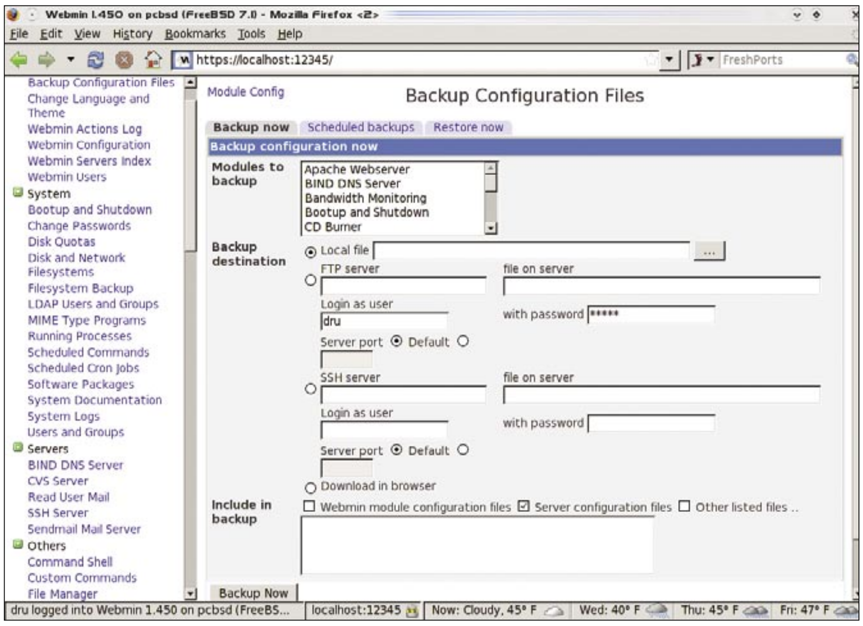
You'll definitely want to first spend some time in *Webmin Configuration* under *Webmin*. Here you can control which IP addresses and users are allowed to connect to your webmin server and the type of authentication to use when connecting to webmin. You can also install additional modules and upgrade your version of webmin.

The System section allows you to easily:

- control which services start at bootup
- change user passwords
- manage disk quotas
- mount filesystems
- backup and restore directories
- install, configure, and use LDAP
- view and control running processes
- schedule commands and cron jobs
- manage packages
- read and search manpages
- manage and read logs
- manage users and groups

If you're running any services on your system, you'll quickly become addicted to the Servers section and the other sections that follow. Here you can control services such as:

- BIND DNS
- CVS
- Sendmail, Qmail and Postfix
- SSH
- Apache
- IPFW and IPFilter
- NFS
- Printers
- Bacula
- MySQL and PostgreSQL
- Samba
- SpamAssassin
- FTP
- Squid



Backup Configuration Files

Backup configuration now

Modules to backup: Apache Webserver, BIND DNS Server, Bandwidth Monitoring, Bootup and Shutdown, CD Burner

Backup destination: [✓ Local file] ...
FTP server / file on server
Login as user: dru / with password: *****
Server port [✓ Default]
SSH server / file on server
Login as user / with password
Server port [✓ Default]
Download in browser

Include in backup: ☐ Webmin module configuration files ☑ Server configuration files ☐ Other listed files ..

dru logged into Webmin 1.450 on pcbsd (FreeBS... | localhost:12345 | Now: Cloudy, 45° F | Wed: 40° F | Thu: 45° F | Fri: 47° F

Figure 4. Webmin

Summary

I hope that you enjoyed these Tips & Tricks and have found something to try on your FreeBSD system. You can find many more tips and tricks for BSD systems in the books BSD Hacks, published through O'Reilly, and The Best of FreeBSD Basics, published by ReedMedia.

About the Author

Dru Lavigne is a network and systems administrator, IT instructor, author and international speaker. She has over a decade of experience administering and teaching Netware, Microsoft, Cisco, Checkpoint, SFreeBSD BasicsCO, Solaris, Linux and BSD systems. A prolific author, she pens the popular FreeBSD Basics column for O'Reilly and is author of BSD Hacks and The Best of FreeBSD Basics.

She is currently the Editor-in-Chief of the Open Source Business Resource, a free monthly publication covering open source and tBSD Certification Group Inc.BSD Certification Group Inc.he commercialization of open source assets. She is founder and current Chair of the BSD Certification Group Inc., a non-profit organization with a mission to create the standard for certifying BSD system administrators.

Maintaining System Configuration Files Using Subversion

Mikel King

Recently I was asked about maintaining a data center full of servers. More specifically about maintaining a repository of the configuration files for all servers in the data center. As our data centers and systems in general become more sophisticated managing the complex array of all the configuration data in and of itself is nearly as important as the user data stored in.

Anyone who's ever lost a server as a result of some catastrophic failure, be it failed equipment or some other nefarious means, knows it is not easy to rebuild a system to its pre-failure state. Let's face it even the best backups can yield less than accurate results should the archive media becomes faulty. As a layer of redundancy I like the idea of a configuration management solution. Of course disaster preparedness is not the only reason one might consider implementing a some sort of configuration management solution.

If you have a large installation of equipment it becomes increasingly difficult to keep track of the numerous system updates and configuration files. Especially if you are in an environment with inconsistent technical staff as result high employee turn over for instance.

Several years ago during a large coding project I was introduced to subversion, and although I had been familiar wit other versioning solutions for whatever reason svn stuck. Initially we started with just the code base, however the more we used it the more we put into the repository. I ended up dropping all of the documentation, apache, php and mysql configuration files into it.

Shortly after completing the beta testing we had to replicate the entire server installation into numerous front end production web servers. It was then that it hit me that if we had the svn client on each server all we would have to do is run a checkout to have 90% of the configuration completed.

This certainly helped expedite server rollouts. Of course it did add an additional step in the pre-deployment built out. In addition to installing php, apache, and mysql we would now have to install svn. Although this is not a huge task it does add a layer of complexity to the overall schema. One could use a svn repository to manage the build options for your system to ensure that you are creating nearly identical deployments.

As you can see this can snowball rather quickly and it's a delicate balancing act determining where to draw the line. I my data center I have opted for maintaining a repository of `/etc` and `/usr/local/etc` of each system for each server.

To keep things simple I shall assume that you have a working repository server. While there are several different ways to organize this repository, I have found that the best is to start with a group of like servers based on function. For instance let's start with the named servers. Of course I am assuming that each server only performs a single function. If you are a jails jockey then this is likely to be the case, however if you are constrained by space, power and hardware it is more likely that each machine fills at least two billets.

Still for the sake of simplicity let's roll with the assumption that you only have one service per server. In addition we shall limit out discussion to a single division, as some organizations will have multiple divisions as well as being dispersed across multiple locations. Again for the sake so simplicity let's assume that you only have the one.

Very well with the basic assumptions in place we need to construct our server repository. After confirming that I am able to access the svn server I begin with adding the server to the 'Servers' repository. From the prompt on the server named thoth, I would run the following command;

```
    thoth: svn mkdir svn://
svn.olivent.com/Servers/THOTH
```

Notice that I placed the server name in all capital letters. This is a habit I picked up from customizing kernels in FreeBSD where one would copy GENERIC to the host name in all caps. You are certainly free to setup your system as suits your personal style best.

The next step is to start importing etc and `/usr/local/etc` into the system. The easiest way to accomplish this is to execute the following in root;

```
    thoth: svn mkdir svn://
svn.olivent.com/Servers/THOTH/etc
    thoth: cd /etc
    thoth: sudo svn import svn://
svn.olivent.com/Servers/THOTH/etc
```

Although the import command should recursively create the target for you at the destination I have found it is better to explicitly create it yourself. The import command assumes that your current working directory is the one you wish to import. If the command is successful then you will see numerous files listed ending

Listing 1. Single user mode rebooting

```
    thoth: mkdir /tce
    thoth: cd /tce
    thoth: svn checkout svn://
svn.olivent.com/Servers/THOTH/etc

*****Reboot to single user
mode*****

    thoth: cd /
    thoth: mv etc old-etc
    thoth: mv tce/etc etc
```

with `Committed revision XX`. Where XX is the actually of the revision number.

Using the same methodology let's add /usr/local/etc into the repository.

```
   Thoth: svn mkdir svn://
svn.olivent.com/Servers/THOTH/usr
   thoth: svn mkdir svn://
svn.olivent.com/Servers/THOTH/usr/local
   thoth: svn mkdir svn://
svn.olivent.com/Servers/THOTH/usr/
local/etc
   thoth: cd /usr/local/etc
   thoth: svn import svn://
svn.olivent.com/Servers/THOTH/usr/
local/etc
```

Observer that once again I explicitly created and specified the destination. Because import will assume the you wish to import everything in the present working directory I change the path to `/usr/ local/etc` to ensure that I do not collect and collateral files. you can imagine what would happen if I imported all of usr. Ok so now we have all of our current configuration files imported into the repository, but that really only helps us half way. One of the main advantages of using a versioning system like subversion is to improve the ability to capture changes to system configuration files as well as document why those changes are being made. Therefore in order to make use of this we need to checkout and place into service our versioned copies of these files. This actually can get a bit tricky

```
   thoth: cd /usr/local/
   thoth: mv etc old-etc
   thoth: svn checkout svn://
svn.olivent.com/Servers/THOTH/usr/
local/etc
```

At this point I have accomplished storing both /etc and /usr/local/etc in the repository for the machine known as THOTH. In addition I have successfully checked out the current repository version of `/usr/local/` etc. Depending on your system and it's activity you may prefer to perform the checkout to a temp folder and drop down to single user mode. If it's a new system you can probably expedite things by not. Also keep in mind that on some systems namely Mac OS X /etc is a symbolic link to `/private/etc` which can make things rather touchy if you do not proceed with caution. Be certain to take the time to make note of your systems' peculiarities.

Continuing with the original assumption that we are experimenting on a FreeBSD based execute the two command blocks outline below. Considering that your system should currently be in multi user mode you should be able to safely checkout the repository to a temp location. I'm using tce which of course is just etc backwards. Next reboot to single user mode, remembering to `mount -w /` before you do or you'll spin your wheels for nothing then execute the later command block (see Listing 1).

If all went as planned then you are now running on your versioned system all that remains to do is boot back up to multi user mode. Once safely back into multi user mode let's try a few things. Suppose that you assign one of your BSDAs to install a new port that requires modifications to your rc.conf as well as its new configuration directory in /usr/local/etc and a new startup script in `/usr/local/etc/` rc.d.

Your Jr sysadmin successfully builds the port and installs the new application and even performs the the appropriate check-ins to the repository complete with commentary documentation as follows;

```
   thoth: cd /etc
   thoth: svn commit
```

The above should only transmit rc.conf if you added the `new_app_enable="YES"` statement as required. Next you will want to add the new configuration to you /usr/ local/etc section of the repository.

```
   thoth: cd /use/local/etc
   thoth: svn add new_app
   thoth: svn add rc.d/new_app.sh
   thoth: svn commit
```

Alright I know that this seems like a lot more work but consider what happens a few weeks later when your Jr sysadmin reboots the sever for some other maintenance and it hangs, dropping to single user mode. Of course it does not take a versioning system to locate the missing quote on `named_enable="YES"` but it's nice to be able to review the logs and determine who was the last person to modify the rc.conf and why.

Obviously there I have demonstrated a rather time consuming manual process for all of this and it is quite possible to script much of the check-in and update process once you are up and running. Additionally after reading this brief introduction to versioning you may be wondering why? Why oh why would I even submit myself to all that effect and action tracking. I do have a good answer for you, concise documentation.

Consider that the server you just added to versioning is not really touched by you for several years. Your Jr sysadmin faithfully maintains the system checking in all of his changes over the years and one day, he leaves the company for a change of career. Now what do you do? How do you know all of the systems that this person maintained? You could start logging in and cataloging this manually, but perhaps is you have a reliable versioning solution in place you could simply run a report on his activity over the last few months.

Another fine example is you have to perform a site audit of all you systems. Perhaps you've wanted to build a network topology diagram for years but of course you just haven't had the resources necessary to catalog hundreds of servers. Suddenly the university you work for has received a small grant to introduce some GREEN initiatives and you sell them on the idea that server consolidation could potentially reduce their power consumption by a sizable amount if only you had the resources namely personnel to complete the task in a timely fashion.

Utilizing your new team of student helpers you task them with the job of cataloging all of the servers. However do you really want to grant them direct access to everything? Perhaps if one were to use a subversion configuration file management system they could grant temporary read only access to the repository. Ultimately allowing this temporary support staff to complete the task in a safe environment.

About the Author

Mikel King has been working in the Information Services field for over 20 years. He is currently the CEO of Olivent Technologies, a professional creative services partnership in NY. Additionally he is currently serving as the Secretary of the BSD Certification group as well as a Senior Editor for Daemon News. Finally he is an active JAFDIP blogger. Drop by mikelking.com and say 'Hi!'

Q&A about DTrace

Could you introduce yourself?

John Birrell: I am an electrical engineer by training, but a software developer in practice. I've been contributing to FreeBSD as a developer for over 10 years. I now work for Juniper Networks in the JUNOS-Core group in Sunnyvale, CA.

George Neville-Neil: I work on networking and operating system code for fun and profit. I also teach various course on subjects related to computer programming. My professional areas of interest include code spelunking, operating systems, networking and security.

I am the co-author with Marshall Kirk McKusick of ”The Design and Implementaion of the FreeBSD operating system” and I am the columnist behind ACM Queue's ”Kode Vicious”.

What is DTrace?

John Birrell: DTrace is a dynamic tracing system developed by Sun Microsystems for their Solaris operating system. The DTrace code was the first part of Solaris to be open-sourced under Sun's *Common Development and Distribution License* (CDDL).

The beauty of DTrace is that it really is dynamic. You can install probes on the fly, look at the output for a while and then remove the probes without restarting any program.

What is your role in porting DTrace to FreeBSD?

John Birrell: Like all things in FreeBSD, the DTrace port happened because I got intrigued after attending a Sun Microsystems Developer Days conference which they hold frequently around the world.

After the conference I was so keen to try DTrace that I tried to install Solaris on my latest PC, but it didn't recognise the hardware so I got nowhere.

Instead I decided to port the code! So, armed with an 86 MB download of OpenSolaris source I set out to find out how Dtrace was coded in Solaris.

I got some help from Sun's Bryan Cantrill who was generous and gave me access to the DTrace test suite before Sun had officially open sourced it.

Is the porting process totally complete? Is there anything that our readers might help you with?

George Neville-Neil: There are still providers to be worked on, such as the PID provider, which is probably the largest remaining piece to add.

Which platforms are supported at the moment?

George Neville-Neil: Intel/AMD x86 32 and 64 bit definitely work. I use DTrace on those every day.

The recent release 7.1 includes support for using DTrace inside the kernel. How can we take advantage of it? Do you expect to use DTrace to profile FreeBSD's kernel for example?

John Birrell: I use DTrace daily. I am working on a build system that uses DTrace to work out the dependencies.

DTrace is a great tool for profiling kernel operation because you don't have to build anything in permanently. The concept of adding printf in kernel code is gone now. To make use of DTrace, you really need to have access to the source code.

That is why DTrace actually makes more sense in FreeBSD than it does in Solaris. Our code is `_always_` available.

Using DTrace is an iterative process. Think of a question and try to probe to test out your theory. Then when you see some results, revise your question to enable different probes.

Where can we find practical examples of how DTrace work?

George Neville-Neil: The best resource is the the Sun manual. The first chapter has examples that work with the FreeBSD version of DTrace.

http://docs.sun.com/app/docs/doc/817-6223

Can we use DTrace to check how our code is exploiting concurrency?

George Neville-Neil: There are ways to get DTrace to show you what CPU is being used by a piece of code but this is the kind of thing I'd think you'd use other subsystems for, such as hwpmc(4).

Can DTrace help sysadmins do their job, or it is a tool more focused on programmers?

John Birrell: There are things that sysadmins can certainly benefit from. As an example, imagine you have a suspect user. You might want to probe what applications the user is running. Or you might want to trace what sockets he/she creates to make outgoing connections.

How many times as a sysadmin have you asked youself the question: *What on earth is this system doing?*

Having DTrace on a system makes it easier to support from outside. Even though a sysadmin may not understand all the kernel code, the fact that DTrace is there allows an external support person to provide the sysadmin a script that can be run by the sysadmin. Before passing on the result for analysis, the sysadmin can check the log to ensure that there is no private data there.

The trick with DTrace is to enable probes which give you a brief summary of exactly what you want to know about instead of dumping everything and forcing you to parse it later.

BSD
CERTIFICATION.ORG

NEW CERTIFICATION EXAM

The BSD Certification Group proudly announces the availability of the BSD Associate Exam (BSDA), the entry level exam for BSD System Administrators.

The BSD Associate Exam is a written proctored certification exam in English only. The BSDCG has worked hard to make this psychometrically valid exam affordable worldwide. See the list of selected conferences and register for an exam seat for $75 USD at **www.bsdcertification.org**.

Get Involved.
Get Certified.
Get Ahead.

www.iXsystems.com
iXsystems is a proud sponsor of BSD Certification Group Inc.

NetBSD FreeBSD OpenBSD DragonFlyBSD

iX-Neutron: A STAR AMONG SERVERS

In striving to bring our customers faster, more reliable servers, iXsystems, Inc. introduces the new iX-Neutron server line. The iX-Neutron server series brings Intel's® newest chip technologies to your business to provide an astronomically fast family of machines. The Intel® 5500 Tylersburg Xeon® processor utilizes these technologies to greatly increase speed, performance, and memory capacity, while saving energy simultaneously. The processor performance scales dynamically based on the requests and demands of the system. Visit us at http://www.iXsystems.com/neutron for more information and pricing.

iX-N1204

- 1U Form Factor with 4 Hot Swap SAS/SATA 3.5" Drive Bays
- Dual Intel® 64-Bit Socket 1366 Quad-Core or Dual-Core, 5500 (Tylersburg/Nehalem) Series Processors
- Intel® 5520 Chipset with QuickPath Interconnect (QPI)
- Up to 96GB DDR3 1333/1066/800 SDRAM ECC Registered Memory (12 DIMM Slots)
- 2 PCI-E 2.0 x8 or 1 PCI-E x16 Expansion Slots
- Intel® 82576 Dual Port Gigabit Ethernet Controller
- Matrox G200eW Graphics
- Remote Management-IPMI 2.0 + IP-KVM with dedicated LAN
- Slim DVD
- 650W Redundant 80%+High Efficiency Power Supply

iX-N2280

- 2U Form Factor with 8 Hot Swap SAS/SATA 3.5" Drive Bays
- Dual Intel® 64-Bit Socket 1366 Quad-Core or Dual-Core, 5500 (Tylersburg/Nehalem) Series Processors
- Dual Intel® 5520 chipsets with QuickPath Interconnect (QPI)
- Up to 144GB DDR3 1333/1066/800 SDRAM ECC Registered Memory (18 DIMM Slots)
- 2 PCI-E 2.0 x16, 4 PCI-E x8, (1 in x16 slot) and 1 PCI-E x4 Expansion Slots
- Intel® 82576 Dual Port Gigabit Ethernet Controller
- Matrox G200eW Graphics
- Remote Management-IPMI 2.0 + IP-KVM with dedicated LAN
- Slim DVD
- 700W Redundant 90%+High Efficiency Power Supply

iX-N3216

- 3U Form Factor with 16 Hot Swap SAS/SATA 3.5" Drive Bays
- Dual Intel® 64-Bit Socket 1366 Quad-Core or Dual-Core, 5500 (Tylersburg/Nehalem) Series Processors
- Dual Intel® 5520 chipsets with QuickPath Interconnect (QPI)
- Up to 144GB DDR3 1333/1066/800 SDRAM ECC Registered Memory (18 DIMM Slots)
- 2 PCI-E 2.0 x16, 4 PCI-E x8, (1 in x16 slot) and 1 PCI-E x4 Expansion Slots
- Intel® 82576 Dual Port Gigabit Ethernet Controller
- Matrox G200eW Graphics
- Remote Management-IPMI 2.0 + IP-KVM with dedicated LAN
- Slim DVD
- 800W Redundant 80%+High Efficiency Power Supply

iXsystems

http://www.iXsystems.com 800-820-BSDI