FREE DVD INSIDE OPENBSD AND DRAGONFLY BSD

MAGAZINE BSD

Vol.1 No.2 Price USD14.99 AUD14.99 Issue 2/2008(2) 1898-9144

OpenBSD in the Limelight

Install & Configure Guide

EXCLUSIVELY

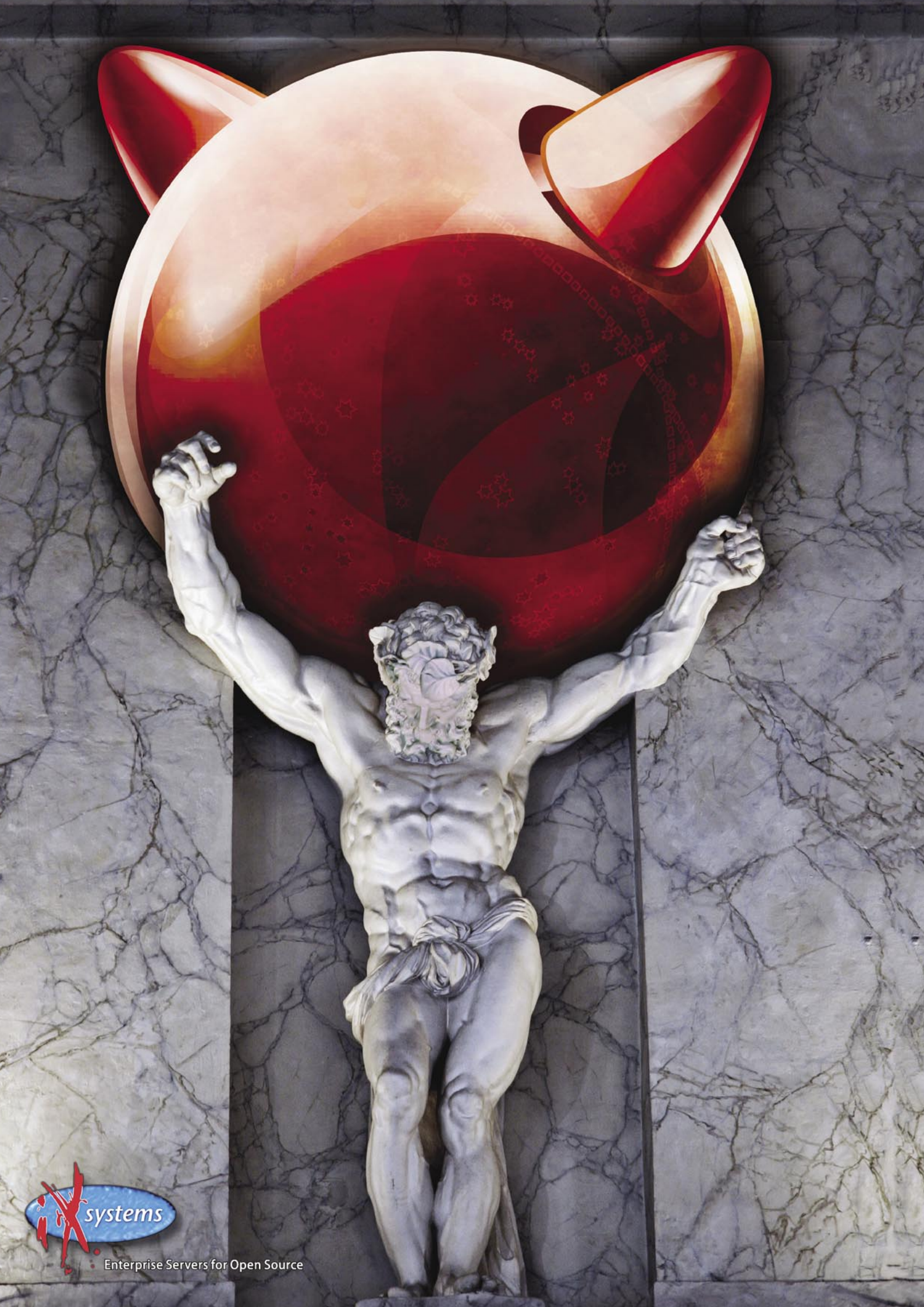
▶ **Interview with OpenBSD developer Damien Bergamini**

INSIDE

OpenBSD Packages in depth
Building an OpenBSD SAMP Server
Securing IM using Jabber/XMPP and TLS
Connecting to Other IM Networks
Kernel File System Development in Userspace
OpenBSD as a Desktop

Many useful articles and how-to's

0 74470 23877 5 02


systems
Enterprise Servers for Open Source

ROCK SOLID FREEBSD SUPPORT

For Your Peace of Mind

iXsystems is stepping up its game. In addition to our existing FreeBSD Hardware Support, we now offer Professional Enterprise Grade FreeBSD and PC-BSD Support as well. This means guaranteed response time support with customized service plans for any OS-related problem. And all from iXsystems, 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.

Here are just a few of the many reasons to purchase professional FreeBSD or PC-BSD Support from iXsystems:

Increased Productivity

Our team can provide you with a wide array of services to maximize your systems' performance. From basic support to complex solutions, the experts on the iXsystems Service Support Team can do it all. This leaves your system administrators free to work on other tasks while iX handles all the heavy-duty lifting, resulting in greater productivity, higher availability, and increased stability.

Improved Security

Your company's information is an invaluable resource. Ensuring the ongoing security of corporate assets is essential to daily operations and maintaining competitive advantage. The iXsystems Service Support Team will use our extensive experience to maximize the security of your valuable data and operations. Your company will be notified of all the latest security updates so that you can keep your resources safe from harm.

Custom Development and Consulting

iXsystems partners with some of the most brilliant minds in the FreeBSD Community to offer custom development and advanced level FreeBSD consulting solutions. Our Account Management Service Professionals will work with you to develop software solutions specific to your business operations. iXsystems offers kernel tuning and system optimization, device driver creation, kernel, userland, and embedded systems development, and a host of other services that allow your company to fully utilize the FreeBSD and PC-BSD platforms.

Engineering Escalation

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.

For more information contact iXsystems at (408) 943-4100 or visit our website at **http://www.ixsystems.com/rocksolid** and fill out the Inquiry form. We will pair you up with an Account Management Service Professional that can assess your needs and create a custom FreeBSD support plan for your organization!

MAGAZINE BSD

Editor in Chief: Ewa Dudzic
ewa.dudzic@bsdmag.org
Executive Editor: Karolina Lesińska
karolina.lesinska@bsdmag.org
Editor Assistant: Katarzyna Kaczor

Director: Ewa Dudzic
ewa.dudzic@bsdmag.org

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

Contributing: Gilles Chehade, Machtelt Garells, Rob Somerville, Petr Topiarz, Svetoslav P. Chukov, Antti Kantee, Michele Cranmer, Xavier Brinon, Peter N.M.Hansteen, Girish Venkatachalam, Eric Schnoebelen, Federico Biancuzzi, Mikel King

Special Thanks to: Matt Olander, Jim Brown

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
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 zo.o. is looking for partners from all over the World. If you are interested in cooperating with us, please contact us by 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.

The editors use automatic DTP system LaTeX

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

Subscription orders can be sent to
subscription@software.com.pl
Customer Service 1 917 338 3631

Dear All,

This is the second time we meet. I hope you enjoyed the first issue of our brand new mag and that you have been looking forward to this issue. As always, you're more than welcome to send in your comments, replies, ideas and suggestions. If you'd like to become a BSD author or betatester, don't hesitate - just sign in.

This time my thanks go to Matt Olander and Jim Brown for their great help in improving the quality of the magazine. In addition, I'd like to thank all of you who got involved with this project and devoted your free time to help and respond to all "emergencies", even in the middle of the night. Thank you!

This issue is devoted to OpenBSD. As usual, we tried to cover the most interesting and useful topics as well as providing how-to's that will help you improve your skills. Gilles Chehade guides you through the process of installation and configuration of OpenBSD 4.3 and Peter N.M.Hansteen gives you a kick-start on using packages. Gilles also teaches you how to provide the best development platform on OpenBSD.

Machtelt Garrels discusses the certification that is being developed by the BSD Certification Group Advisory Board. Rob Somerville demonstrates how to build an OpenBSD server from scratch, Petr Topiarz, from the Czech OpeBSD community, provides a guide for people who use Linux or FreeBSD and would like to give OpenBSD a try on the desktop and Svetoslav P. Chukov presents PBI - PC BSD installer.

In the administration section, Eric Schnoebelen and Michele Cranmer explain how to create a gateway between the Jabber network and closed networks and how to secure client-to-server and server-to-server communications using XMPP/Jabber features. Antti Kantee describes the kernel as a programming and testing environment.

We also decided to cover BSD in context of its use in business and education: Girish Venkatachalam explains how to use OpenBSD to make money and iXsystems presents the use of PC BSD in schools.

For those who don't really feel like getting into more technical details, Federico Biancuzzi interviews OpenBSD developer Damien Bergamini, Michel King introduces Mac OS X as the „other" BSD and Xavier Brinon reviews Absolute FreeBSD (2nd Edition) by Michael W. Lucas.

Enjoy!
all the best

Karolina Lesińska
Executive Editor

what's new

06 **BSD news**
Karolina Lesińska
Here you will find the future projects from BSD world, new releases and solutions, and much more- directly from people involved most in BSD community.

dvd contents

08 **DVD contents description**
Karolina Lesińska
If you are curious what is covermounted this time in our magazine you can find everything here

get started

10 **OpenBSD 4.3 Installation&Configuration**
Gilles Chehade
If you are new to OpenBSD distribution, Gilles guides you through the process of installing and configuring.

18 **You have installed it? Now what? Packages!**
Peter N.M. Hansteen
Peter give you the kick-start on packages, shows how to use them effectively and without much effort.

22 **OpenBSD**
Gilles Chehade
Gilles teaches how to provide the best development platform in form of a step-by-step tutorial for development station, server development, setting up the accounts and mail notification.

26 **BSD Certification by BSD Certification Group**
Machtelt Garells
Machtelt discusses the certification that is being developed by the BSD Certification Group Advisory Board- people who are actively involved in different BSD projects, key figures in their communities.

how-to's

30 **Building an OpenBSD SAMP Server with Content Filtering Proxy**
Rob Somerville
In this article Rob demonstrates how to build an OpenBSD server from scratch with Squid, Apache, MySQL, PHP and Webadmin.

38 **OpenBSD as a Desktop**
Petr Topiarz
Petr provides you with a guide for people who use Linux or FreeBSD and would like to give OpenBSD a try on the desktop and explains some general Unix routines.

40 **Inside the PBI System...**
Svetoslav P. Chukov
The author presents PBI – PC BSD installer with its unique and very useful package management system.

admin

44 **Connecting to Other IM Networks**
Eric Schnoebelen, Michele Cranmer
Eric and Michele follow up the article from the first issue. This time, they explain the mechanism to allow the creation of a gateway between the jabber network and closed networks- AOL Instant Messenger, Yahoo!, ICQ and others.

50 **Kernel File System Development in Userspace**
Antti Kantee
In this article Antti describes the kernel as a programming and testing environment. He also describes the kernel code way of testing and developing – all that to make it more comfortable for the user.

54 **Securing IM Using Jabber/XMPP and TLS**
Eric Schnoebelen, Michele Cranmer
This time, the authors will discuss how to secure client to server and server to server communications using XMPP/ Jabber features.

in business

58 **OpenBSD and Making Money**
Girish Venkatachalam
Even though corporations accuse Open Source for being unable to bring in the profits, in this article Girish shows that it is a serious bussiness that can make you rich.

review

61 **Absolute FreeBSD 2nd Edition**
Xavier Brinon
In this article Xavier analyses the Absolute FreeBSD 2nd edition – the Complete Guide to FreeBSD, a book written by Michael W. Lucas

62 **PC-BSD in Schools**
iXsystem
iXsystem presents PC-BSD in schools on the example of Polux School success story.

interview

64 **Interview with OpenBSD developer Damien Bergamini**
Federico Biancuzzi
Federico Biancuzzi talks about WPA with Damien Bergamini, the developer who made a huge work for OpenBSD wireless subsystem.

column

66 **Mac OS X the Other BSD**
Mikel King
Mikel King introduces Mac OS X – the other BSD.

Eldorado, Maximus and GSA

During the last 20 years malware (malicious software) has been constantly evolving and security programs have evolved in parallel. We have seen boot sector viruses, parasitic file-infecting viruses, macro viruses, mass-mailing worms, stand-alone backdoors, password stealers and various types of Trojans.

In the past two years we have seen better financed and organized malware development, possibly due to involvement of organized crime, but the other recent major development is the apperance of server-side polymorphism. Twenty years ago the lifespan of malware was measured in months, even years. Today, it is measured in hours. Dedicated servers distribute malware that changes every few minutes, faster than any anti-virus company can respond. Adding detection of each individual variant is of limited use, as it will no longer be in active distribution by the time the users of the anti-virus product receive the detection update.

The approach the F-PROT developers have taken is to increase the emphasis on heuristic detection, which detects potentially malicious behaviour in advance – not requiring updates for every single new variant. F-PROT pioneered heuristic scanning back in 1992, and over the years we have introduced various innovations, such as heuristics based on neural networks. The latest development in the F-PROT engine has been the introduction of three independent heuristic engines, code-named Eldorado, Maximus and GSA. Those engines use fundamentally different methods and are maintained by different teams, with a bit of a friendly in-house rivalry. The goal is that by the end of 2008 those three heuristic scanning engines will provide proactive detection of the vast majority of new malware – detect it as soon as it is released by the authors, without requiring any updates. It will never be possible to detect all malware proactively – any such claims are just irresponsible marketing hype, but Eldorado, Maximus and GSA will provide F-PROT users with a significant level of protection. That is our goal.

NetBSD now has UDF write support

Reinoud Zandijk has been working on support for the Universal Disk Format in NetBSD for quite some time, and in mid-May he reached another major milestone by adding write support to NetBSD's UDF file system. `It can now read and write files and directories on CD-R/RW, CD-MRW, DVD-R/RW, DVD+R/RW, DVD+MRW, (USB) flash media and harddisc partitions. Media like Iomega Rev should also work fine,' he said. In fact, this means that within NetBSD you now can mount any UDF formatted media and use your favorite tools, like cp, mv, rm, or even X11 file manager over it.

New default license for NetBSD

Following from a vote amongst the membership of the NetBSD Foundation and in recognition of the changing face of software licensing, the Foundation has changed its recommended license to be a two clause BSD license. Dropped clauses are the advertising clause and the "endorsement" clause (3 and 4 respectively). We have seen organizations and people concerned about the old clause 3 in the license, to the extent where NetBSD code could not be used in commercial products; the new license means that these concerns are no longer valid, said Alistair Crooks, The NetBSD Foundation's president. Also, the members of the NetBSD Foundation no longer considered clause 4 to be useful in today's software world.

All third parties are allowed and encouraged to change any previously used NetBSD Foundation license to the new two clause NetBSD license. Updated NetBSD copyright and licensing terms can be found at http://www.NetBSD.org/about/redistribution.html.

Getting ready for best release

The NetBSD source tree has been frozen in preparation for a new release. During the freeze period, no new functionality is being added to the tree, and only bug fixing is allowed. The pkgsrc, another major NetBSD project, has used freeze periods ever since it started making branches, in order to stabilize features in preparation for a stable branch. This practice has been successful over time, and now it was decided to try it for NetBSD releases too. As soon as source tree entered into freeze period, the NetBSD Release Engineering Teem, who manages all stable branches, is controlling all commits. The Releng will also define how long the freeze will take. It is expected that the upcoming NetBSD 5.0 release will contain many interesting features, like improved threading and SMP, new kernel scheduler supporting real-time classes, write support for UDF, Automated Testing Framework, EM64T/AMD64 and PAE support for Xen, as well as support for new hardware platforms and numerous devices.

by Mike M. Volokhov

Software News:

Firefox 3- Released June 17th. Just in case you've been living under a rock Firefox runs under the X Windowing System on all current versions of BSDs, as well as Mac OS X, Solaris, and of course Microsoft Windows. To download a binary version for your particular Operating System go to http://www.mozilla.com, however on most of the BSDs you will need to either install it from the ports or use pgksrc. OpenOffice.org 3.0- The public beta release of OpenOffice.org 3.0 is now ready for testing. This beta release is made available to allow a broad user base to test and evaluate the next major version of OpenOffice.org, but is not recommended for production use at this stage. If you are a regular user of OpenOffice.org, here's a great opportunity to help us make the next release the best ever. For more details, refer to the following URL: http://www.openoffice.org/project/marketing/3.0/announcementbeta.html

Update on the BSD Certification Group

The BSD Certification Group has recently released the BSD Associate certification exam. This exam is the first in a series of exams that focus on BSD systems. The exam covers seven diverse knowledge domains- Installing & Upgrading the OS and Software; Securing the Operating System; Files, Filesystems, and Disks, etc. The complete list is on the website.

The exam has been active since February 2008, and to date the exam has been held eight times in various cities in North America and Europe: Los Angeles, Ottawa, Krakow, Brussels, Toronto, Berlin, Ede, and Chemnitz. You might think that it's an easy exam and everyone passes, but that isn't the case. The failure rate is currently about 20%.

Why is the failure rate so high? Without a doubt, it's because people come to the exam expecting it to be a breeze and they find out it's not. The exam tracks very closely to the objectives that were published by the BSDCG in October 2005, the distribution of questions is pretty evenly distributed among the above domains, and it covers the four BSD versions- FreeBSD, OpenBSD, NetBSD, and DragonFly BSD.

The result? It's not a cake-walk. If you come to the exam with experience in a single version of BSD, you won't pass the exam. Comments from those who have taken it, said it was harder than I thought it would be and it made you think.

Now that the exam is out, there are many projects that the BSDCG would like to get started, such as the BSD Professional certification. This certification will probe even deeper into complex administrative tasks that BSD system administrators have to perform every day- filesystem issues and access controls, process control, virtualization, multiple network configurations, firewalls, and so forth. The good news is that there is a rich load of material to draw on- BSD systems contain a wealth of well documented features, thanks to developers all over the world.

The certification effort is community driven and everyone can help by spreading the word to local user groups, forums, schools and universities, etc.

To find out more about how you can help visit the website at www.bsdcertification.org.

Finally, Professional Support, Consulting, and Development for FreeBSD!

iXsystems has announced the launch of its Professional Enterprise Services and Support Division for FreeBSD and PC-BSD. We feel that offering Professional Level Support for FreeBSD and PC-BSD is one of the main barriers that the platforms face to expand adoption. While there may be some companies that are capable of supporting them, there are none, to my knowledge, currently offering services and support on an Enterprise Class level specific to FreeBSD and PC-BSD, says Matthew Olander, CTO of iXsystems. This is a barrier we are happy to remove. The service and support offerings will include customer support as well as customized offerings across a wide range of issues such as installation support, large deployments and kernel tuning.

It is also worth noting that iXsystems decided to open its own support center in the Midwestern United States as opposed to using an outside customer service firm. This has a number of advantages that company officials believe will enhance customer satisfaction. We have in-house professionals who have been working on various levels of the FreeBSD and PC-BSD projects for a very long time, who will be much more concerned about providing successful solutions for our customers and much more responsive than an outside firm, explained iXsystems CEO Michael Lauth.

A FreeBSD Laptop with Everything (mostly) Working?

In addition to launching its support division, iXsystems is currently putting the finishing touches on the Invincibook, a FreeBSD compatible laptop that will soon be available. The Invincibook is made with an anti-shock mounting design that protects the LCD and Hard Drive from damage and data loss. Additionally, it is water resistant to protect the internal components from accidental spills. The Invincibook will ship with Fibonacci, the upcoming release of PC-BSD, a powerful OS running FreeBSD 7 under the hood and featuring a powerful GUI for graphical system installation. PC-BSD installs applications via the Push Button Installer (PBI), a graphical utility to remove and install software in a simple to use, self-contained format.

PC-BSD Fibonacci Edition also features various new server tools and enhancements including speed improvements with the ULE Scheduler, experimental ZFS support during install, and UFS Journaling through GEOM.

Who are these Guys?

Formerly BSDi's hardware division, iXsystems, Inc. is a premier builder of FreeBSD-certified servers, storage, and related products. iXsystems develops custom hardware solutions that address a company's technical and budgetary needs within their specific network architecture.

OS compatibility is a key component of iXsystems' Open Source Hardware Design process. This means that they will work backwards to develop a custom solution ideal for the customer, instead of requiring the customer to compromise their specific hardware requirements and limit their choice of OS to fit within the parameters and specifications of a product line.

iXsystems is also the corporate sponsor of the PC-BSD Operating System and recently acquired FreeBSD Mall and BSD Mall, two providers of high quality BSD software, apparel, and literature. For more information visit the iXsystems website at http://www.ixsystems.com.

OpenBSD 4.3

This is a partial list of new features and systems included in OpenBSD 4.3. For a comprehensive list, see the changelog leading to 4.3.

- New/extended platforms:
  - OpenBSD/sparc64 SMP support. This should work on all supported systems, with the exception of the Sun Enterprise 10000.
- OpenBSD/hppa K-class servers like the K200 and K410 are supported now.
- OpenBSD/mvme88k SMP support on MVME188 and MVME188A systems. 88110 processor, and thus MVME197LE/SP/DP boards, are supported now.
- OpenBSD/sgi Contains many new drivers, however the kernel requires an important errata fix.

New tools:

- snmpd(8), implementing the Simple Network Management Protocol.
- The snmpctl(8) program controls the SNMP daemon.
- The pcidump(8) utility displays the device address, vendor, and product name of PCI devices.
- ldattach(8) is used to attach a line discipline to a serial line to allow for in-kernel processing of the received and/or sent data.

For more information about OpenBSD 4.3 please visit http://www.openbsd.org/.

Ampache 3.4.1

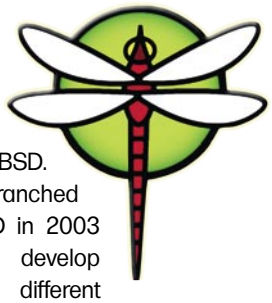
Ampache is a Web-based Audio file manager which is implemented with MySQL and PHP. It is one of the oldest applications of that type. Ampache's goal is to maintain a secure and fast web front end that will run on platorm that supports PHP and any hardware. It allows to create user accounts and share the music with other Ampache servers. It also allows you to modify your audio files via the web and it has support for playlists, album art, artist and album views, playback via Http/On the Fly Transcoding and Downsampling, Integrated Flash Player, Vote based playback, Icecast and Mpd, as well as per user themes and song play tracking. Ampache also provides an API for pulling out meta data in the form of XML documents.

The latest version – Ampache 3.4.1. contains many changes such as:

- Complete re-write in PHP5,
- AJAX'd interface,
- Active Playlist concept added,
- XML API,
- Dynamic Playlists,
- vastly improve browsing system.

For more infrmation please see:
http://www.ampache.org/

DragonFly 1.12.2



DragonFly is an operating system and environment originally based on FreeBSD. DragonFly branched from FreeBSD in 2003 in order to develop a radically different approach to concurrency, SMP, and most other kernel subsystems.

DragonFly belongs to the same class of operating system as BSD and Linux and is based on the same UNIX ideals and APIs. DragonFly gives the BSD base an opportunity to grow in an entirely different direction from the one taken in the FreeBSD, NetBSD, and OpenBSD series.

For more infrmation about 1.12.2 release please visit:
http://www.dragonflybsd.org/community/release1_12.shtml

MirBSD

MirOS BSD is a secure computer operating system from the BSD family for 32-bit i386 and sparc systems. It is based on 4.4BSD-Lite (mostly OpenBSD, some NetBSD). It is a derivative of OpenBSD. Source code from OpenBSD is regularly imported and merged. MirOS BSD often anticipates bigger changes in OpenBSD and includes them before OpenBSD itself. For example, ELF on i386 and support for gcc3 were available in MirOS first. Controversial decisions are often made differently from OpenBSD; for instance, there won't be any support for SMP in MirOS. The most important differences to OpenBSD are:

- Completely rewritten bootloader and boot manager without an 8 GiB limit and with Soekris support
- Slim base system (without NIS, Kerberos, Bind, i18n, BSD games, etc.), Bind and the BSD games being available as a port
- Binary security updates for stable releases
- ISDN support
- IPv6 support in the web server software
- wtf, a database of acronyms
- Some of the GNU tools (like gzip and *roff) were replaced by original UNIX code released by Caldera

For more infrmation please see:
http://www.mirbsd.org/main.htm

F-PROT Antivirus for BSD Workstations

For home users using the BSD open-source operating system, we offer F-PROT Antivirus for BSD Workstations. F-PROT Antivirus for BSD Workstations utilizes the renowned F-PROT Antivirus scanning engine for primary scan but has in addition to that a system of internal heuristics devised to search for unknown viruses.

F-PROT Antivirus for BSD was especially developed to effectively eradicate viruses threatening workstations running FreeBSD, NetBSD, or OpenBSD. It provides full protection against macro viruses and other forms of malicious software – including Trojans.

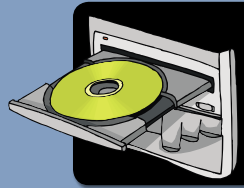
F-PROT Antivirus for BSD Workstations is FREE for Home Users

F-PROT Antivirus for BSD Workstations is FREE for use by personal users on personal workstations

Features

F-PROT for BSD Workstations features:

- Scans for over 1001738 known viruses and their variants
- Ability to perform scheduled scans when used with the Unix cron utility
- Scans hard drives, CD-ROMS, diskettes, network drives, directories and specific files
- Scans for images of boot sector viruses, macro viruses and Trojan Horses



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

MAGAZINE BSD 2/2008

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

OpenBSD 4.3
Installation& Configuration

Gilles Chehade

This issue of BSDMAG comes with a DVD containing the installation program for the OpenBSD 4.3 operating system. This article will help you go through the installation process and first steps at configuring and making use of this increasingly popular system.

OpenBSD is one of the four major BSD systems and follows the long tradition of giving away quality software without any strings attached. It is known for having a strong goal of security and advertising only two remote holes in the default install in more than ten years, but to be honest this is a side effect of a strong focus on keeping the code clean and not accepting dirty hacks for convenience. In the last few years, with other systems accepting to incorporate more and more closed-source objects (also known as blobs) in their systems, OpenBSD has gained another reputation of strong commitment to free software by refusing to sign non-disclosure agreements, removing support for non-friendly vendors and reverse-engineering drivers when other systems accepted the closed drivers provided by vendors and eventually made their integration easier. This is a rather short description but there are plenty of goals and going through all of them would probably make an article by itself.

So, let's get started with the installation !

Installation

OpenBSD has a reputation of having a very difficult installer for those who are used to the so-called *modern* GUI-based installers. In practice, despite the fact that it is console-based, the installation process is very easy if you take time to follow the instructions that are available in the FAQ and on-screen as installation goes on. After you are familiar with the very few steps, you will be able to perform complete installs in just a few minutes and amaze your friends.

Getting the media

OpenBSD is as free as can be and you can download it from the several FTP, HTTP, AFS and RSYNC mirrors listed on the official website; however it is strongly encouraged that you buy yourself a cd set as it is the main source of revenue for the project. Also, they are cool looking and come with stupendous stickers. To start installation, boot your computer on the DVD. You will be facing a boot prompt which is the entry point for you either to boot the system or the installer. You can simply press *enter* or wait for the bootloader to boot the default image. The installer will then load the kernel and you will see a lot of lines scrolling with information as to which devices were found or/and supported. After that, the following prompt will appear:

```
I)nstall, (U)pgrade or (S)hell? <i>
```

The options are quite self-explanatory, you can proceed to install by simply typing '`i`'. Next prompt will request for terminal type and keyboard mapping:

```
Terminal type: [vt220] <enter>
kbd(8) mapping? ('L' for list) [none] <enter>
```

Again, no dark magic, the terminal can be left to default if you are not doing the intall in a *weird* setup (from another machine connected to the setup machine through a serial cable for example). The keyboard mapping is up to you for obvious reasons, default will be an US qwerty. Even though I am a froggy, I happen to have a qwerty so no need for `fr` in my case. Make sure not to use any incorrect mapping or you may end up in an uncomfortable position when requested to enter a password.

The installer will then remind you that the install process is a destructive operation and that you should do backups. Seriously, do it.

```
Proceed with install? [no] <y>
```

Next step is where things get trickier and where reading skills are required in order not to break things. First, you are

prompted for the disk you will be installing OpenBSD on:

```
Available disks are: wd0
Which one is the root disk? (or done)
[wd0] <enter>
```

In this case I only have one disk so the list of available disks is pretty short. Once you validate the disk, you are asked if the disk will be fully dedicated to OpenBSD or not. Replying no will drop you into the fdisk utility where you can manage your partitions. I have not done a dual boot in years so I can only suggest you read the OpenBSD FAQ which explains the steps to do so, but to summarize you need to select which partition to use, set its type to A6 (OpenBSD) and write the MBR. These are three commands I will leave as an exercise to you.

After the disk has been selected (and eventually partitions set up), we will be dropped into the disklabel utility to slice the disk and define the mount points:

```
Initial label editor (enter '?' for
help at any prompt)
>
```

The help menu here should be sufficient to get you going, but to make it even more simple, here is the hint:

```
     (a)dd a slice
(d)elete a slice
(p)rint informations regarding the
slices
```

Each time you add a slice, you are prompted for information regarding the slice:

```
> <a>
name: [a] <enter>
offset: [0] <enter>
size: [78165360] 80M
Rounding to cylinder: 164304
FS type: [4.2BSD] <enter>
mount point: [none] /
>
```

It is recommanded that you create slices for `/`, `/home`, `/usr`, `/var`, `/tmp` and the swap though as long as you have a `/` slice OpenBSD should be happy. The sizes are really up to you and very dependent of what you plan to do with your system.

Keep in mind that if you create all the recommanded slices, `/` will not be very populated, `/usr` will be growing with each third party application or library you install, `/var` will be growing with each email, logs and runtime data that are going to be written to disk (runtime data includes databases if you plan on installing a package such as postgresql & friends). It is not too important that you get partitions right, but it is important that you do not get them wrong as it is easier to deal with adding a new slice than to deal with a *disk full* error. So, try to think from the beginning about what your computer will do and make sure each slice has enough space to work with.

There are no *standard* sizes, but if you create the five (`+ swap`) recommanded slices, a good rule is to have:

- `/ be` 150MB as there is not really any need for more – swap be close enough from your memory size so that in a worst case scenario where your kernel would crash, the core could be written on disk
- `/tmp`, this depends on your needs for temporary files, usually it can remain quite small; I usually make them 128MB and consider them already way too big.

The remaining space has to be balanced with your need to provide users with space for their home directories, your need to use third party applications and/or get a copy of the OpenBSD

Listing 1. Configure the network

```
Configure the network? [yes] <enter>
Available interfaces are: rl0.
Which one do you wish to initialize? (or 'done') [rl0] <enter>
Symbolic (host) name for rl0? [lappy] <enter>
The media options for rl0 are currently
   media: Ethernet autoselect (100baseTX full-duplex)
Do you want to change the media options? [no] <enter>
IPv4 address for rl0? (or 'none' or 'dhcp') dhcp
Issuing hostname-associated DHCP request for rl0.
DHCPDISCOVER on rl0 to 255.255.255.255 port 67 interval 1
DHCPOFFER from 192.168.0.1
DHCPREQUEST on rl0 to 255.255.255.255 port 67
DHCPACK from 192.168.0.1
bound to 192.168.0.42 -- renewal in 1800 seconds.
IPv6 address for rl0? (or 'rtsol' or 'none') [none] <enter>
No more interfaces to initialize.
DNS domain name? (e.g. 'bar.com') [my.domain] poolp.org
DNS nameserver? (IP address or 'none') [192.168.0.100] <enter>
Use the nameserver now? [yes] <enter>
Default route? (IP address, 'dhcp' or 'none') [dhcp] <enter>
Edit hosts with ed? [no] <enter>
Do you want to do any manual network configuration? [no] <enter>
```

Listing 2. Useradd comand live

```
# useradd -s /bin/sh -d /home/gilles -m gilles
# userinfo gilles
login   gilles
passwd  *************
uid     1000
groups  users
change  NEVER
class
gecos
dir     /home/gilles
shell   /bin/sh
expire  NEVER
#
```

source tree (installed in `/usr/src`), and your need to store email, databases, logs, websites, and other random data in `/var`. No one can make the choice for you, you are on your own.

Once we are done with the slicing, we only need to save and quit disklabel to go on. That is respectively the `(w)rite` and `(q)uit` commands. This was the trickiest part of the installation process:

```
> <w>
> <q>
```

Next step will have us confirm our slices and make sure we want to proceed to the formatting of our slices which will erase disk content:

The next step `*DESTROYS*` all existing data on these partitions! Are you really sure that you are ready to proceed? `[no] <y>`

Then the installer will set up its slices in an operation that takes more or less time depending on slice's size. When done, you are prompted for the system hostname:

```
   System hostname (short form, e.g.
'foo'): <lappy>
```

Since the installer allows installation through other media than a CD or DVD, like an ftp server for example, the network can be configured at install time. The configuration will be saved so that there is nothing to do post-install. Here I will use DHCP, but the configuration of a statically assigned IP address is straightforward: see Listing 1.

At this point, the network is configured and you are already able to ping the pc from another computer if you want to. Before going further, we are prompted for the root password:

- Password for root account? (will not echo)
- Password for root account? (again)

Make sure to use strong passwords, a mix of alphanumeric and punctuation is a minimum.

The last steps are selecting the install media:

- Location of sets? (cd disk ftp http or `done`) `[cd] <enter>`
- Available CD-ROMs are: `cd0`
- Which one contains the install media? (or `done`) `[cd0] <enter>`
- Pathname to the sets? (or `done`) `[4.3/i386] <enter>`

A list of sets is then displayed and we are prompted:

- Set name? (or `done`) `[bsd.mp] <*>`
- `[list of all selected packages]`
- Set name? (or `done`) `[bsd.mp] <done>`

Unless you know what you are doing, which would be doubtful if you are reading this far, you should install all sets. Not all are required, but OpenBSD is small enough that you do not need to go through the hassle of figuring out what is needed and what is not. A description of sets is available in the FAQ. If you still want to minimize the install, however keep in mind that a full install in OpenBSD is not at all the same than a full install on some other systems, a full install will not get you a multimedia player, ten text editors and twelve compilers. It will only install applications which are part of the OpenBSD base system.

As soon as you type `done`, the sets will start extracting. It should only take a few minutes depending on your system. Finally, the installer will prompt just a few questions regarding which services to start at boot time and your timezone:

Listing 3. Exit the installation

```
# exit

OpenBSD/i386 (lappy.poolp.org) (ttyC0)

login: gilles
Password:

OpenBSD 4.3 (GENERIC) #848: Tue Apr 29 20:30:06 MDT 2008

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

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

$ id
uid=1000(gilles) gid=10(users) groups=10(users)
$


I am logged as ``gilles'', let's see if I can issue commands as root:

$ sudo id

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

Password: <IlUvBsD42>
uid=0(root) gid=0(wheel) groups=0(wheel), 2(kmem), 3(sys), 4(tty),
5(operator), 20(staff), 31(guest)
$
```

- Start sshd(8) by default? [yes] <y>
- NTP server? (or none or default) [none] <default>
- Do you expect to run the X Window System? [no] <y>
- What timezone are you in? (? for list) [Canada/Mountain]? <Europe/Paris>

After a few seconds you should see:

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

That is all, OpenBSD is now installed and I will be able to log into it right after I issue a reboot. It took me what, five minutes ? Talk about an unfair reputation.

Post installation

Creating an account. After I boot for the first time on my brand new system, the following prompt welcomes me:

- OpenBSD/i386 (*lappy.poolp.org*) (ttyC0)
- login:

I can now log in as user root to start setting up the system. First thing to notice is that I got mail and that I am prompted for a terminal type. No need to argue, I will accept the default:

- You have new mail.
- Terminal type?: [vt220] vt220
- #

In case you cared, the mail was from Theo de Raadt, OpenBSD's project leader. A lot of useful information was in it. I would be happy to sum it up, but I guess it would spoil your fun. Now that I am logged in as root, first thing to do is create myself an account so that I can stop being logged as root. There are (more than) two tools which will allow me to do that:

- adduser is an interactive utility, a perl script if you are curious
- useradd is a command line utility

Both will get me through my goal of setting up an account, but they use different interface so it really is a matter of taste. Since I am not a big fan of interactive tools, my example will use useradd and you will get to read a couple man pages [adduser(8), useradd(8)] to see what other options I have not told you about. Happy? You'd better be, because in OpenBSD-land you will be reading a lot. See Listing 2.

Here, I only created the account gille' and specified it is shell and home, the -m option being to force creation of the home directory in case it does not exist. useradd has plenty of configuration options to ease account creation. One could for example set up an expiry time for account or password, or even a user class or group. the userinfo can, amongst other things, display a short summary with all information regarding a particular account. There is more use to it, but guess what ? Yup, [userinfo(8)]

For now, we do not really care about all this, all we want is to log in as user gilles and leave root as soon as possible. To do so, I lack a password:

- # passwd gilles
- Changing local password for gilles.
- New password: <foo>
- Please enter a longer password.
- New password: <foo123>
- Please use a more complicated password.
- Please use a different password. Unusual capitalization,
- control characters, or digits are suggested.
- New password: <IlUvBsD42>
- Retype new password: <IlUvBsD42>
- #

In the example above, the passwords enclosed in <' and '> did not show up on the terminal, however now that you see what I typed, you get to realize that a strong password policy is enforced by the passwd utility. It must not be too short, it must not

Listing 4. Interface name

```
$ ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33208
   groups: lo
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
rl0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
   lladdr 00:19:21:4c:6e:eb
          groups: egress
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 192.168.0.42 netmask 0xffffff00 broadcast 192.168.0.255
        inet6 fe80::219:21ff:fe4c:6eeb%rl0 prefixlen 64 scopeid 0x1
enc0: flags=0<> mtu 1536

pflog0: flags=141<UP,RUNNING,PROMISC> mtu 33208
        groups: pflog
$
```

Listing 5. Configure the network

```
for dhcp:
$ sudo tcsh
# echo "dhcp" > /etc/hostname.rl0
# exit
$
for my statically assigned address:
$ sudo tcsh
# echo "inet 192.168.0.42 255.255.255.0 NONE" > /etc/hostname.rl0
# exit
$
```

be too easy, it must be a real password. Obviously there is a way to get around this, but that kind of trick I will not tell you.

Now, I am able to log out from root and login in as user `gilles`, but I still need to do one last thing. Since being logged in as `root` is unsafe, and since my main account is `gilles`, I will configure the sudo utility to give me the ability to execute privileged commands as user `gilles`. The command `visudo` will allow me to edit the sudoers file.

```
# visudo
```

This will launch a text editor, vi by default, and by adding:

```
gilles    ALL=(ALL) SETENV: ALL
```

in the *User privilege specification* section of the file, I will be able to execute commands as root by prefixing them with `sudo`. Beware that executing commands through sudo is not safer than executing them through root, but this forces you to ask yourself the question *do I really want to do this?* every time you start typing `sudo` in your shell.

Now is time to kiss root goodbye! (see Listing 3)

It seems to work pretty well. Note that there is a way to disable the password prompting, but keeping it makes it annoying enough that you do not end up doing sudo commands all the time. It forces you to think about what you are doing in your session instead of blindly prepending sudo everywhere, as a side effect a session that you would forget to lock will not make your system compromised. A user that shares your computer and attempts to *brute force* your sudo account will trigger mail being sent to `root`. We will later see how to alias the root account to my unprivileged account as root happens to receive mail we DO care about.

Just a few words before we go further

As I said in previous section, in OpenBSD-land we get to read a lot. This is a habit that is kind of strange to newcomers who are used to having their hands herd and being walked from a problem to its solution. However, in OpenBSD-land you do not get helped if you do not try to get to a solution by yourself. Since a lot of work is done on keeping documentation up-to-date, the first step to a solution is often to start reading the documentation that is shipped with the system.

Why do I mention this? Well, the very first thing you get to do when booting your OpenBSD system and logging in to your account is to actually read a man page `[afterboot(8)]`. It holds a description of the first checks to perform after the first boot. It will tell you about files that were configured during installation as well as files and commands that you should really know how to use as soon as possible. Since repeating its content here would be a waste of bytes, I will only suggest that you read it and follow the pointers to other man pages that are in the "READ ALSO" section of the man page.

Configuring the network

One of the first thing you will want to do is configuring the network since you will probably want to interact with the rest of the world. Depending on your network configuration, this is going to be easy, or super easy. First of all, you will need to know your interface name: see Listing 4.

Here, my interface is `rl0` (`lo0` being the loopback interface, `enc0` and `pflog0` being of interest to you only when you will be familiar enough that you will want to setup ipsec or `pf`). This means that my network card is attached to the rl driver `[rl(4)]`. Good, now:

- If I have a DHCP server: `$ sudo dhclient rl0`
- If I want to statically assign my address: `$ sudo ifconfig rl0 192.168.0.42`

The changes are not permanent and to make sure they are kept after the next reboot, all that is required is to write them to the `/etc/hostname.rl0` file which will be read at boot time. For each interface you want to configure, a `/etc/hostname.<interface>` file should exist with the configuration written in it (Listing 5).

If we go for DHCP, there is nothing left to do. If we go for the statically assigned address, we still need to configure the gateway and nameservers. Configuring the gateway is simple:

```
$ sudo route add default 192.168.0.1
and to make the change permanent:
$ sudo tcsh
# echo "192.168.0.1" > /etc/mygate
# exit
$
```

Nameservers are configured in the /etc/resolv.conf file:

Listing 6. The list of ports

```
$ make search name="tcsh"
Port:   tcsh-6.15.00
Path:   shells/tcsh
Info:   extended C-shell with many useful features
Maintainer: The OpenBSD ports mailing-list <ports@openbsd.org>
Index:  shells

L-deps:
B-deps:
R-deps:
Archs:  any
$
```

Listing 7. etc/vc.conf file

```
[...]
# use -u to disable chroot, see httpd(8)
httpd_flags=NO    # for normal use: "" (or "-DSSL" after reading ssl(8))

# For normal use: "-L sm-mta -bd -q30m", and note there is a cron job
sendmail_flags="-L sm-mta -C/etc/mail/localhost.cf -bd -q30m"
[...]
```

```
$ sudo vi /etc/resolv.conf
search poolp.org
nameserver 192.168.0.2
nameserver 192.168.0.3
lookup file bind
$
```

Once you get more familiar with the system, you can run the shipped `named` server and configure your system to use its own name server.

Wi-Fi is slightly more difficult, you must... no, actually in OpenBSD Wi-Fi is configured using ifconfig which recognizes a few additional Wi-Fi-specific options. I would love to put an example, sadly I do not have wifi so you will have to trust my word.

What is in OpenBSD

Now that we have a network, let's see what tools we have at hand and eventually install from the Internet additional software.

Contrary to popular belief, OpenBSD comes with many tools which makes it usable out of the box to achieve many of the goals you would expect from a UNIX-like system:

OpenBSD can be configured as a simple firewall with simple rules to block incoming and outgoing traffic, just as it can be used to control bandwidth or provide high availability redundant setups, multihoming or ipsec gateway. It is the system of choice to use as a gateway between a network and another, a very robust system with advanced network related features.

It can also be configured as a server for a wide variety of services including http, smtp, dns, dhcp, pop, ftp, ssh, ntp, and more... Services are integrated and for the most part will run out of the box if you enable them as will be shown in this article. Services which cannot work out of the box because they require specific configuration come with examples that will allow an unfamiliar admin to get them running in minutes. It should also be noted that most of these services are either written by OpenBSD hackers or are modified to improve their overall security with techniques that have proved to be efficient such as privilege separation and chrooting, privilege dropping, use of safe alternatives to potentially dangerous code constructs, and so on. Some services are even able to cooperate with the packet filter to provide elegant solutions to problems which usually force admins to rely on hacks, such as ftp-proxy, spamd or relayd.

It makes a great development station. Xorg is available by default in a more secure OpenBSD-ized version. vi and mg, an emacs-like editor, are there out of the box, as are cvs, gcc, gdb and more. The documentation is probably the best out there with every function documented, some even providing examples of correct and incorrect uses. It is not rare that I rely on OpenBSD man pages while developing for Linux, and I know of many people with the same habit.

As you can see, the system comes with a set of applications which are available out of the box and which will allow you to do quite a few things in many areas without having to install third party applications.

Ports and packages

At some point, you will feel limited because you need a particular tool to do your job, or you will miss an application you are familiar with and which does not ship with the system. I tried a lot in the past to limit myself to base applications but in the end I always end up needing something that's missing. Fortunately, OpenBSD provides two mechanisms to ease the installation of third party applications and have them installed and running painlessly: `packages` and `ports`.

Packages are a collection of archives containing software and libraries that are under a license which allows the OpenBSD project to host them on a public ftp server and redistribute them. This does not necessarily mean that they are free software, it only means that they are software that is allowed to be distributed. Packages are managed through a set of commands:

```
pkg_add, pkg_delete and pkg_info
```

Actually there is more, but by now you should be use to me telling you to read man pages.

To install a package, you need to tell `pkg_add` where to find it. This is done by setting the `PKG_PATH` environment variable to the ftp directory that contains the package you want to install. A list of these servers is available at [1]. Since we want to be nice to with the main server and we want the application to install fast, I will chose a server that's geographically close to me, *ftp.arcane-networks.fr*, to install the `screen` utility that I like so much:

```
$ export PKG_PATH=
ftp://ftp.arcane-networks.fr/pub/
OpenBSD/4.3/packages/i386/
$ sudo pkg_add screen
Ambiguous: screen could be screen-
4.0.3p1 screen-4.0.3p1-shm screen-
4.0.3p1-static
$
```

The `pkg_add` utility detected that there are 3 different packages for `screen`, and it is up to me to decide which one I will want to use. In this case, I do not really care about the various versions and will go for the default:

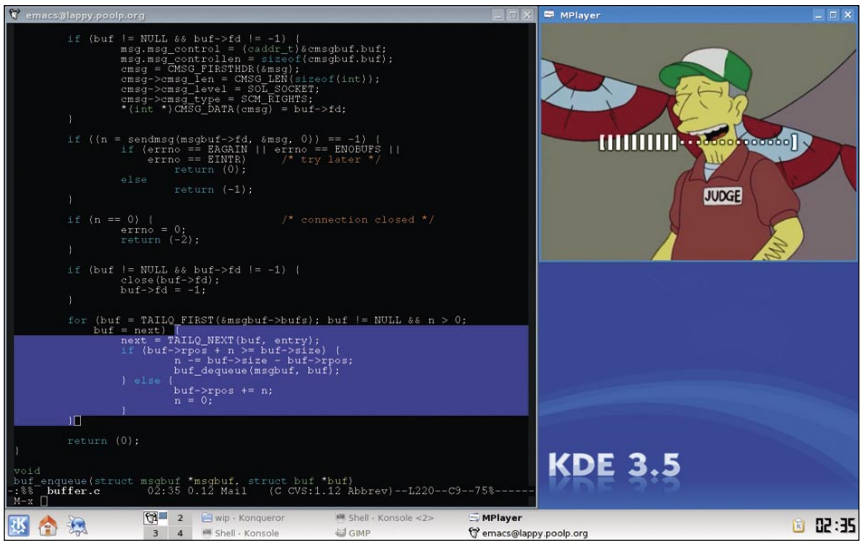


Figure 1. Emacsm player

```
$ sudo pkg_add screen-4.0.3p1
screen-4.0.3p1: complete
$
```

Just note that usually, the existence of more than one *flavor* of a package is an indication that you should educate yourself as to what the different versions do. In many occasions, a flavor is here to compensate for the lack of an option in the default package.

Whoops, what I really wanted was the -static flavor. No problem, uninstalling it is simple and will clean up every file that was created at install time:

```
$ sudo pkg_delete screen
screen-4.0.3p1: complete
Clean shared items: complete
$
```

Now I install the right version:

```
$ sudo pkg_add screen-4.0.3p1-static
screen-4.0.3p1-static: complete
$
```

The `screen` example is simple because it does not have dependencies, but to be honest it does not make a difference as `pkg_add` resolves and installs all of the dependencies transparently.

Unlike packages, ports are a collection of Makefiles that are organized in a hierarchy of directories (typically under /usr/ports) and which allow you to download, build and install any of the (slightly more than) 5000 ported software and libraries by typing `make install` in the appropriate directory. To obtain the ports, you need to download the `ports.tar.gz` archive that is available on every mirror, or use cvs. There is not really any advantage to use ports if a package already exists for the application you want to install as building the port will result in the package itself. Ports are handy when you are dealing with situations which cannot be solved by packages, for example if you need to build with a particular option, or if the application has a restrictive license that does not allow OpenBSD to distribute a package.

To install the ports subsystem, you need to extract the `ports.tar.gz` that you will find on every mirror inside `/usr/ports`:

```
$ sudo mkdir /usr/ports
$ ftp ftp://ftp.arcane-networks.fr/
pub/OpenBSD/4.3/ports.tar.gz
$ sudo tar -C /usr/ports -zxpf ~/
ports.tar.gz
$
```

This will create the ports hierarchy where application are classified by category. For example, if I wanted to install the `tcsh` shell, I would issue a `make install` in `/usr/ports/shells/tcsh/` which would in turn download the source for tcsh from a master site, compile it, create a package out of it, and install the package like we have seen earlier. The list of ports is in `/usr/ports/INDEX` which can be parsed easily from the command line or searched through with make commands, for example: Listing 6.

With this knowledge, you should already be able to customize your OpenBSD system and set up an environment that you will enjoy working in within a few minutes.

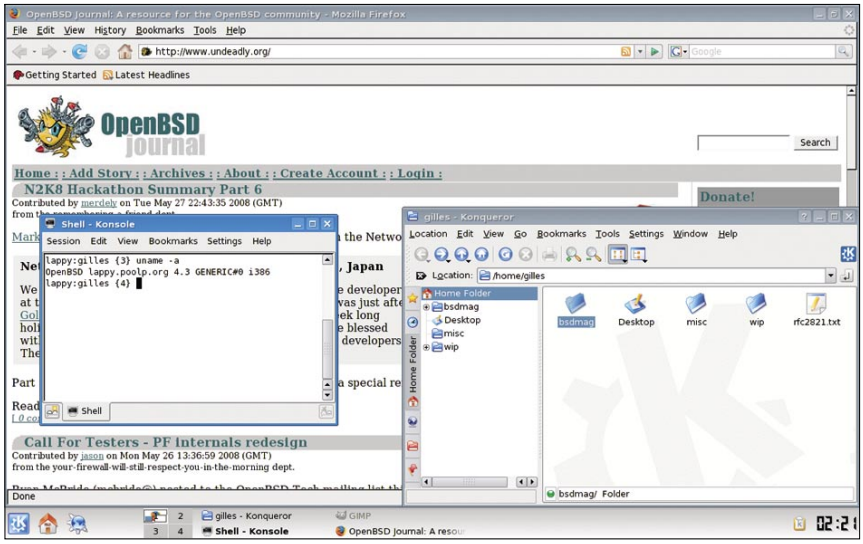


Figure 2. kde

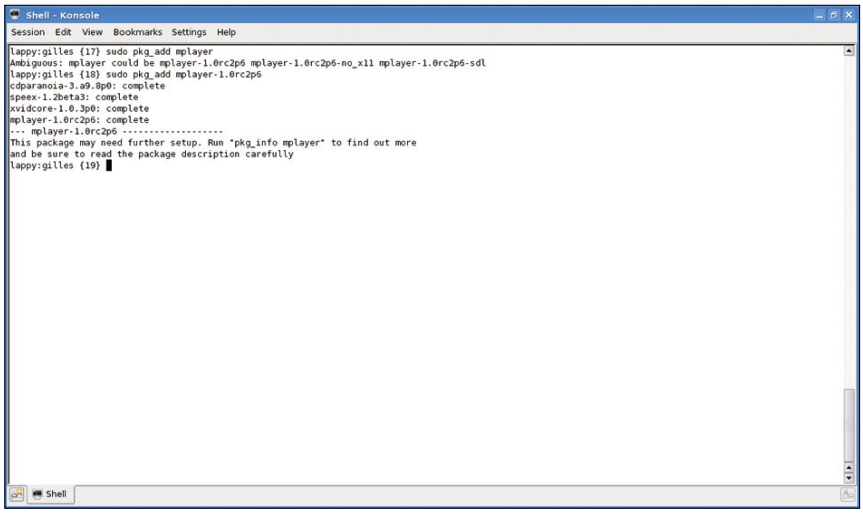


Figure 3. pkgadd

Basic administration

X configuration – I am pretty sure you want X running by now. OpenBSD ships with an Xorg and you should not need any configuration as settings are auto-detected. The only thing you may want to do if the default window manager, `fvwm2`, does not suit you is to install the window manager of your choice:

```
$ sudo pkg_add ion
ion-20070318p1: complete
$
```

Then if you do not plan to use `xdm`, add the command line to your `~/.xinitrc` file so that starting X will start the window manager:

```
$ echo /usr/local/bin/ion3 > ~/
.xinitrc
```

X can now be started with the well known command:

On the 'Net

- [0] The OpenBSD project: *http://www.openbsd.org/*
- [1] FAQ: *http://www.openbsd.org/faq/*
- [2] Goals: *http://www.openbsd.org/goals.html*
- [3] Mailing lists: *http://www.openbsd.org/mail.html*

```
$ startx &
```

Making changes to user account – Sometimes you may need to change some of the settings for your account. I like the tcsh shell which is not shipped with OpenBSD, so how do I change from the `/bin/sh` shell that I got when I created my account to the `/usr/local/bin/tcsh` shell that I got from running `pkg_add tcsh`?

```
$ chpass -s /usr/local/bin/tcsh gilles
```

or:

```
$ chpass
```

When invoked without parameters, the `chpass` command will launch a text editor which will allow me to change a few settings such as my name (as will appear in `/etc/passwd`, finger output and automatic filling from various mail clients) or shell.

Starting daemons

If you are familiar with other Unix(-like) systems, you probably know that most administrative files are stored in the `/etc` hierarchy.

Files that are of particular interest are the rc files which are used to decide what will or will not be done at boot (and reboot) time by the `/etc/rc` script.

First file to take a look at is `/etc/rc.conf` which holds a series of knobs to enable and disable services at boot time. For example: see Listing 7.

The `/etc/rc.conf` file ends up with the inclusion of `/etc/rc.conf.local` if it exists. The smart way of doing things is to override the variables we want changed in `/etc/rc.conf.local` and not making any changes to `/etc/rc.conf` so that they do not get overwritten during next upgrade. So, if I were to enable httpd and disable sendmail, I would simply add the following lines to `/etc/rc.conf.local`:

```
httpd_flags=""
sendmail_flags=NO
```

Obviously, OpenBSD cannot (and would not) take into account inside `/etc/rc.conf` every single service that are present in ports and packages. So an additional file, `/etc/rc.local` is executed at boot time and may contain command lines of your choice. For example, if I had installed the dovecot imap server and wanted it to be started automatically at next boot, I would simply add the following to

```
/etc/rc.local:
if test -x /usr/local/sbin/dovecot;
then
   /usr/local/sbin/dovecot; echo -n '
imapd';
fi
```

Other rc files exist, such as `/etc/rc.shutdown` and `/etc/rc.securelevel` but I doubt they deserve much more explanations.

Tweaking the kernel settings

There are some kernel settings which you can change from userland without having to rebuild a kernel. Amongst these settings, the ability to forward packets (required if you plan to use your OpenBSD computer as a gateway) or even Linux and FreeBSD binary emulation if you plan to run an application for which you do not have source code and which does not exist for OpenBSD.

These knobs can be listed and altered with the `sysctl` command; however the changes are not persistent between reboots. The file `/etc/sysctl.conf` is a good place to save these changes.

Password files

OpenBSD does not use a text file to store user accounts and passwords. It uses a database out of which the `/etc/passwd` and `/etc/master.passwd` are generated. This means that any changes to `/etc/passwd` or `/etc/master.passwd` will not be taken into account and will be overwritten next time the files are regenerated from the database. Changes to accounts must be done through the several utilities that manipulates the database itself. A notable exception is `/etc/group` which can be edited by hand, though using utilities is still smarter and less error prone.

Learning more

This article was just to get you started and running by holding your hand for the first few minutes. The next step for you is to start reading from the project's FAQ and man pages to get more familiar with the tools and how they work.

Our community is active and you should be able to find help on almost any topic through the official website, the *misc@openbsd.org* mailing list, or even through the *www.undeadly.org* website which often posts worthy information about new tools that can make your life easier.

One thing to note is that the OpenBSD community does not attempt to bring users at all costs and people tend to be direct and unfriendly when faced to anyone who begs for help without doing the slightest effort to find a solution by himself. When someone asks for help, it is expected that he did his best to find a solution, describe the problem clearly with logs and error messages that will help others understand, list what was attempted to solve the issue and where you are stuck. Not doing so is very likely to make people yell at you because being lazy saves your time but wastes the time of others which is considered by many as a rude and impolite behavior.

I hope you enjoyed reading this article and you will enjoy using OpenBSDas much as I do !

About the Author

Gilles Chehade is a research and development engineer for french search engine Exalead, and has had experiences in various areas as a security consultant, system and network administrator or instructor for Unix and programming classes in the past. He has been a BSD user for nearly a decade with a large preference for OpenBSD which he joined late 2007 to contribute on userland and daemons.

You've installed it.
Now what? Packages!

Peter N. M. Hansteen

A freshly installed machine nice, but it's when you start using the package tools that the real visitas open. Read on for a kickstart on packages.

Installing OpenBSD is easy, and takes you maybe 20 minutes. Most articles and guides you find out there will urge you to take a look at the files in `/etc/` and explore the man pages to make the system do what you want. With a modern BSD, the base system is full featured enough that you can in fact get a lot done right away just by editing the relevant files and perhaps starting or restarting one or more services. If all you want to do is set up something like a gateway for your network with basic-to-advanced packet filtering, everything you need is already there in the basic install.

Then again, all the world is not a firewall, and it is likely you will want to use, for example, a web browser other than the venerable lynx or editing tools that are not vi or mg. That's where packages and package systems come in. I will skip a little ahead of myself and make a confession: The machine I am writing this piece on reports that it has some 260 packages installed.

Before we move on to the guts of this article, some ceremonial words of advice: If you are new to OpenBSD or it is your first time in a while on a freshly installed system, you could do a lot worse than spending a few minutes reading man afterboot. That man page serves as a handy checklist of things you should at least take a peek at to ensure that your system is in good working order.

Some packages will write important information, such as strings or stanzas to put in your `rc.conf.local`, `rc.local` or `sysctl.conf` files, to your terminal. If you are not totally confident what to do after the package install finishes, it may be a good idea to run your ports and packages installs in a script session. See man script for details.

When dinosaurs roamed the Earth...

The story of the ports and packages goes back to the early days of free software when we finally found ourselves with complete operating systems that were free and hackers^H^H^H^H^H^H system administrators found that even with full featured operating systems such as the BSDs, there were sometimes things you would want to do that was not already in there. The way to get that something else was usually to fetch the source code, see if it would compile, make some changes (or a lot) to make it compile, possibly introduce the odd `#ifdef` block and keep at it until the software would compile, install and run. In the process you most likely found out what, if any, other software (tools or libraries) needed to be installed to complete the process. At that point, you could claim to have `/ported/` the software to your platform. If you had been careful and saved a copy of the original source files somewhere, you could use the diff utility to create a patch you could then send to the program maintainer and hope that he or she would then incorporate your changes in the next release.

But then, why wait for the next release? Why not share those diffs with others? How about putting it into a CVS repository that would be available to everyone? That idea was tossed around on relevant mailing lists for a while, and the first version of the `/ports system/` appeared in FreeBSD 1.0 in December 1993.

The other BSD systems adopted the basic idea and framework soon after, with small variations. On NetBSD, the term `'port'` was already in use for ports of the operating system itself to specific hardware platforms, so on that operating system, the ports tree is referred to as `'package source'`, or `/pkgsrc/` for short. The ports and packages tools are still actively maintained and developed on all BSDs, and most notably Marc Espie rewrote the `pkg_*` tools for OpenBSD's 3.5 release.

Parallel development has lead to some differences in the package handling on the various BSDs, and some of the operations I describe here from an OpenBSD perspective may not be identical on other operating systems. Around the same time the BSDs started including a ports tree and packages, people on the Linux side of the fence started developing package systems too. With distributed development taken to the point where the

kernel, basic system tools and libraries are maintained separately, perhaps the need there was even greater than on the BSDs. In fact, some Linux distributions such as the Debian based ones have taken the package management to the point where *everything is a package* - every component on a running system is a package that is maintained via the package system, including basic system tools, libraries and the operating system kernel. In contrast, the BSDs tend to treat the base system as a whole, with the package management tools intended solely for managing software that does not come as a part of the default install.

The anatomy of ports and packages

The ports system consists of a set of 'recipes' to build third party software to run on your system. Each port supplies its own Makefile, whatever patches are needed in order to make the software build and optionally package message files with information that will be displayed when the software has been installed.

So to build and install a piece of software using the ports system, you follow a slightly different procedure than the classical fetch - patch - compile cycle. You will need to install the ports tree, either by unpacking `ports.tar.gz` from your CD set or by checking out an updated version via cvs, or for that matter cvsup or the rewritten version called csup. With a populated ports tree in hand, you can go to the port's directory, say

```
$ cd /usr/ports/print/lyx
```

to see about installing lyx, the popular latex front end. On a typical OpenBSD system, that directory contains the following files:

```
$ ls -l
total 8
-rw-rw-r--  1 root  wheel  1825 May
18 21:57 Makefile
-rw-rw-r--  1 root  wheel   274 Apr
5  2007 distinfo
drwxrwxr-x  2 root  wheel   512 Nov
1  2007 patches
drwxrwxr-x  2 root  wheel   512 Nov
1  2007 pkg
```

here, the Makefile is the main player. If you open it now in a text editor or viewer such as less, you will see that the syntax is quite straightforward. What it does is mainly to define a number of variables such as the package name, where to fetch the necessary source files, which programs are required for the compile to succeed and which libraries the resulting program will need to have present in order to run correctly.

The file defines a few other variables too, and you can look up the exact meaning of each in the man pages, starting with man ports and man bsd.port.mk. With all relevant variables set, at the very end the file uses the line:

```
.include <bsd.port.mk>
```

to pull in the common infrastructure it shares with all other ports.

This is what makes the common targets work, so for example, typing:

```
$ make install SUDO=sudo
```

(probably the most common port-related make command for end users and administrators) in the port directory will start the process to install the software. But before you type that command and *press Enter*, you may want to consider this: This command will generate a lot of output, most likely more than will fit in the terminal's buffer. If the build fails, it is likely that the message about the first thing that went wrong will have scrolled off the top of your screen and out of the terminal buffer. For that reason, it is good sysadmin practice to create a record of lengthy operations such as building a port by using the script command. Typing script in a shell will give you a subshell where everything displayed on the screen will be saved in a file. Escape sequences, asterisk-style progress bars and *twirling batons* will end up a bit garbled, but that essential message you are looking for will be there too. man script will give you the details, and unless you are an incurable packrat, do remember to delete the typescript file afterwards. That process will start with checking dependencies, go on with downloading the source archive and checking that the fetched file matches the cryptographic signatures stored in the distinfo file. If the signatures match, the source code is extracted to a working directory, the patches from the patches/ directory are applied, and the compilation starts. If the dependency check finds that one or more pieces are missing, you will see that the process fetches, configures and installs the required package before continuing with the build process for the original package.

After a while, the package build most likely succeeds and the install completes. At this point you will have a new piece of software installed on your system. You should be able to run the program, and the installed package will turn up in the package listings output by `pkg_info`, such as:

```
$ pkg_info | grep lyx

lyx-1.4.3p2-qt      graphical frontend
for LaTeX (nearly WYSIWYG)
```

This information is taken from the package's subdirectory in `/var/db/pkg`, where the information about currently installed packages is stored.

If you paid close attention during the make install process, you may have noticed that the install step was performed from a binary package. This is one of the distinctive features of the OpenBSD version of the package system. The package build always generates an installable package based on a `'fake'` install to a private directory, and software is always installed on the target system from a package.

But you do not need to do that!

This means several things. If you have built and installed a package by typing `'make install'` in the relevant ports directory and later run the `'make deinstall'` or `pkg_delete` to remove the software, any subsequent install of the software will take place from the package file stored in a subdirectory of `/usr/ports/packages`. But more importantly, in most cases you can keep your system's packages up to date without a ports tree on the machine. (See Note [1]) For each release, a full set of packages is built and made available on the OpenBSD mirrors, and by the time you read this, there is reason to hope that running updates to -stable packages will be available for supported releases too.

The way to make good use of this is to set the `PKG_PATH` variable to include the packages directory for your release on one or more mirrors close to you and/or a local directory, and then run `pkg_add` with the `-u` flag. (See Note [2])

My laptop runs -current and I am in Europe, so the PKG_PATH is set to

```
PKG_PATH=ftp://ftp.eu.openbsd.org/pub/
OpenBSD/snapshots/packages/`machine
-a`/
```

On a more conservatively run system, you may want to set it to something like

```
PKG_PATH=ftp://ftp.eu.openbsd.org/pub/
OpenBSD/4.3/packages/`machine -a`/
```

Once your PKG_PATH is set to something sensible, you can use `pkg_add` and the package base name to install packages, so a simple

```
$ sudo pkg_add lyx
```

would achieve the same thing as the `'make install'` command earlier, and most likely a lot faster too. Once you have a set of packages installed, and keeping in mind that you need a meaningful `PKG_PATH`, you can keep them up to date using `pkg_add -u`. If you want more detailed information about the package update process and want `pkg_add` to switch to interactive mode when necessary, you can use something like this command:

```
$ sudo pkg_add -vui
```

I have at times tended to run my `pkg_add -u` with some of the `-F` flags in order to force resolution of certain types of conflict, but given the quality of the work that goes into the packages, most of the `-F` options are rarely needed.

`pkg_add` and its siblings in the `pkg_*` tools collection has a number of options we have not covered here, all intended to make your package management on OpenBSD as comfortable and flexible as possible. The tools come with readable man pages, and may very well be the topic of future BSD Magazine articles.

More information on the net

The main source of information about the OpenBSD ports and packages system is to be found on the OpenBSD project's web site. The FAQ's ports and packages section at *http://www.openbsd.org/faq/faq15.html* has more information about all the issues covered in this article, and goes into somewhat more detail than space allows here. If you encounter problems while installing or managing your packages, it is more than likely that you will find a solution or a good explanation there. And of course, if nothing else works or you can't figure it out, there is always the option of asking the good people at *misc@openbsd.org* or *ports@openbsd.org* or search the corresponding mailing list archives.

How do I make a package then?

That is a large question, and the first question you should ask if you think you want to port a particular piece of software is, *Has this already been ported?*. There are several ways to check. If you are thinking of creating a port, you most likely already have the ports tree installed, so using the ports infrastructure's search infrastructure is the obvious first step. Simply go to the `/usr/ports` directory and run the command:

```
$ make search key=mykeyword
```

Where mykeyword is a program name or keyword related to the software you are looking for. One other option with even more flexible search possibilities is to install databases/sqlports. And of course, searching the ports mailing list archives (*http://marc.info/?l=openbsd-ports*) or asking the mailing list works too.

When you have determined that the software you want to port is not already available as a package, you can go on to prepare for the porting effort. Porting and package making is the subject of much usenet folklore and rumor, but in addition you have several man pages with specific information on how to proceed. These are, ports, package, packages, packages-specs, library-specs and `bsd.port.mk`.

Read those and use your familiarity with the code you are about to port to find your way. The OpenBSD web offers a quite a bit of information too. You could start with re-reading the main ports and packages page at *http://www.openbsd.org/faq/faq15.html*, and follow up with the pages about the porting process at *http://www.openbsd.org/porting.html*, testing the port at *http://www.openbsd.org/porttest.html* and finally the checklist for a sound port at *http://www.openbsd.org/checklist.html*.

All the while, try first to figure out the solution to any problems that pop up, read the supplied documentation, and only then ask port maintainers via the ports mailing list for help. Port maintainers are generally quite busy, but if you show signs of having done your homework first, there is no better resource available for helping you succeed in your porting or port maintenance efforts.

One fine resource for the aspiring porter is Bernd Ahlers' ports tutorial from OpenCon 2007, you can look up Bernd's slides at *http://www.openbsd.org/papers/opencon07-portstutorial/index.html*, and it is possible he can be persuaded to repeat the tutorial at a conference near you.

Notes

[1] The main exceptions to the rule that precompiled packages are available from the mirrors are software with licenses that do not allow redistribution or require the end user to do specific things such as go to a web site and click a specific button to formally accept a set of conditions. In those cases it cant' be helped, and you will need to go via the ports system to create a package locally and install that.

[2] If you want to find out what packages are available at your favorite mirror, you can get a listing of package names by fetching the file `$PKG_PATH/index.txt`. The OpenBSD web site offers a listing of available packages with short descriptions, too. For OpenBSD 4.3, the listing is available from *http://www.openbsd.org/4.3_packages/*, from there you click on the link for your platform

About the Author

Peter N. M. Hansteen is the author of The Book of PF (No Starch Press, December 2007). Peter has been tinkering with computers and networks since the mid-1980s, found the Freenixes in the early 1990s and is a frequent lecturer on PF and other OpenBSD and FreeBSD topics. He is a consultant, sysadmin and writer based in Bergen, Norway who occasionally blogs at http://bsdly.blogspot.com/ and welcomes your comments to *peter@bsdly.net*.

Bringing the *BSD UNIX Community Together in New York City

October 11-12, 2008, Columbia University

NYCBSDCon 2008

nycbsdcon.org

in association with:

BUG
NEW YORK CITY *BSD USER GROUP

OpenBSD
the best development platform

Gilles Chehade

Amongst the many goals of OpenBSD, there is one which is important enough that it is listed in the first position of the goals page: "Provide the best development platform possible."

This is a goal that works hand in hand with the hard focus on code quality. If the system provides good tools and documentation for the developers, then they will be more likely to contribute good code. Looking at the *tech@ mailing* list shows this behaviour with thousands of diffs being called *incomplete* for not providing the associated documentation, or being asked for changes if they are not doing things the appropriate way. Undocumented code does not get in and bad code does not get in either.

As a direct result, OpenBSD has become an amazing development platform:

- Functions are documented through complete man pages which often show some examples of correct and incorrect uses when it is easy to do things wrong and misuse an API. For example, the `realloc()` function is often used in a way that leads to a memory leak and the man page reflects this with a short explanation. It is common that people who are not writing code for OpenBSD still use its man pages rather than the ones provided by the system which they write code for (I know that myself and many other OpenBSD-oers do
- All of the source code is available and it can be used as a reference for many different projects and algorithms. This can also be said of other open source systems, but the strong position adopted on what code gets in makes it safer to assume that an example is correct. If the code went in, it means that at some point many people decided it was correct. Errors do happen sometimes as no system is bug free, but they are less likely
- As a means to improve code correctness, some features were implemented for OpenBSD which benefit all developments. For example, `malloc()` had changed to rely on `mmap()` and while at it enforced a strict releasing so that the assumptions that a memory chunk is still usable after being freed would no longer remain valid. The result was that applications that did a poor job at managing their memory would crash (way) more often and help people spot the bugs and fix them rather than leave them around. This produces a higher quality code and more robust applications as people who want their code to be portable to OpenBSD will eventually find out their memory related bugs as they port.

In this article I will give an overview of how you can make use of OpenBSD for both a development server and workstation. Obviously, it cannot be complete and I cannot go through all the different setups for all the different needs, this is just a way to introduce you to OpenBSD as a development platform, and make you familiar with some of the tools that can get you started. So... Here is my own setup !

Development Station

My workstation is just a plain setup with all of the tools I need to write, compile, debug and commit code to a remote server. Since I often work with other people and they do not necessarily use the same tools as I do, I tend to install popular tools so that they can grab a terminal and work without being annoyed by my own environment.

Text editors

By default, OpenBSD provides nvi, a vi variant, and mg, an emacs-like editor without all the kludge and written in C. Both can be used to write code and are actually used by many developers out there, however they are limited by design and will not provide some of the features many hackers expect

from a text editor used for programming, like syntax highlighting for instance.

While writing this article, I went polling around and it turns out that the first tool many developers install is a feature-rich text editor with support for syntax highlighting and programming modes.

The two most popular editors cited were `vim` and `emacs`.

Both of which are available as OpenBSD packages:

```
$ export PKG_PATH=ftp://insert.your/
favorite/ftp/mirror/here/

 $ sudo pkg_add vim
 $ sudo pkg_add emacs
```

Since I use emacs for coding, here is a configuration file that i'm willing to share and which helps writing readable KnF style code:

```
http://www.poolp.org/~gilles/emacs/
```

Code browser

Another useful utility is Cscope, a tool which helps developers browse code and search for references to symbols, definitions, declarations and quite a lot more. This is a very handy tool which makes it easy to browse through a large amount of code and eases the understanding of how things work in code you are not too familiar with.

Luckily, Cscope is also available as a package:

```
$ export PKG_PATH=
ftp://insert.your/favorite/ftp/mirror/
here/
$ sudo pkg_add cscope
```

Vim and Emacs both are both able to work with Cscope, and ease the browsing without having to leave the editor. I prefer to use cscope and to have it start my favorite editor through the EDITOR environment variable.

```
$ export EDITOR=emacs
$ cscope
```

Once you get hooked up, you will find it hard to stop using it.

Compilers

The system ships with compilers and interpreters for various languages. The C compilers include the well-known GCC (Gnu CC) with local extensions which aim at improving security and easing error detection in code at compile and run time. It also includes the PCC compiler that was recently imported and can already be used to build most of the OpenBSD userland.

PCC works fine but is still a work in progress and as such is not the compiler by default, however it is often a good idea to use it aside and make sure that the code that compiles under GCC does not contain and spread GCC-isms.

More compilers, including more recent versions of GCC are packaged but I do recommand you to use the versions that ship with the system unless you have a very specific need that cannot be fullfilled with these. Considering that a full operating system including kernel, userland and ports works with the default compilers, attempts at explaining why one NEEDS the latest GCC is a usual source of fun and excitement.

Debuggers

True hackers code bug-free to save time. However, true human beings fail to think of all the implications of slight changes to code, just as they can not write code for hours and hours and hours without introducing slight errors, just as coding at night increases the risks of typos, wrong arithmetics and interesting logic. OpenBSD ships with two debuggers, the full blown `gdb` for the hackers that need plenty of features and the simple `pmdb` if the bloat of `gdb` needs to be avoided.

To be honest, my use of pmdb was rather limited, and it is my understanding that it is usually used to debug kernels at an early stage of development for new architectures. However, it is interesting to know that there is a simple debugger and hopefully it can bring more people to improve it.

Versioning

OpenBSD comes with `cvs` which is the versioning tool used by developers of the project. Despite a lot of criticism from supporters of alternative version control

Listing 1. Obtaining the anoncvs shell archive

```
   $ lynx http://www.poolp.org/mirrors/OpenBSD/anoncvs.shar
Then extract it in its own directory:

   $ mkdir anoncvs
   $ mv anoncvs.shar anoncvs/
   $ cd anoncvs/; sh anoncvs.shar
   x - Makefile
   x - README
   x - anoncvssh.c
   $
```

Listing 2. Building the anoncvs shell

```
-#CVSROOT=anoncvs@anoncvs1.usa.openbsd.org:/cvs
+CVSROOT=anoncvs@cvs.poolp.org:/cvs

-BINDIR=/open
+BINDIR=/usr/local/bin

Once this is done, you can simply `make`, then `make install`:

$ make
cc -O2 -pipe    -c anoncvssh.c
cc   -o anoncvssh anoncvssh.o
$ sudo make install

install -c -s -o root -g bin  -m 4111 anoncvssh /usr/local/bin/anoncvssh
$
```

systems, `cvs` does quite a good job and encourages communication between hackers.

If you really feel the need to install another versionning utility, there is a few available as packages, including the widely used subversion:

```
$ export PKG_PATH=ftp://insert.your/
favorite/ftp/mirror/here/
$ sudo pkg_add subversion
```

The OpenBSD project has been using CVS for over 10 years and it has proven to work, which is why there is not really any interest in using alternatives. There is an ongoing project to provide a more sane CVS implementation, OpenCVS, which plans on providing compatibility with GNU CVS in a first release. OpenCVS will then work on providing new features that do not break compatibility and that improve developer experience at the same time.

Source tree

Wether you plan to work on OpenBSD related code or not, it is always a good idea to have a checkout of the system's source tree at hand.

When you do not know or have a doubt about how a programming interface works, you can bet a piece of code provides a clear and functionnal example of use.

Since you are free to reuse the code and modify it, you can even prevent having to roll a new version of something that already exists and save yourself time and bug tracking efforts.

```
$ cd /usr
$ sudo cvs -d anoncvs@your.local.mirr
or:/cvs co -P src
```

Using Cscope can turn this copy of the source tree into a large library of code samples and examples. Definitely a good tool.

Development Server

A development server can do many things, and has a different meaning depending on who sets it up. My development server provides the following:

- A repository shared between a group of coders with read-write privileges.
- Anonymous read-only access to that same repository.
- CVS log notifications through mail.

Since it is a bit trickier to setup than a few `pkg_add`, I will explain how you can achieve the same result:

Setting up CVS

OpenBSD has a shell archive, anoncvs.shar, which is available directly from one of the mirrors and which provides all we need to setup a CVS repository that can be written to by coders and read by anonymous users.

You can start by downloading the archive: see Listing 1.

anoncvssh.c, when built, is a special shell that is really a wrapper to the `cvs` utility. All it does is setup the environement for read-only access and execute `cvs`.

First, edit the Makefile to change the following lines as suits you. To increase readability, I prepended removed line with `-`, and added lines with `+`: see Listing 2.

Now, it would be too easy if that was it. The README file explains all of the steps to create the chroot jail, and to populate it with a mirror. We will follow the steps but ignore mirror stuff so that we simply have an empty repository inside the chroot jail.

I like my repositories to be accessed at `/cvs`, so we will simply create the base directory and initialize a repository named 'cvs' inside of it. When a user executes anoncvssh, he will be chrooted to the base directory and the repository can then be referenced as `/cvs`.

```
$ sudo mkdir /var/cvs
```

Then, create the anoncvs account by adding the following line to the passwd database, using the command `vipw`:

Listing 3. Setup the chroot environment:

```
      $ sudo mkdir bin dev tmp usr var etc
      $ sudo cp /bin/{cat,pwd,rm,sh} bin/

      $ sudo mknod dev/null c 2 2
      $ sudo chmod 666 dev/null

      $ sudo cp /etc/{group,hosts,passwd,protocols} etc/
      $ sudo cp /etc/{pwd.db,resolv.conf,services,ttys} etc/

      $ (cd var && sudo ln -s ../tmp tmp)
      $ sudo chmod a+rwx tmp

      $ sudo mkdir usr/{bin,lib}
      $ sudo cp /usr/bin/cvs usr/bin/

      $ sudo mkdir usr/libexec
      $ sudo cp /usr/libexec/ld.so usr/libexec/

   Finally, copy all of the libraries that ``cvs'' depends on inside the
chroot at their identical location.
   i.e: cp /usr/lib/libz.so.4.1 usr/lib/libz.so.4.1

      $ ldd /usr/bin/cvs
      /usr/bin/cvs:
         Start    End      Type Open Ref GrpRef Name
        00000000 00000000 exe  1    0   0      /usr/bin/cvs
        0a1fa000 2a202000 rlib 0    1   0      /usr/lib/libz.so.4.1
        08243000 28248000 rlib 0    1   0      /usr/lib/libgssapi.so.5.0
        0c9fb000 2ca0b000 rlib 0    1   0      /usr/lib/libkrb5.so.16.0
        086d7000 28706000 rlib 0    1   0      /usr/lib/libcrypto.so.13.0
        0f363000 2f368000 rlib 0    1   0      /usr/lib/libdes.so.9.0
        0d734000 2d768000 rlib 0    1   0      /usr/lib/libc.so.45.0
        09377000 09377000 rtld 0    1   0      /usr/libexec/ld.so
```

```
$ sudo vipw
```

Copy/paste the line:

```
"anoncvs::32766:32766::0:0:Anonymous
CVS User:/var/cvs:/usr/local/bin/
anoncvssh"
```

You may need to tweak your SSH configuration to PermitEmptyPasswords or else all attempts to log in as anoncvs will fail.

Now that the account is set, you need to setup the chroot environment. While this may look tricky it is quite simple when you understand what you're doing and you can always use the README as a reminder. Create base directory:

```
$ cd /var/cvs
```

Create a few files for the anoncvs account, you may want to edit `.profile` and `.plan` to display proper information:

```
$ sudo touch .hushlogin .profile .plan
```

Setup the chroot environment: see Listing 3.

Once this is done, edit `/etc/fstab` to make sure the `/var` filesystem doesn't have the nodev option or else things won't work too good when attempting any operation on dev/null. If it was nodev, remove the option and ... reboot.

What do we do from now ? Well, we have just created the environment to host the anonymous access but we still do not have a repository initialized !

```
$ cd /var/cvs
$ sudo cvs -d /var/cvs/cvs init
```

This is not a typo, our base directory is /var/cvs, and the repository uses cvs as its name. When accessing the repository using the anonymous account, the CVSROOT will look like this:

```
anoncvs@cvs.poolp.org:/cvs
```

It is a bit annoying because if you're not connecting as anoncvs and you do have read/write access, you will not execute the anoncvssh shell which will not chroot you and your CVSROOT will look like this:

```
anoncvs@cvs.poolp.org:/var/cvs/cvs
```

The fix is trivial ...

```
$ cd /
$ sudo ln -s /var/cvs/cvs /cvs
```

Voila, CVS repository is setup.

Setting up the accounts

At this point, we have a CVS that's installed with a repository that can be accessed read-only by the user anoncvs, but this is quite useless without a real user with write access to the repository.

How you create developers accounts is up to you, and there are as many ways to deal with this as there are administrators with creative ideas. I like to keep things simple so I make use of groups and permissions.

First, I create a group called `coders`:

```
$ sudo groupadd coders
```

Then I make myself part of the group:

```
$ sudo usermod -G coders gilles
```

Finally, I change permissions and group ownership on the repository we have created earlier so that members of the group `coders` can create modules in the repository.

```
$ sudo chgrp -R coders /var/cvs/cvs
$ sudo chmod 775 coders /var/cvs/cvs
```

Whenever we need to add a new developer, we can simply add her to `coders`, then she'll be able to commit to any module inside the repository. Also, we can restrict commit to specific modules by creating a group specific to the module, make the module group-writable for the new group and making the new developer part of that group instead of *coders*.

Mail notifications

When working with other developers, it is nice to be notified by mail when a change is made to the tree. This can be setup in a matter of minutes and only requires the setting up of an alias for sendmail and a one liner to a file in `/cvs/CVSROOT`. See Listing 4.

Sending mail to anoncvs will now send mail to everyone listed in the `/etc/mail/lists/anoncvs` file. Adding new people will only require us to execute `newaliases` so that the database is rebuilt.

Now, we need to tell CVS that it has to send mail to `anoncvs` whenever a commit is done to the repository. This is done by adding the line:

```
DEFAULT (echo ""; echo %{sVv}; cat)
| mail -s 'CVS: cvs.poolp.org' anoncvs
```

To the file `/cvs/CVSROOT/loginfo`. You can actually do notifications that are more precise and that apply to certain modules and directories, but I will let you read the header of the loginfo file which explains how this works.

There are many other things you could do depending on your need and with more or less effort. Many tools are available to browse through a web interface, create graphs and statistics, or create snapshots. The loginfo file could even be used to implement some kind of *continuous integration* bot, it is all about your needs and the ideas you come up with to solve your problems ;)

Listing 4. Creating the mailing list

```
$ sudo mkdir /etc/mail/lists/
$ sudo sh

# echo "gilles" > /etc/mail/lists/anoncvs
# echo "anoncvs: :include:/etc/mail/lists/anoncvs" >> /etc/mail/aliases
# newaliases
/etc/mail/aliases: 47 aliases, longest 52 bytes, 714 bytes total
# exit
$
```

The BSD certification
by the BSD Certification Group

Machtelt Garrels

This article is not about just any BSD certification. We will discuss the certification that is being developed by the BSD Certification Group Advisory Board.

The Advisory Board and the rest of the group consists of people who are actively involved in the different BSD projects (DragonFly BSD, FreeBSD, NetBSD and OpenBSD) – many of them are key figures in their communities and help develop their systems. The BSDCG is working with Subject Matter Experts (SMEs) and a psychometrician to ensure that both the question items and the testing method are a fair and unbiased assessment of the candidate's abilities.

Why is it important to have a *BSD certification?

- We need to break the myth that says that *BSD is offering no support.
- We need to ease and fasten adoption of BSD in business world: match companies that are using or that want to use *BSD with people who are up to the task of managing a BSD environment. There is a chicken and egg problem: people think that there is no support, so the business world does not like BSD, so there is no interest in supporting BSD.
- There is a need for (standard) objectives for training centers, course developers and publishers. A (standard) certification encourages development of course materials.
- Companies need help when hiring BSD people. To put it blunt, we need to point out for them which words to do a keyword search on in a CV.
- We need a revaluation of IT professionals: after the boom of the nineties, we now get the lash-back of the phenomenon where everybody went into IT without really knowing what they were doing. Now, IT environments are running slow and are badly managed, because most IT professionals are not up to the job. As a result, they are always busy and as a result of their busy schedule, they do not want to change, update or migrate to better solutions.

Note

We call it *BSD because we do not test any specific BSD distribution. *BSD includes all distributions of the BSD family.

There are some problems with traditional certifications that we do not want for our *BSD certification:

- Certifications are made to sell software.
- Certifications are accompanied by official course materials that examinees more or less are forced to buy. There is no free documentation, it is not freely distributable and not easy to find.
- Certifications, like software, expire in order to sell upgrades.
- Knowledge of tools is tested instead of knowledge of techniques.
- There is no input from examinees.

Value of a certification for employers

Some reports, trivially from Microsoft but also from members of more or less independent analyzing businesses, like for instance IDC, point out that employees for a UNIX-like environment on the average cost 30% more than *normal* employees. Hence they jump to the conclusion that the total cost of ownership of such an environment, which can be equipped for instance with freely available BSD software on PC hardware, is more expensive, even though it is cheaper in almost every other respect.

Note

BSD is part of the UNIX family, a collection of robust operating systems that where originally designed for big environments. Since many names of family members end in -NIX, they are sometimes called *NIX to refer to all UNICES together.

However, these reports fail to mention (on purpose?) that *NIX professionals have a much wider knowledge, while e.g.

Microsoft *professionals* tend to be niche specialists – and that you need only 1/3 of the people normally required to maintain a Microsoft environment, when you have a free *NIX environment.

Employers tend to forget that finding adequate personnel, not so much as costs, is the real problem. Somebody who knows how to do the job, somebody who can start on the job right away, rather than going through a learning period, is to be preferred by far above someone who has to learn on-the-job.

Without wanting to be an evil gossip aunt, whom would you prefer: the freshman (or worse, the would-be graduate who quit college) who installed Linux at home and who has learned everything on his/her own, or the veteran who has enough practical experience to get a certificate?

The problem with certificates, of course, is that there is no consensus. Which certificate proves that a candidate has a professional *NIX experience?

Remember not to always believe the hype. For instance, bsdcertification.com comes to mind. From their name, it is obvious enough that this is a commercial organization, and not a community-driven one.

Their last press release is from 2006, testing is for FreeBSD only, and an old version for that. Certifications for OpenBSD and NetBSD were promised, but were never created. As far as we can tell from the web site, this organization is dead.

Even though we have to deal with the little details, a BSD certification remains a good investment if you do not know yet what additional bonus you can offer your employees.

All BSD systems are focused on evolution, contrary to for instance Microsoft, which is based on revolution. BSD/UNIX competence hardly becomes outdated: you can build on it and what you learned in the past will still be valuable in ten years time from now.

Knowledge acquired is not invalidated because of new things that you have to learn now in order to survive in today's IT world. Exams become exponentially more difficult and standards are raised, guaranteeing that fiascoes like the one with the MCSE certification can not occur in our world.

Other reasons to prefer a BSD certification over a traditional one:

- It is relatively cheap.
- It is rather difficult, a good test for the candidate's experience: there are not only multiple-choice questions, but also multiple answer questions, which make it nearly impossible to pass without experience.
- BSDCG values community input and candidates can provide new questions or new objectives through regular update requests. The next update round is currently scheduled for the last quarter of 2008.
- BSDCG is vendor-independent, so there is a large item pool of exam questions and a high variation in questions. This has a positive effect on the level of difficulty of the exams.

Some people say that it is a disadvantages not to have a practical test. BUT:

- Time is limited.
- Practical tests require expensive infrastructure and the extra costs would be charged to candidates taking the exam.
- Having a practical test would almost certainly pinpoint the certification to a specific BSD distribution or version.
- We have to get rid of the idea of performance based testing and move towards performance based learning instead: learn students how to use their experience instead of learning them how to use their memory.

Pros and cons for employees

The most important reason for certification remains of course that you will acquire an extra asset when compared to that other applicant for your dream job. Especially when you just finished school or university, a certificate is a nice addition to your education. But let's be honest, among the working crowd in the BSd world, who really needs a certificate? BSD people know what they know and they do not need to prove anything to anybody, do they?

No serious BSD user or administrator has ever needed to provide prove of what he or she knows. Once you have a job and experience, the rest follows.

Another reason to take the exam, which is becoming more fashionable as we speak, is that your employer asks you to get the certificate. That is also one that is easy to understand. But if we want to find more reasons, things get harder. Maybe you could say that you want to get a certificate in order to prove your knowledge, or maybe you want to know for yourself where you stand, or you decide with a couple of friends to do a contest and see who gets the highest score.

You might also get a certificate because you are confident as to what the future will bring, or because you want to protect your career. If we believe the predictions of economic analysts, free software is going to expand dramatically during the decade to come. We are already past the file and print server phase, and well into the database or Java development platform stage, as more and more companies admit to.

You can probably name some cases of adoption right off the top of your head. Even the newspapers are telling everybody who wants to hear that free software is really making it in the business world. It is obvious that we have reached a tipping-point: there will be more free software systems, more *BSD professionals or people claiming to be so, and more incentive to divide them into *the good* and *the bad*.

If you are smart, you will make sure that when that time comes, you fall into the right category and make sure that you can show some paper.

I would have to think really hard to come up with more reasons to certify... When it comes from your own pocket, it is still an investment, however small it may be. After the boom of the nineties, wages in IT are back to normal or at least seriously reduced.

You will probably want to study a bit, too, and that takes time. Time off from work, be it with the approval of your boss, or you would have to sacrifice your own free time. And all that to prove that you can do something that you know for yourself you are capable of doing...

And then there is the risk that you don't pass, and maybe you will have to explain that mishap to your boss, who meant so well with you and sponsored your exam.

One of the less evident disadvantages of certification is that you force an upper limit onto your own competences. Imagine: Another applicant has a master level certificate, while you only have an entry level certificate because you never felt like going further. Who will be chosen for the job? The candidate who is more experienced, or the candidate who has more certificates? So once you start on a given certification path, you need to go through to the highest level that you can reach, or you run the risk to ruin your chances on the job market.

Progress report

The BSDCG did not just come up with a bunch of questions. In order to be credible, first the needs were analyzed with the help of a professional test developer (a psychometrician). She made us perform a *Job Task Analysis* (JTA), were we assembled input from many people.

That makes our certification a good one: it does not only contain the opinions of individual BSDCG Advisory Group members, it also has the input of thousands of others who expressed their opinions about the subjects to test (the exam objectives).

The initial exam, which got out of beta-testing by the end of November of 2007 and is now ready, is available in English only. During the beta-testing period, hundreds of testers with all kinds of competences took the exam. The results were then used to make a statistically valuable analysis that can be used to compare examinees.

Figure 1. bsds

The exam objectives are already translated in Mexican Spanish and Russian.

Currently, the BSDCG is focusing on the BSD Associate (BSDA) exam, which is oriented towards beginning users and administrators. Later the BSDCG plans to release a BSD Professional (BSDP) exam, which will test advanced administration skills. The details about this exam will be available by the end of 2008.

In order to bring the exam to the candidate, the BSDCG is developing a test platform which consists of a Live CD and a secured environment, lead by one or more of the proctors of our network. A proctor is somebody who has signed a Non-Disclosure Agreement and who leads the exam and makes sure candidates respect our security procedure.

We are currently looking for sponsoring and translators to make this platform available in different languages and countries. We specifically choose for this method of exam delivery, as we are on a tight budget and do not want to waste our money on commercial exam centers like Vue or Prometric. Besides, we do not want to run our test environment on MS Windows.

Until the test platform is finished, we work with paper-based exams forms. Apart from anything else, this helps us to reduce costs. We are very concerned that the certification remains accessible for everyone who wants to take the exam.

Hence the candidates' contribution is really only a small part of the total cost to publish an exam. The tests, needed for NOCA certification and thus for credibility, cost about 35.000 USD - NOCA being the quality control organization for certifications bodies. Vue and Prometric, the traditional certification bodies, charge +/- 8.000 USD per exam per language (and per version of the same exam!). We calculated that the development of our own test platform would cost about 15.000 USD. Copyrights and trademark registration would be another 4.000 USD.

As for the BSD flavors that we check for, the exam questions currently deal with FreeBSD, NetBSD, OpenBSD and DragonFlyBSD.

When tested, the candidates will be asked questions about all types of BSD systems, there is no possibility to opt for a specific distribution or version. As a consequence, we are probing for understanding, not for knowledge of details and memory capacity. Also, the BSDA is not a requirement for the BSDP.

In cooperation with the communities, we arrived at the conclusion that test objectives can be divided into 7 categories with the following weighting:

- Installation and upgrading the operating system and software: 13%.
- Securing the operating system: 11%.
- Files, file systems and disks: 15%.
- User and group management: 12%.
- Basic system administration: 12%.
- Basic network administration: 15%.
- Basic UNIX knowledge: 17%.

The BSDA exam has 100 questions covering these subjects. From the web site, you can download a command reference mapping each of the BSDA commands to the four operating systems covered by the BSDA. Furthermore, the BSDCG conceived a document describing the BSDA Certification Requirements, which can also be downloaded from the web site.

In order to gather funds, the BSDCG created a courseware DVD that gathers all the study materials from the web site. The collection consists of the exam objectives, the command reference, an explanation on our quality control mechanisms, and software and documentation for FreeBSD, OpenBSD, NetBSD and DragonFlyBSD.

Certification standards

We want our exam to be a quality test. Therefore, we apply the rules as defined by NOCA, the National Organization for Competence Assurance, which defines the standards for certification bodies.

Among other criteria, NOCA certification requires that you use psychometrics for the analysis and quality control of your exams. According to the dictionary, psychometrics is the *Mathematical analysis of psychological processes*. In other words, psychometrics

is the science that measures human variables: not only knowledge, but also practical experience. This science is also devoted to the development of tests by means of statistics.

A test is just a tool to measure the amount of Knowledge, Skills and Abilities (KSAs) that a person has in some area. It is often difficult to comprehend a quantity of knowledge, since it seems to be so abstract. But in actuality, any quantity of measurement is just an abstraction.

For instance, the measurement of height in inches, feet or meters appears on the surface to be a real and concrete measurement. But if you think about it, the inch was simply created and defined by people. There is no naturally occuring inch and there are no natural units of measurement at all. One cannot hold an inch, and it really is just an abstraction that is generally agreed upon. It is this general agreement that makes the inch a useful measurement tool. It is this common frame of reference that makes a unit of measurement functional and useful. Psychometricians do the same with exams: they create a common frame of reference that enables us to measure knowledge about a given subject.

A psychometrician has a university degree in psychology and usually additional degrees in the measurement of the human mind, in industrial psychology or in quantitative psychology. He or she is trained in the development of questions that test human features, including those features that indicate mastery of a given field of competence. A trained psychometrician is the difference between a bunch of questions and a tool that accurately measures and documents knowledge and experience. For the development of their tests, psychometricians use scientific methods to assure that the exam complies with the four rules of a good test:

- The questions are fair: no trick questions, only objective answers are possible, brain dumpers and others who do not play the game in a fair way stand no chance.
- The questions are accurate: they are updated regularly, especially in the volatile world of IT.
- The questions are clear and the wording specific, they can not be misinterpreted and all candidates can understand them without difficulties.
- The questions allow the test body to perform precise measurements of the competence of the examinees.

The psychometrician also uses scientific methods to determine the following:

- Scoring procedures: when do you get points for a good answer, and how many?
- Passing score levels: how much do you have to score in order to pass the test? Subject matter experts assist the psychometrician to determine this.
- Different versions of a test are equal: by means of statistical calculations the exam is compiled. New questions are piloted first: the answers to those questions are not scored until the validity of the question has been proved statistically, during this test phase the statistical information about the quality of the item is gathered.
- Planning of the rotation scheme, which is important for the security of an exam (again a measure against brain dumpers).

While other certifications (like RedHat and Novell) might also use psychometrics (they did not answer our questions), given the lower numbers of certified examinees, it is unsure whether the use of psychometrics is useful for them.

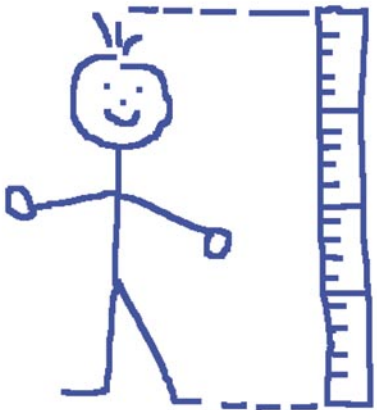


Figure 2. Metan

More information

- *http://www.bsdcertification.org*
- Mailinglist: *bsdcert@lists.nycbug.org*

Recertification

Once you get your BSDA, it will not expire. BSDP on the other hand is testing somewhat more volatile subjects. The BSDCG is as yet undecided what the recertification scheme will be for this certificate.

Summary

BSD Associate (BSDA) Certification
Language: English
Available: 2008
Re-certification: 5 years
Requirements: good knowledge of UNIX, at least 1 year of
experience on BSD systems
Domains covered:

- Installing/Upgrading OS/Hardware
- Securing the OS
- Files, Filesystems and Disks
- User and Account Management
- Basic System Administration
- Network Administration
- Basic UNIX Skills

BSD Professional (BSDP) Certification
Language: English
Available: estimated Q4/2008
Re-certification: 5 years
It is not necessary to be BSDA certified as a prerequisite.
The BSDP certification is for system administrators with extensive knowledge of UNIX and BSD Systems. Experienced system administrators of BSD systems can register for the exam directly.

Registration process:

- Get a BSDCG-ID at *http://register.bsdcertification.org/register/get-a-bsdcg-id*
- Choose an exam location
- Pay the fee by credit card or Paypal (USD 75, Eur 50).

About the Author

Machtelt Garels is in the Advisory Council of the BSD Cert Group. He gives presentations about the certification and helps promote it, among other at conferences in Berlin, Istanbul, Kopenhagen etc.

Building an OpenBSD SAMP server with content filtering proxy

Rob Somerville

In this article we will build an OpenBSD server from scratch with Squid, Apache, MySQL, PHP and Webmin (for remote management) which will allow you to serve web pages from your own network and cache the content reaching your browser.

OpenBSD is very secure, and while it does not use *bleeding edge* applications, is very stable. As a default, OpenBSD has a specially hardened version of Apache that runs in a chroot jail. This means if an attacker were to compromise the site, they would be unable to access anything outside the jail and cause considerable damage. While this is very good practice, it is down to the systems administrator to ensure that security is kept tight by not running unwanted daemons, processes or software etc.

Prerequisites

OpenBSD runs on many platforms including Intel i386 based processors and AMD 64. As the majority of people will have access to the i386 platform, this will be the basis for the server. For the test box I am using an AMD Athlon 64 bit PC with a single 15GB SCSI hard drive with 256MB of RAM and a single 100MB Ethernet card. Obviously the higher specification the better the performance and the more flexibility (e.g. to use the server to store backups etc.), so your mileage may vary depending on the hardware you have available – certainly a larger hard disk and more RAM would not be wasted. You will also need a working ADSL or cable connection to the internet via an Ethernet router, a blank CDR and a PC or laptop with a CD writer and software that is capable of writing ISO images to CDROM. Please note that a USB cable modem or a wireless internet connection is not suitable for this install. To perform the initial installation you will need a keyboard and monitor connected to the host machine, but once the machine is configured it is possible to run in in *headless* mode, that is without a keyboard and monitor.

Preparation

Preparation is the key to any successful project and we will need to perform the following actions to configure our server box (Table 1). Table 2 shows the default settings I have used for the configuration of the server. You will need to modify these to reflect your own internal network and personal requirements.

Stage 1 – Get network settings

Before we proceed, you will need to find a free IP address on your internal network and both the gateway and DNS settings. Use *ifconfig* to discover your current IP address, *route* to discover your default gateway, *ping* to discover if an IP address is in use and *dig* to discover your DNS settings.

Once you have collected the required network settings, note them down as you will need them later on in the install.

Stage 2 – Download and burn OpenBSD 4.2 boot CDROM

OpenBSD 4.2 can be downloaded via HTTP or FTP from a mirror site. To preserve bandwidth, download the ISO image from the mirror closest to you. See *http://www.openbsd.org/ftp.html* for further details. The image you will require is install42.iso and will be in the i386/4.2 directory of most mirror servers. NOTE: If you are outside the USA, do not use a USA mirror as this will contravene US law due to export restrictions. Once you have downloaded the image, you will need to burn this to CDROM using CD writer software that supports the burning of a CD ISO image. It is important that the image is written correctly, as copying the ISO image will result in a CD that will not boot. Suitable software for this purpose includes K3B on the BSD / Linux platform, and Nero Burning ROM on the Microsoft platform.

Stage 3 – Install Operating system

Insert the newly created CDROM into the CDROM of the host machine and reboot. After a short while you will be presented with the following Figure 1. After a short while, OpenBSD will boot and you will be prompted with `(I)nstall`, `(U)pgrade` or `(S)hell?`. At

this prompt press l [*ENTER*] then [ENTER] again to accept the default terminal type. For the keyboard mapping I will be using uk as I am using a UK keyboard. To see the available list of keyboard mappings, press *L* [*ENTER*] and select what is appropriate for your keyboard Figure 2.

You will be warned that OpenBSD is about to modify the contents of your hard disk. Type yes [*ENTER*] to proceed and you will be prompted for the root disk. While OpenBSD can run in dual boot configurations, this is beyond the scope of this article and we will be allocating all of the hard drive to OpenBSD. Press [*ENTER*] to accept the default configuration. You will then be asked if you wish to use all of the hard drive, answer yes [*ENTER*] to access the label editor Figure 3.

Creating the partitions and mount points

Referring to table 2, we will configure the partitions prior to formatting the hard disk. First of all, if you have partitions already installed on the disk these will have to be removed. Type *p* [*ENTER*] to view all partitions defined. If any partitions other than c: are present, delete them by pressing d [*ENTER*] followed by the partition letter until only the c: partition remains Figure 4.

To add a partition type a [*ENTER*] at the › prompt and accept the default free partition by pressing [*ENTER*]. You will be asked for the offset, press [*ENTER*] again and you will be prompted for the size. Type the partition size in Gigabytes you require (e.g. 2.5G for the root partition) and press [*ENTER*]. You will be prompted for the file system type, press [*ENTER*] to accept the default. You will then be prompted for the mount point, enter this (e.g. / for root, /tmp for tmp etc.) and press [*ENTER*] to finish the partition entry. Repeat this process for the swap, tmp, var and usr partitions but do not specify a size for the final var partition – OpenBSD will calculate the remainder for you.

NOTE: You will not be prompted for a mount point for the swap partition.

Finally type w [*ENTER*] then *q* [*ENTER*] followed by *done [ENTER*] and *yes [ENTER*] to commit the changes to disk and format the drive.

Configuring networking

You will then be asked for a short hostname and if you want to configure the network. Type your domain name and press [*ENTER*] and continue to press [*ENTER*] until you are prompted for the IPv4 address. In our test rig, this is *192.168.0.1*, but your network will probably be different from this. Type the desired IP address and type [*ENTER*] and press [*ENTER*] again to accept the default netmask if this is appropriate. When prompted for an IPv6 address press [*ENTER*] and for

Table 1. Installation steps

	Description
1	Get network settings
2	Download and burn OpenBSD 4.2 boot CDROM
3	Install operating system
4	Check networking
5	Download and install packages
6	Configure Apache, PHP, MySQL Squid and Webmin
7	Test

Listing 1. Output of ifconfig showing current IP address

```
eth0  Link encap:Ethernet  HWaddr 00:0D:61:49:7D:E1
       inet addr:192.168.0.147  Bcast:192.168.0.255  Mask:255.255.255.0
       inet6 addr: fe80::20d:61ff:fe49:7de1/64 Scope:Link

       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:29460 errors:0 dropped:0 overruns:0 frame:0

       TX packets:14026 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:37093437 (35.3 MB)  TX bytes:1104087 (1.0 MB)
       Interrupt:19 Base address:0xa000
```

Listing 2. Output of route -v showing default gateway

```
Kernel IP routeing table

Destination     Gateway         Genmask       Flags Metric Ref    Use Iface
192.168.0.0     *               255.255.255.0   U     0      0      0  eth0
link-local      *               255.255.0.0     U     1000   0      0  eth0
default         border          0.0.0.0         UG    0      0      0  eth0
```

Listing 3. Output of the ping command showing an allocated IP address and a free IP address

```
PING border (192.168.0.254) 56(84) bytes of data.
64 bytes from border.merville.intranet (192.168.0.254): icmp_seq=1 ttl=64
time=0.115 ms

....

PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
From 192.168.0.147 icmp_seq=2 Destination Host Unreachable
```

WARNING

FOLLOWING THE INSTRUCTIONS BELOW WILL RESULT IN THE TOTAL DESTRUCTION OF ALL DATA ON THE HARD DRIVE INSTALLED ON THE HOST MACHINE. ENSURE YOU HAVE AN ADEQUATE TESTED BACKUP IF YOU WANT TO RETAIN ANY DATA ON THE TARGET DRIVE OF THE HOST MACHINE.

the domain name type your domain name (in our example merville.intranet) and press [*ENTER*] to accept. Enter the IP address of your nameserver (192.168.0.254 in our example) and press [*ENTER*]. When prompted to use the nameserver, press [*ENTER*] and you will be asked for the default gateway. Enter this IP address here (in our example 192.168.0.254) and press [*ENTER*]. Press [*ENTER*] twice to accept the defaults and you will be asked for the root password. Type in test [*ENTER*] and test [*ENTER*] when prompted again Figure 5.

Installing software sets

When asked for the location of the sets, accept the default location of the CD by pressing [*ENTER*] 3 times. You will be prompted for a set name, type xbase42.tgz [*ENTER*]. The `xbase42` software set should now have a [*X*] next to it Figure 6. Type done [*ENTER*] [*ENTER*] to install the software from cdrom. Once the sets are installed, press [*ENTER*] to perform the final configuration. When prompted to use sshd press [*ENTER*], press [*ENTER*] to accept no ntp server and [*ENTER*] as you are not using X. Respond by pressing [*ENTER*] when prompted for the default console, and enter your timezone and press [*ENTER*] *to accept this option. If you are unclear as to what timezone to use, type ? [ENTER] to view a list of timezones.*

At this point we are ready to reboot, type halt [ENTER] at the prompt, and when the blue text with please press any key appears, eject the CDROM and press [ENTER]. The machine should now boot into a clean OpenBSD install.

Table 2. Default settings for the installation

Setting	TEST.MERVILLE.INTRANET Value	Recommended value
Hostname	test	Whatever you choose provided this name is not used by another server or client on your network.
Domain name	merville.intranet	The domain name of your internal network.
Network	192.168.0.0	Your network address
IP address	192.168.0.1	Any free IP address on your internal network. NOTE: Using an IP address which is is use will break your network!
Netmask	255.255.255.0	The Netmask used on your internal network
Gateway	192.168.0.254	The internal address of the router or ADSL modem on your network.
DNS	192.168.0.254	Either the internal address of your router or ADSL modem if it supports DNS lookups or your ISP's DNS server settings
Root password	test	An 8-12 character Alphanumeric password. We use test in the initial configuration and change it once we know the system is up and running.
MySQL root password	password123	An 8-12 character Alphanumeric password.
User Account	merville	A user name of your choice
User Password	testing	An 8 character Alphanumeric password.
Partition sizes		
Root (/)	2.5G	Small root partition as we will not have any user data in /home. Use a larger drive if you intend to use the server for storage and create a separate /home partition
Swap (swap)	0.5G	2 times installed memory
Tmp (/tmp)	1G	Temporary storage area cleaned at each reboot
Var (/var)	9G	Largest partition used for web server and proxy cache. the bigger the better
User (/usr)	~ 2G	Binary system files are stored here. Shouldn't need more than this unless you are install other software
Other		
Keyboard	uk	Use you country code
Timezone	GB	Use your timezone setting
OpenBSD download location	http://www.mirrorservice.org/sites/ftp.openbsd.org/pub/OpenBSD/4.2/i386/	Use your fastest local mirror
PKG location	ftp://ftp.mirrorservice.org/pub/OpenBSD/4.2/packages/i386/	Use your fastest local mirror

Stage 4 – Check Networking

Once we have configured networking and rebooted, we need to check that we have access to the internet to download the package files. Login as root with the temporary password (test), and at the shell prompt, type ping *-c3 www.google.com* [*ENTER*] and you should get a packet back from google Figure 7. Some notes on the default shell. If you type part of a command, pressing [*TAB*] will attempt to complete the command for you. For example, to change to /etc, type cd /et [*TAB*] will change the line to cd /etc.

If all is well, we can proceed to install the packages. If at this stage you cannot ping google, you will not be able install packages from the mirror site so further investigation will be required. Check your network settings are correct by typing cd /etc [*ENTER*] and typing the commands at the # prompt Figure 8. NOTE: Your network card may not be called pcn0 – look for a file in the /etc directory called hostname.xxx where xxx is your network card name. If the settings in resolv.conf or hostname.xxx are incorrect, change them by using the vi editor (vi filename). Using vi is beyond the scope of this article, but there are plenty of resources on the web to help.

Stage 5 – Download and install packages

If networking is OK, we need to set up the package source. At the prompt type:

```
export PKG_PATH=ftp://
ftp.mirrorservice.org/pub/OpenBSD/
4.2/packages/i386/ [ENTER]
pkg_add -r nano-2.0.6 [ENTER]
```

Replace the mirror site I am using with one that is closer to you to improve download speeds. If all goes well, edit the *.profile* file in the /root directory with the following command:

```
nano /root/.profile [ENTER]
```

Then add export `PKG_PATH=ftp:/xxx/` as used above at the end of the *.profile* file. This will save you having to type the export command every time you want to install software. To check this works, type exit [*ENTER*] and then login again. We will now test package downloading for the webserver etc:

Table 3. Generic commands for discovering network settings

Operating system	Commands
Linux	ifconfig, route -v, ping, dig *www.google.com*
FreeBSD	ifconfig, route get *www.google.com*, ping
Microsoft XP	ipconfig /all, ping

Listing 4. Output of dig command showing DNS server in use

```
; <<>> DiG 9.4.1-P1 <<>> www.google.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29756
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 13, ADDITIONAL: 10

;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         539223  IN      CNAME   www.l.google.com.
www.l.google.com.       281     IN      A       64.233.183.99
...
;; AUTHORITY SECTION:
com.                    34990   IN      NS      A.GTLD-SERVERS.NET.
...

;; ADDITIONAL SECTION:
A.GTLD-SERVERS.NET.     28376   IN      A       192.5.6.30

...
;; Query time: 51 msec
;; SERVER: 192.168.0.254#53(192.168.0.254)
;; WHEN: Sun Jan 20 13:46:14 2008
;; MSG SIZE  rcvd: 508
```

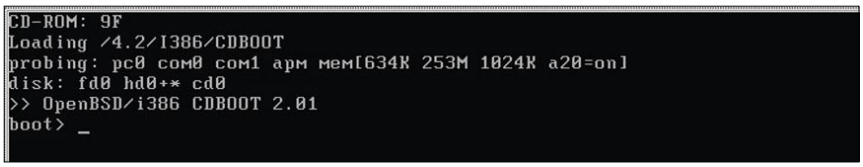


Figure 1. Instal Operating System

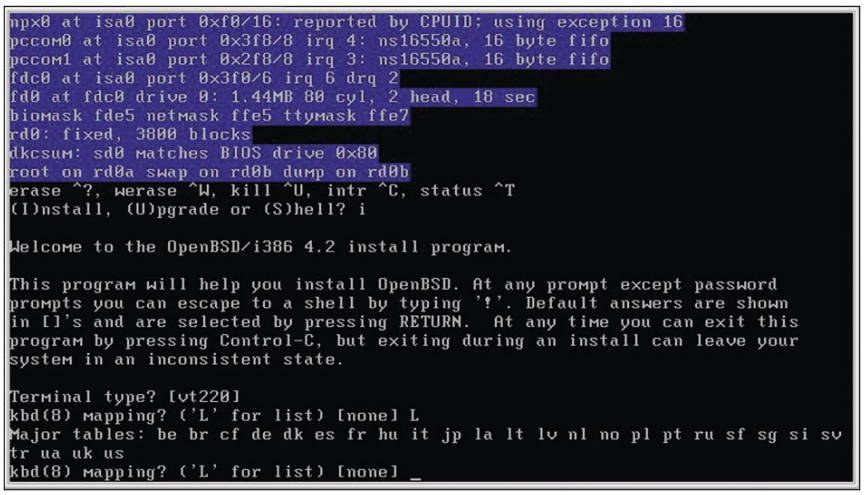


Figure 2. Keyboard mappings

Download packages

```
pkg_add -r wget-1.10.2p0 [ENTER]
pkg_add -r   squid-2.6.STABLE13
[ENTER]
pkg_add -r  mysql-server-5.0.45
[ENTER]
pkg_add – php5-core-5.2.3 [ENTER]
```

MySQL configuration

```
/usr/local/bin/mysql_install_db
[ENTER]
/usr/local/bin/mysqld_safe & [ENTER]
and after a few seconds [ENTER] again
/usr/local/bin/mysqladmin -u root
password 'password123' [ENTER]
```

```
Proceed with install? [no] yes
Cool! Let's get to it.

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

Available disks are: sd0.
Which one is the root disk? (or 'done') [sd0]
Do you want to use *all* of sd0 for OpenBSD? [no] yes
Putting all of sd0 into an active OpenBSD MBR partition (type 'A6')...done.

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

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

# Inside MBR partition 3: type A6 start 63 size 31455207
Treating sectors 63-31455270 as the OpenBSD portion of the disk.
You can use the 'b' command to change this.

Initial label editor (enter '?' for help at any prompt)
> _
```

Figure 3. Label edition

```
start of the disk, NOT the start of the OpenBSD MBR partition.

# Inside MBR partition 3: type A6 start 63 size 31455207
Treating sectors 63-31455270 as the OpenBSD portion of the disk.
You can use the 'b' command to change this.

Initial label editor (enter '?' for help at any prompt)
> p
device: /dev/rsd0c
type: SCSI
disk: SCSI disk
label: VMware Virtual S
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 255
sectors/cylinder: 16065
cylinders: 1958
total sectors: 31457280
free sectors: 31455207
rpm: 7200

16 partitions:
#             size           offset  fstype [fsize bsize  cpg]
  c:         31457280                0  unused      0     0
> _
```

Figure 4. Configoring networking

```
System hostname? (short form, e.g. 'foo') test
Configure the network? [yes]
Available interfaces are: pcn0.
Which one do you wish to initialize? (or 'done') [pcn0]
Symbolic (host) name for pcn0? [test]
The media options for pcn0 are currently
        media: Ethernet autoselect (autoselect)
Do you want to change the media options? [no]
IPv4 address for pcn0? (or 'none' or 'dhcp') 192.168.0.1
Netmask? [255.255.255.0]
IPv6 address for pcn0? (or 'rtsol' or 'none') [none]
No more interfaces to initialize.
DNS domain name? (e.g. 'bar.com') [my.domain] merville.intranet
DNS nameserver? (IP address or 'none') [none] 192.168.0.254
Use the nameserver now? [yes]
Default IPv4 route? (IPv4 address, 'dhcp' or 'none') 192.168.0.254
add net default: gateway 192.168.0.254
Edit hosts with ed? [no]
Do you want to do any manual network configuration? [no]
Password for root account? (will not echo)
Password for root account? (again)

Let's install the sets!
Location of sets? (cd disk ftp http or 'done') [cd] _
```

Figure 5. Rout Password

```
mysql -uroot -ppassword123 [ENTER]
```

This should display the MySQL prompt. Type exit [ENTER] to return Using nano or an editor of your choice, create a file /etc/rc.conf.local and add the following line:

```
MYSQL=YES
```

Stage 6 – Configure Apache, PHP, MySQL Squid and Webmin

Install Webmin

```
cd /usr/local/share [ENTER]
wget http://prdownloads.sourceforge.n
et/webadmin/webmin-1.390.tar.gz
tar -xvzf webadmin/webmin-1.390
.tar.gz [ENTER]
cd  webmin-1.390
./setup.pl [ENTER]
```

Then follow the prompts.

Configure Apache to start on boot

```
nano /etc/rc.conf [ENTER]
Change httpd_flags=NO to   httpd_
flags=""
```

Save and quit

Configure PHP and Apache

```
/usr/local/sbin/phpxs -s [ENTER]
cp /usr/local/share/examples/php5/
php.ini-recommended /var/www/conf/
php.ini [ENTER]
pkg_add -r php-mysql-5.2.3 [ENTER]
/usr/local/sbin/phpxs -a mysql [ENTER]
```

Edit `/var/www/conf/httpd.conf` and uncomment (remove the `#  from`) the following:

```
#AddType application/x-httpd-php .php
```

On the line that says DirectoryIndex `index.html` change this to read:

```
 DirectoryIndex index.html index.php
```

Configure a PHP script

Create a test script `phpinfo.php` in `/var/www/`htdocs with the following content:

```
<?php phpinfo(); ?>
```

Save and quit

Configure Squid

Create the cache:

```
squid -z [ENTER]
```

Add the below at the end of /etc/rc.c/local:

```
/usr/local/sbin/squid
```

Edit the /etc/squid.conf file:

Add this below the http_access allow manager localhost:

```
http_access allow local_network
```

Add this line below acl CONNECT method CONNECT (replace network range as required):

```
acl local_network src 192.168.0.1-192.168.0.254
```

```
        [X] comp42.tgz
        [X] man42.tgz
        [X] game42.tgz
        [ ] xbase42.tgz
        [ ] xetc42.tgz
        [ ] xshare42.tgz
        [ ] xfont42.tgz
        [ ] xserv42.tgz
Set name? (or 'done') [bsd.mp] xbase42.tgz

        [X] bsd
        [X] bsd.rd
        [ ] bsd.mp
        [X] base42.tgz
        [X] etc42.tgz
        [X] misc42.tgz
        [X] comp42.tgz
        [X] man42.tgz
        [X] game42.tgz
        [X] xbase42.tgz
        [ ] xetc42.tgz
        [ ] xshare42.tgz
        [ ] xfont42.tgz
        [ ] xserv42.tgz
Set name? (or 'done') [bsd.mp] _
```

Figure 6. xbase42 software set

```
OpenBSD/i386 (test.merville.intranet) (ttyC0)

login: root
Password:
OpenBSD 4.2 (GENERIC) #375: Tue Aug 28 10:38:44 MDT 2007

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

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

You have mail.
Terminal type? [vt220]
# ping -c3 www.google.com
PING www.l.google.com (64.233.183.99): 56 data bytes
64 bytes from 64.233.183.99: icmp_seq=0 ttl=240 time=59.175 ms
64 bytes from 64.233.183.99: icmp_seq=1 ttl=240 time=49.094 ms
64 bytes from 64.233.183.99: icmp_seq=2 ttl=240 time=68.714 ms
--- www.l.google.com ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 49.094/58.994/68.714/8.013 ms
# _
```

Figure 7. Checking Networking

```
# cat resolv.conf
lookup file bind
nameserver 192.168.0.254
# cat hostname.pcn0
inet 192.168.0.1 255.255.255.0 NONE
# route get www.google.com
   route to: nf-in-f147.google.com
destination: default
       mask: default
    gateway: border
  interface: pcn0
 if address: test
      flags: <UP,GATEWAY,DONE,STATIC>
     use  hopcount       mtu    expire
       3         0         0         0
# _
```

Figure 8. The default shell

Please remember to save before you quit.

Add default user and change root password

At the shell prompt execute the following: `adduser merville [ENTER]` and follow the prompts. When prompted to add user to other groups, add the wheel group. Change the root password to something secure:

```
passwd [ENTER]
```

And follow the prompts.

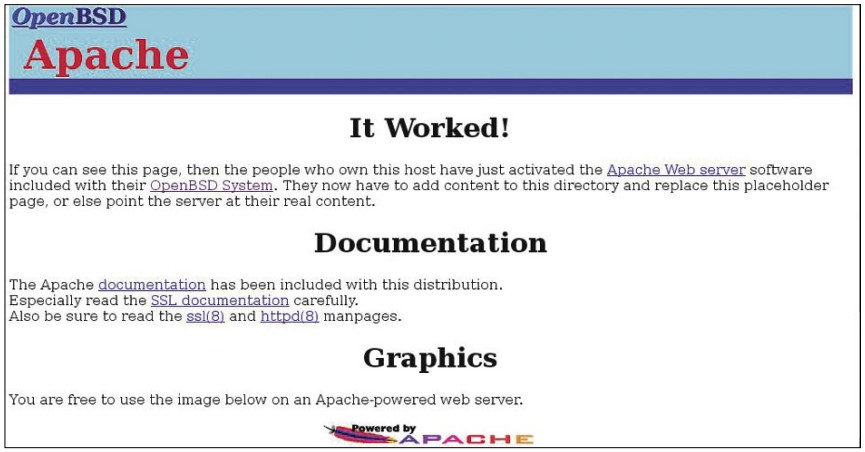


Figure 9. Apache

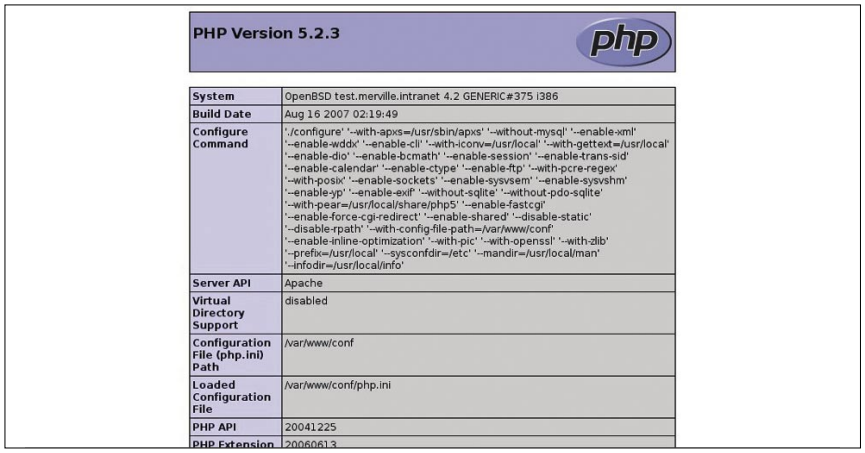


Figure 10. Apache Web Server

Figure 11. Login to Webmaster

Figure 12. Webmaster

Testing

Now reboot the machine with a *HALT* then press [*ENTER*].when prompted. First, point your browser to *http://192.168.0.1* you should see a web page similar to Figure 9.

Point your browser at *http://192.168.0.1/phpinfo.php*. You should see a web page similar to Figure 10. Point your browser at *http://192.168.0.1:10000* You should see a web page similar to Figure 11. Finally, change your proxy server settings on your browser to 192.168.0.1 using port 3128. You should be able to browse the net. Point your browser at *http://z* and you should see a screen similar to Figure 12.

Cleaning up and further improvements

This configuration, while reasonably robust requires a lot more work to be highly secure in today's internet environment. For instance, if the rig is install behind a firewall SSH, Webmin and Sendmail will not be visible to the outside world.

However, if these programs are exposed there is the possibility of an attack. Read up on security and only run processes that are absolutely vital. SSH is used for remote management and Sendmail is the default SMTP mail server. For normal day to day operations it best practice to login as a normal user then su to root.

Further reading

- http://www.openbsd.org
- *http://www.apache.org*
- *http://www.squid-cache.org*
- *http://www.mysql.com*
- *http://www.php.net*

About the Author

Rob Somerville has a keen passion for all things Open Source and has been working with software and hardware since the early Eighties. His biggest claim to fame was designing an on-line search engine for a database company long before Google or Yahoo caught the public eye. Married with 1 daughter, he shares the house with 12 computers, a cat and an extensive collection of O'Reilly books.

Open Source Days 2008

October 3rd - 4th 2008 in Copenhagen - Denmark

Will you open the source dear ?

Keynotes by

Louis Suarez-Potts

Dan Klein

Robin Rowe &
Gabrielle Pantera

Buy your conference ticket at

`http://www.opensourcedays.org/`

OpenBSD as Desktop

Petr Topiarz

This guide is intended for people who use Linux or FreeBSD and would like to give OpenBSD a try on the desktop. The guide does not claim to be en expert's advisor, so intentionally some general unix routines are also explained, while others simplified.

Many tutorials have been written on using OpenBSD as a server, however, few deal with OpenBSD as the main desktop and everyday office work and internet box. Surprisingly, that is what OpenBSD can do very well too. The jump from 4.1 release to 4.2 was great for Gnome users, as Gnome has been updated from 2.10 to 2.18. The new 4.3 release has besides the update of Gnome 2.18 to 2.20 brought a lot of useful packages especially in printing area, e.g. Gutenprint or HPLIP has been introduced and Firefox and Thunderbird updated too. For the coming release, Ekiga is in the ports for telephony and the KDE users can finally enjoy the advantage of K3B for burning CDs. So overall the improvements are huge.

However in this article we are going to see more practical information on how to make life with an OpenBSD desktop really easy. Let's start with the basics. We will add a group, user, mount devices, deal with the network and set up a printer.

Adding a group is basic if more people login to the PC, so that they can share documents:

```
$ groupadd -g 1200 friends
```

creates a group with id number 1200 and name *friends* and the following:

```
$ useradd -u 500 -g friends -G wheel,operator -k /etc/
skel -s /bin/sh -d /home/caroline -m caroline
```

creates a user *caroline* as a member of friends and with administrative power (`wheel,operator`). Interesting is that with a `-d` switch you can identify a different home directory than the default. Another practical stuff is to omit `-m` if your home directory already exists.

Similarly, you can add other users. Of course, change the `-u` number and `-d` directory. e.g.:

```
$ useradd -u 501 -g friends -G wheel,operator -s /bin/sh
-d /mnt/usb/my_data peter
```

Now to set a password and allow people to login you need to:

```
$ passwd caroline
```

which will ask you for the password and then for repeating it. Noticeable thing is that the system, for security reasons, does not show anything while you write the password, it does not even print stars or other cryptic symbols, however it accepts your typing.

Now we will allow Caroline to access usb and cdrom devices. First we need to create mounting points

```
$ mkdir /mnt/usb /mnt/cdrom
```

then we have to dedicate these to caroline

```
$ chown caroline /mnt/usb /mnt/cdrom
```

similarly we have to adjust the permitions in `/dev`

```
$ chmod 660 /dev/sd0i /dev/cd0a
```

and now we are going to send the information about mounting points to `/etc/fstab`

```
$ echo "/dev/sd0i /mnt/usb msdos rw,nodev,noexec,nosuid,n
oauto 0 0 " >> /etc/fstab
```

and

```
$ echo "/dev/cd0a /mnt/cdrom cd9660 ro,nodev,noexec,nosui
d,noauto 0 0" >> /etc/fstab
```

and finally there is one and last change, we need to inform the kernel about our idea to let users mount devices. So we write to the configuration:

```
$ echo "kern.usermount=1
   # enable user mounting devices" >>
/etc/sysctl
```

and if you want to try that immediatelly:

```
$ sysctl -w kern.usermount=1
```

now feel free to plug in a usb stick and you do not have to be a root to type:

```
$ mount /mnt/usb
```

and you are there!

To make the nice feeling more complete, KDE, Gnome, and Rox environments will allow you to mount these devices just by clicking at the mounting points, which makes it even more fun. Now our Caroline can login, mount CD, or USB. So, how do we start the graphics? We need to make a configuration file where we tell the system which graphical environment we would like to use. Let's be very spoiled:

Make sure you are in your home directory:

```
$ cd /home/caroline
```

create the config file

```
$ touch .xinitrc
```

This will ensure, that our settings is valid both for graphical login and the black ugly command line:

```
$ ln -s .xinitrc .xsession
```

Now the command to start kde session:

```
$ echo " exec startkde " >> .xinitrc
```

And finally we can happily type:

```
$ startx
```

and if things go well we will enjoy a nice environment, almost like in the spoiled Linux distros of today.

If you want to enable graphical login by default, go to `/etc/rc.conf` and change `xdm_flags=NO` to `xdm_flags=''`. For a really nice look this would need a little tuning, but basically it will work. Maybe you will argue, that KDE environment is not part of the OpenBSD release, of course not, so lets add it:

```
$ export PKG_PATH=ftp://
ftp.openbsd.org/pub/OpenBSD/4.3/
packages/i386
```

as you see I am expecting you to run a regular simple PC, so change the architecture at the end of the line if you have PPC or AMD64

`$ pkg_add -v` kdebase kdemultimedia mozilla-firefox mozilla-thunderbird amarok gwenview

Ok, I have made a random choice, but you can similarly add many more, you will find them at *http://openports.se/* which is a very clever web interface providing detailed info about packages and their sources called ports. A very important point here is to read the post install messages and do what they instruct you to do. Advices are simple and exact.

Now having such a nice graphical environment it would be a shame to be without a network.

```
$ ifconfig -a
```

will show you the interfaces, among them, for example the ethernet `fxp0` or wireless `wi0` will appear. `$ dhclient fxp0` will connect you to the net if you have the line plugged in and the net is not blocked.

If you want it after every start of the system, we need to write a configuration:

```
$ touch /etc/hostname.fxp0
$ echo " dhcp NONE NONE NONE " >>
/etc/hostname.fxp0
```

If you come to a place with wireless network you can enjoy the advantage of the genius simplicity of OpenBSD.

```
$ ifconfig wi0 up
$ ifconfig -M wi0
```

will provide you with a list of networks around, you can pick one

```
$ ifconfig wi0 nwid CoffeShopNetwork
$ dhclient wi0
```

and there you go... Well unless the network is marked private, then you need the password, in our case *Jimmy*, and then:

```
$ ifconfig wi0 nwid CoffeShopNetwork
nwkey Jimmy
$ dhclient wi0
```

and there you go really straight to the internet! In case you want your PC to remember this setting, then

```
$ touch /etc/hostname.wi0
$ echo " dhcp nwid CoffeShopNetwork
nwkey Jimmy " >> /etc/hostname.wi0
```

and after the restart, if the network is running, you will automagically connect to it.

OpenBSD is definitely a leader in using wireless technologies and allows you to use cards such as those with Prism Intersil, Asus or TNETW chipsets with their native drivers. They have been reversely engineered by geeks and experts to avoid using the Windows' driver with ndiswrapper, as Linux and Freebsd or Netbsd tend to do. Now the last thing that you need to have running on a laptop or a desktop is printing. That used to be a real issue earlier, however with the latest releases of OpenBSD it is merely fun. Just add few packages and configure it with a web interface:

```
$ pkg_add -v cups foomatic-db
foomatic-filters ghostscript hplip
```

that should be enough for most usual printers, then you enable and start the cups print-server:

```
$ /usr/local/sbin/cups-enable
$ /usr/local/sbin/cupsd
```

fire up your web browser and type:

```
http://127.0.0.1:631/
```

which will bring you to a very friendly web interface, that allows you to add a printer or configure the print-server to share printers.

An OpenBSD machine can also run JAVA, FLASH plugin, play realplayer streaming, and emulate Linux environment, but that would need a little more time to describe. If you are interested and cannot wait, then I strongly recommend *http://www.openbsd101.com/* and *http://www.softwareinreview.com/bsd_tutorials/using_openbsd_4.2.html* as well as the famous *http://www.onlamp.com/* server, where you can learn a lot of wisdom from the real BSD gurus.

Inside the PBI system...

Svetoslav P. Chukov

PBI stands for PC-BSD Installer. It is a unique and very useful package management system. If you are familiar with other systems, you will notice some similarities and also some differences that make it unique.

If one just clicks on the *.pbi* file, an automated installer appears and offers to guide the user through the installation process. On the whole this is probably like a wizard that helps people install the program, and I would say it is very successful in that task.

Features

Every operating system is based on some small parts of software that create the whole solid foundation of the OS. So, I would say that these small pieces in the GNU/Linux world are the packages, but it is interesting how this question is answered in the PC-BSD world. What are these parts that PC-BSD is built on? The packages of FreeBSD and of the PBI. They create the system and everything that lies on it; libraries, applications and all other data need to be used. For successful integration of an operating system into the market, the OS needs to be designed for that market. So, a server OS is designed with the main goals of being secure and stable, a desktop OS is designed to be user-friendly and easy to use. Everything is built for and aims to be used in a particular target market. PC-BSD itself is a desktop-oriented OS. Yes, and I would say that it really achieves that goal pretty well. But this is not all. The basics of its success are mostly because of the great system called the PC-BSD installer. What exactly is so great in PBI? What makes it special for installing software? The answer: its ease of use and its simplicity. The basic reason why I think PBI is so successful is that it contains all the data it needs to install the application. So, if the application needs library X, then the installer should contain that library.In the installation process it should extract and prepare that library to work, and that makes the application work properly.

This design concept solves an entire pool of problems and troubles with package dependencies and inconsistencies. One other big plus for PBI is the support for advanced scripting. That is a very huge plus for it.

PBI offers:

- A completely graphical installation in step-by-step style.
- Scripting support - a really powerful feature that makes PBI not only an executable installer, but an installer that can *think*.
- A check for package integrity.
- Icon Management - this allows developers to set icons for both the desktop and the K-Menu.
- Error Detection, if something goes wrong with the installation
- Easy installation and un-installation. There is a utility to do this in the graphical environment, but a command line tool is also available.

Understand what is inside

Basically, the front side of PBI is a visible-to-the-user, nice, user-friendly graphical interface, but the engine under the hood is nothing more than FreeBSD packages. Yes, PBI is something like an upgrade of FreeBSD packages, and it adds additional functionality. So, instead of being just a binary package that should be extracted to result in useful files, PBI consists of several parts that empower the plain packages with extra features. And these extra features make the PCBSD installer flexible and scalable. What I want to do is to show you what exactly is inside a PBI package, how it works, how it processes data, and how it decides to do this instead of that. After this article you will be able to understand what actually is a PBI package and the magic inside it.We will start with the setup script. The purpose of the setup script is to setup the first actions and configurations of the subsequent operations. And, the next-executed script sets up the environment and

options for work. All the pre-tasks need to be done, and the control of the installation is taken up by the next script. Basically this is a very comfortable model, because of the modularity of the scripts and processes. One could separate some tasks to different scripts easily, without any of them interfering with each other.

Probably at this point you have some questions, like, scripts? In PBI? What scripts?. Yes, exactly, what scripts? The scripts that manage the installation and the un-installation processes, and the scripts that prepare the processes and make it possible for a simple package installer to interact with the user.

OK. PBI is not just a plain archive, but it is much more than that. It is a package with binary data and executable scripts that do some work. Basically the structure is very simple: the scripts are executed in the installation and the un-installation processes, and they handle all the necessary tasks. So, let's take a closer look at this. I assume for our example that work is to be done on the gFTP. This isacomfortable GTK+ ftp client without many requirements for libraries and resources. So, it is suitable for our use.

When one starts installing a particular .pbi file, a nice, user-friendly, wizard-like installer appears. So, the next step is to specify the folder where the program files are to be put, and then the actual work of installation begins. As you may notice, it is clearly a simple procedure with two or three clicks on the *Next* button. But behind these three clicks a huge amount of work is done. There are pre- and post-installation scripts that need to be explained at this point.

These are the `PBI.SetupScript.sh` and `PBI.FirstRun.sh`.

The `PBI.FirstRun.sh` script will run before the program is extracted into the target directory. I understand that you can not wait to see what is behind this script, and how it works to benefit the whole PCBSD installer. Let's proceed deeper...;See Listing 1, which is the code fragment of the `PBI.FirstRun.sh`

As you may see, this script contains logic that makes the decision whether to be installed or not to be installed. This example is pretty simple and understandable. Now I would say

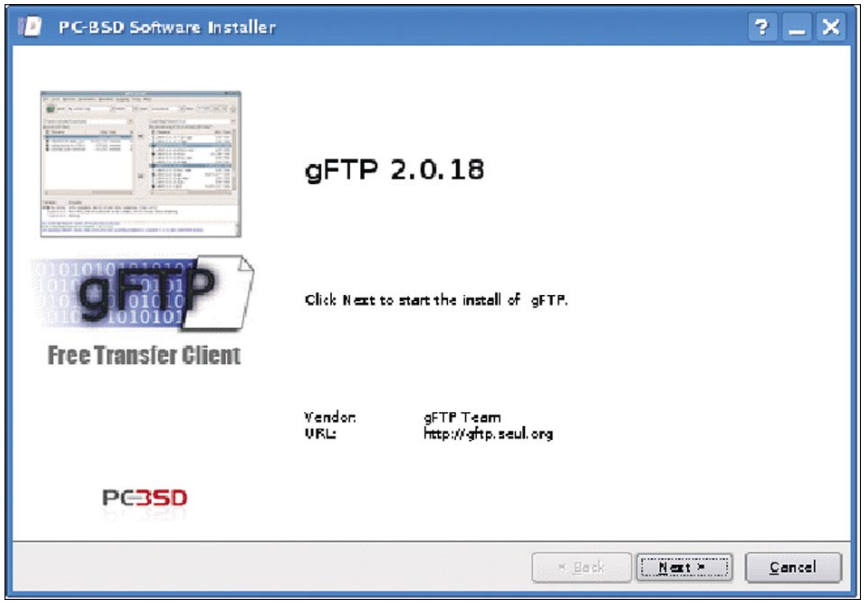


Figure 1. Start the PC-BSD installer

Listing 1. Code fragment of the PBI.FirstRun.sh

```
#!/bin/sh

if [ -e "/usr/local/bin/gftp" ]
then
  # Looks like FF is installed, ask if they want to remove the old one
  ls -al /usr/local/bin/gftp | grep Programs 2>/dev/null
  if [ "$?" = "0" ]
  then
   kdialog --yesno "gFTP is already installed, do you wish to uninstall
it?"
   if [ "$?" = "0" ]
   then
    FF="`ls -al /usr/local/bin/gftp | cut -d '>' -f 2 | cut -d "/" -f 3`"
    echo $FF | grep gFTP 2>/dev/null
    if [ "$?" = "0" ]
    then
     PBIdelete -remove ${FF}
    else
     kdialog --sorry "gFTP could not be automatically removed... Please
remove it in Add / Remove Programs and try again."
     return 2
    fi
   else
    kdialog --sorry "gFTP is already installed, it must be uninstalled
before loading this PBI"
    return 2
   fi

  else
   # Could not find a link to PBI folder
   kdialog --sorry "gFTP is already installed, it must be uninstalled
before loading this PBI"
   return 2
  fi
fi
```

Listing 2. Code fragment of PBI.SetupScript.sh

```
ln -s /Programs/${PROGDIR}/.sbin/gftp /usr/local/bin/gftp
ln -s /Programs/${PROGDIR}/.sbin/gftp-gtk /usr/local/bin/gftp-gtk
ln -s /Programs/${PROGDIR}/.sbin/gftp-text /usr/local/bin/gftp-text
ln -s /Programs/${PROGDIR}/man/man1/gftp.1.gz /usr/local/man/man1/gftp.1.gz
ln -s /Programs/${PROGDIR}/share/gftp /usr/local/share/gftp
sed 's:prefix=/usr/local:prefix=/Programs/gFTP2.0.18:g' /Programs/${PROGDIR}/bin/gftp > tempfile && mv -- tempfile
/Programs/${PROGDIR}/bin/gftp
# Copy over all the LANG files
LANGFILE="gftp.mo"
cd /Programs/${PROGDIR}/locale
for i in `ls`
do
 mkdir -p /usr/local/share/locale/${i}/LC_MESSAGES >/dev/null 2>/dev/null
 cp /Programs/${PROGDIR}/locale/${i}/${LANGFILE} /usr/local/share/locale/${i}/LC_MESSAGES/${LANGFILE}
done
chmod +x /Programs/${PROGDIR}/bin/gftp
echo "LAUNCHCLOSE: /usr/local/bin/gftp"
```

Listing 3. Uninstall with PBI.RemoveScript.sh and PBI.RemoveScript2.sh

```
#!/bin/sh
if [ -e '/Programs/gFTP2.0.18/PBI.RemoveScript2.sh' ]
then
  sh /Programs/gFTP2.0.18/PBI.RemoveScript2.sh "${@}"
fi
rm -rf '/Programs/gFTP2.0.18'
#!/bin/sh
rm -fR /usr/local/bin/gftp-gtk
rm -fR /usr/local/bin/gftp-text
rm -fR /usr/local/bin/gftp
rm -fR /usr/local/man/man1/gftp.1.gz
rm -fR /usr/local/share/gftp
# Remove the old locale files since we are uninstalling
LANGFILE="gftp.mo"
cd /Programs/${PROGDIR}/locale
for i in `ls`
do
 rm /usr/local/share/locale/${i}/LC_MESSAGES/${LANGFILE}
done
if [ ! -z "$DISPLAY" ]
then
 # Ask if we want to remove the user profiles
 kdialog --yesno "Do you want to remove gFTP user settings?" --title "Remove user settings"
 if [ "$?" = "0" ]
 then
 cd /home
 for i in `ls`
 do
  if [ -e "/home/${i}/.gftp" ]
  then
  rm -rf /home/${i}/.gftp
  fi
 done
 fi
fi
```

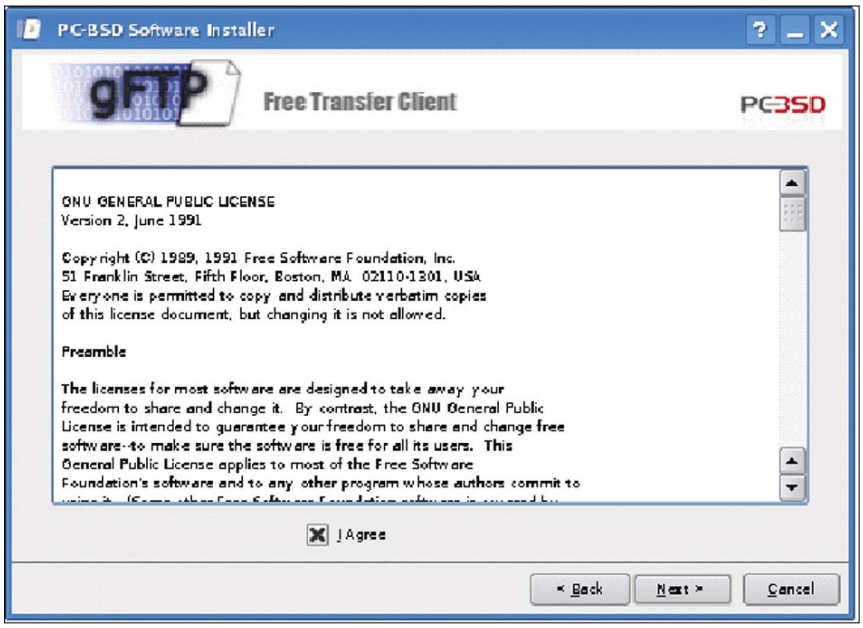


Figure 2. Agree to the terms of the license

how surprised I was when I saw for the first time these lines of code. I expected to see more universal code with limited opportunities for user interaction. Instead, I saw code that is highly manual and modular. In fact, this is a SHELL script, and it is based on command-line tools and applications, but it also uses a GUI application to interact with the user. And of course the script could be altered to any new form you want. Since it is a SHELL script, you could execute another tool, an application, an interactive shell, or even another script, if you wish. That gives the feature of being easily extensible. So, basically this script aims to do all the steps needed. Is PBI already installed or not? Is there an older or newer version? Are there perhaps corrupted files in the target directory? These are some of the questions this script should answer.

OK. We have the FirstRun script, but what about the second run? What about the script that does the work after the package has been extracted? That is the place for the `PBI.SetupScript.sh`.

It handles additional questions like: How to set up the application and its libraries? How to put the data files so they are visible to the application? What about local files? Or new language support? So, after the package has been extracted into the target directory, this script takes on the job of setting up and solving all these questions. Of course this executable file is a SHELL script and that gives us more oppor-tunities for work. I like the SHELL, it is well known to the users of all the UNIX-like systems, and such a script could be easily changed to match new criteria or goals.

Let's see now how actually this executable works. For that reason take a closer look at the code itself. (See Listing 2.)

This code just creates symbolic links between the real files at the /Programs folder and the folders in the $PATH at `/usr/local`. After that, other additional data files needed for the application's work are prepared. So, basically the target directory is the directory that contains the files visible in the $PATH

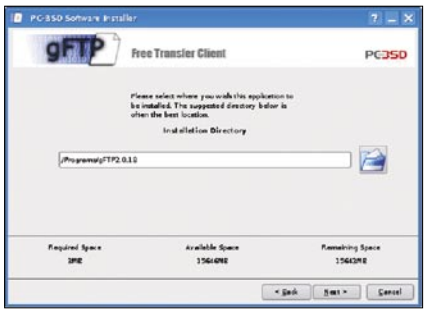


Figure 3. Choose installation directory

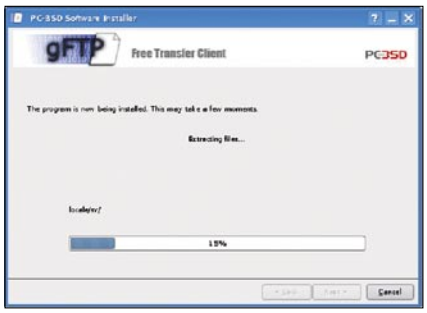


Figure 4. Installation

Figure 5. Sit back and relax while the extracting of the files is done

variable, and thatmakes them visible to the system. Every file needed by the application should be visible to the system and copied or linked to the folders in the `$PATH`.

That was for the installation tasks; let's see how the un-installation works.

The remove actions are taken at un-install time by the script PBI.RemoveScript.sh. Basically, it runs the next following script, called `PBI.RemoveScript2.sh`. (See Listing 3.)

The result

You have now seen an example of the PC-BSD installer from the inside, and you could imagine how all the parts of the installer work together. Now I will show you what is the result of this. It is really pleasant. The first time I saw PBI, I did not understand how it works, but after a few hours of in-depth experience with PC-BSD, I got it. So, here are the actions of PBI shown on screenshots: see Figures 1-5.

Summary

I would say PBI is a wonderful way to manage an application, and I really liked it because of the way it works. In the world of Unix the ordinary method of installingan application or data on your computer is via packages. Many systems use many different package managers, but I was so impressed by PBI because of the step-by-step-and-you-are-done style in which it works. I am a technically oriented person, but I always appreciate good and valuable solutions that make my work easier. And PBI is one of them. It makes software management tasks moreunderstandable to the user.

Connecting
to Other IM networks

Eric Schnoebelen,
Michele Cranmer

You have taken the plunge, you have adopted Jabber as your instant messaging system of choice. How do you keep in contact with all your poor, un-enlightened friends who are still using the proprietary *walled garden* networks?

The solution is designed into Jabber/XMPP. The solution is *transports*, a mechanism to allow the creation of a gateway between the jabber network and closed/proprietary networks, such as *Yahoo! Instant Messenger*, *Microsoft Live Messenger*, AOL *Instant Messenger*, ICQ, China's QQ, *FaceBook*, Poland's *GaduGadu* and other networks. GTalk is not listed as GTalk is XMPP-based and already federating, all it takes to converse with friends on GTalk is to add them to your roster.

Unfortunately, you can't connect to the *Walled Garden* networks without having an identity on those networks. However, using the transports, you can use your favorite Jabber client, and connect to all the networks you have identities on. All the roster information is contained in one location, on your jabber server.

Why not use a multi-protocol client?

Multi-protocol clients have to focus on supporting lots of protocols, and probably don't support them all as well they might. Jabber-only clients support Jabber extremely well, and leave the supporting of the other protocols to the transport, installed on the jabber server. Granted, the Jabber transports probably wouldn't be nearly as good as they are, if it weren't for the people working on reverse engineering the proprietary protocols for the multi-protocol clients.

Lets build some transports

We are going to look at building the following transports and configuring them to work with the jabberd2 server we have been configuring. The transports are for AOL, MSN and Yahoo!.

The transports we're building are all written in Python. The AOL and MSN transports use the Twisted framework while the Yahoo! transport uses the `xmpp.py` framework. Obligatory pkgsrc recommendation: I have packaged all of pyAIMt, PyMSNt and YIMt in `pkgsrc-wip` on Source Forge. The packages are `py-jabber-aim-t`, `py-jabber-msnt`, and `py-jabber-yahoo-transport`.

Changing into the appropriate directory, and typing `[b]make install` will download, will build and install the packages and all their dependencies. FreeBSD has pyAIMt and pyMSNt in the ports collection as net-im/jabber-pyaim and `net-im/jabber-pymsn`. *Ok*, now for building things the *hard way*.

Listing 1. Needed Packages and where to find them

```
Twisted-2.5.0
 http://tmrc.mit.edu/mirror/twisted/Twisted/2.5/

pyOpenSSL-0.6
 http://dl.sourceforge.net/sourceforge/pyopenssl/

Imaging-1.1.6
 http://effbot.org/downloads/

dnspython-1.6.0
 http://www.dnspython.org/kits/1.6.0/

xmpppy-0.4.1
yahoo-transport-0.4
 http://dl.sourceforge.net/sourceforge/xmpppy/

pyaim-t-0.8a
 http://pyaimt.googlecode.com/files/

pymsnt-0.11.3
         http://delx.net.au/projects/pymsnt/tarballs/
```

Listing 2. Packages and their dependencies

```
pyaim-t-0.8a
 Twisted-2.5.0
 Imagine-1-1.6
 pyOpenSSL-0.6

pymsnt-0.11.3
 Twisted-2.5.0
 Imagine-1-1.6
 pyOpenSSL-0.6

yahoo-transport-0.4
 xmppy-0.4.1
 dnspython-1.6.0
```

Listing 3. Working pyaim-t configuration from jabber.cirr.com

```
<pyaimt>
    <!-- The JabberID of the transport. -->
    <jid>aim.jabber.cirr.com</jid>

    <!-- The JabberID of the conference room handler.
-->
    <!-- GROUPCHAT IS NOT STABLE YET -->
    <confjid>chat.aim.jabber.cirr.com</confjid>

    <!-- The component JID of the transport. Unless
you're doing
      - clustering, leave this alone -->
    <!-- <compjid>aim1</compjid> -->

    <!-- The location of the spool directory.. if
relative, relative to -->
    <!-- the src dir.  Do not include the jid of the
transport. -->
    <spooldir>/var/spool/jabberd</spooldir>

    <!-- The location of the PID file. if relative,
relative to the src dir. -->
    <!-- Comment out if you do not want a PID file -->
    <pid>/var/run/jabberd/pyaimt.pid</pid>

    <!-- The IP address of the
 main Jabber server -->
 <mainServer>jabber.cirr.com</mainServer>

 <!-- The JID of the main Jabber server -->
 <mainServerJID>jabber.cirr.com</mainServerJID>

 <!-- The website of the Jabber service -->
 <website>http://jabber.cirr.com/</website>

 <!-- The TCP port to connect to the Jabber server on
-->
 <!-- (this is the default for Jabberd2) -->
 <port>5347</port>

 <!-- The TCP port that the web admin interface will
answer on -->
 <!-- (uncomment to enable) -->
 <!-- <webport>12345</webport> -->

 <!-- The authentication token to use when connecting
to
 - the Jabber server -->
 <secret>***************</secret>

 <!-- The authentication token to use when connection
to
      - the web interface -->
    <websecret>letmein</websecret>

    <!-- The default language to use (for error/status
messages) -->
    <lang>en</lang>

    <!-- The hostname of the AOL login server you wish
to connect to -->
    <aimServer>login.oscar.aol.com</aimServer>

    <!-- The port of the AOL server you wish to
connect to -->
    <aimPort>5190</aimPort>

    <!-- Send message on successful registration -->
    <registerMessage>You have successfully registered
with PyAIMt</registerMessage>

    <!-- You can choose which users you wish to have
as administrators.
      - These users can perform some tasks with Ad-Hoc
commands that
      - others cannot -->
    <admins>
        <jid>eric@jabber.cirr.com</jid>
    </admins>


    <!-- You can select which event loop PyAIMt will
use. It's probably
      - safe to leave this as the default -->

    <!-- Use epoll for high-load Linux servers running
kernel 2.6 or above -->
    <!--<reactor>epoll</reactor>-->

    <!-- Use kqueue for high-load FreeBSD servers -->
    <!--<reactor>kqueue</reactor>-->

    <!-- Use poll for high-load Unix servers -->
    <reactor>poll</reactor>

</pyaimt>
```

Dependencies

The first, and probably biggest dependency is python itself. I am assuming you have already gotten python built and installed. `pyaim-t` and pymsnt require the python Twisted framework, at least 2.5 or later. The `Twisted -core`, `-zopeinterface`, `-web`, and `-words` sub-modules of Twisted are required. All are part of the Twisted-2.5.0 archive. The python OpenSSL (pyopenssl) module is also required. If you want to support avatars, the Python Imaging module is also required. Yahoo transport requires `xmpppy`, dnspython, and python expat, which is sometimes optional portion of the python distribution.

Build and install all the modules using the standard python mechanism of changing in the package directory, and executing:

Listing 4. Working pymsn-t configuration file from jabber.cirr.com

```
<pymsnt>
 <!-- This file contains options to be configured by the
server
 - administrator. -->
 <!-- Please read through all the options in this file
-->

 <!-- The JabberID of the transport -->
 <jid>msn.jabber.cirr.com</jid>
 <!-- The public IP or DNS name of the machine the
transport is
 - running on -->
 <host>msn.jabber.cirr.com</host>

 <!-- The location of the spool directory.. if
relative, relative
 - to the PyMSNt dir. Do not include the jid of the
transport -->
 <spooldir>/var/spool/jabberd/</spooldir>

 <!-- The location of the PID file, relative to the
PyMSNt directory -->
 <pid>/var/run/jabberd/pymsnt.pid</pid>
 <!-- If set, the transport will background itself
when run -->
 <background/>

 <!-- The IP address of the main Jabber server to
connect to -->
 <mainServer>jabber.cirr.com</mainServer>
 <!-- The TCP port to connect to the Jabber server on
(this is
      - the default for Jabberd2) -->
    <port>5347</port>
    <!-- The authentication token to use when
connecting to the
      - Jabber server -->
    <secret>***************</secret>

    <!-- The default language to use -->
    <lang>en</lang>
    <!-- The website of the Jabber service -->
    <website>http://jabber.cirr.com</website>

    <!-- Comment out the following options to disable
them, or
      - uncomment them to enable them -->
    <!-- Send email notification messages to users -->
    <mailNotifications/>
    <!-- Send greeting on login -->
    <sessionGreeting>
        You have just started a session with PyMSNt
    </sessionGreeting>
    <!-- Send message on successful registration -->
    <registerMessage>
        You have successfully registered with PyMSNt
    </registerMessage>
    <!-- Allow users to register with this transport
-->
    <allowRegister/>
    <!-- Get all avatars. If this is set to true then
avatars are
      - grabbed for all your contacts immediately. If
false then avatars
      - are only grabbed when you're in a chat with a
contact -->
    <getAllAvatars/>
    <!-- File transfer settings -->
    <!-- The maximum size of a file transfer (in
bytes). For
      - unlimited, comment out, or set to 0 -->
    <ftSizeLimit>524288</ftSizeLimit>
    <!-- The maximum rate for file transfer (in bytes).
For unlimited,
      - comment out, or set to 0 -->
    <ftRateLimit>2048</ftRateLimit>
    <!-- You can choose which users you wish to have
as administrators.
      - These users can perform some tasks with Ad-Hoc
commands that
      - others cannot -->
    <admins>
        <jid>eric@jabber.cirr.com</jid>
    </admins>
    <!-- Log settings -->
    <!-- The logging level
        0 -> No logging
        1 -> Log tracebacks
        2 -> Log tracebacks, warnings and errors
        3 -> Log everything -->
    <debugLevel>2</debugLevel>

    <!-- The file to log to. Leave this disabled for
stdout -->
    <debugFile>/var/log/jabberd/pymsnt.debug</
debugFile>

</pymsnt>
```

Listing 5. Working yahoo-transport configuration file from jabber.cirr.com

```
<?xml version="1.0" ?>
<pyyimt>
    <!-- This file contains options to be configured by the server
         administrator. -->
    <!-- Please read through all the options in this file -->

    <!-- The JabberID of the transport -->
    <jid>yahoo.jabber.cirr.com</jid>

    <!-- The JabberID of the conference room handler. -->
    <confjid>chat.yahoo.jabber.cirr.com</confjid>

    <!-- The location of the spool file.. if relative, relative to the
         PyYIMt dir. -->
    <!-- Include the jid of the transport, if running multiple copies of
         the same transport -->


    <spoolFile>/var/spool/jabberd/yahoo</spoolFile>

    <!-- The location of the PID file, relative to the PyYIMt directory -->
    <!-- Comment out if you do not want a PID file -->
    <pid>/var/run/jabberd/yahoo-transport.pid</pid>

    <!-- The IP address or DNS name of the main Jabber server -->
    <mainServer>127.0.0.1</mainServer>

    <!-- The JID of the main Jabber server -->
    <mainServerJID>jabber.cirr.com</mainServerJID>

    <!-- The TCP port to connect to the Jabber server on
         (this is the default for Jabberd2) -->
    <port>5347</port>

    <!-- The authentication token to use when connecting to the
         Jabber server -->
    <secret>***************</secret>

    <!-- Allow users to register with this transport -->
    <allowRegister/>

    <!-- Allow users to use the Yahoo! chat rooms with this transport -->
    <enableChatrooms/>

    <!-- You can choose which users you wish to have as administrators.
         These users can perform some tasks with Ad-Hoc commands that
         others cannot -->
    <admins>
        <jid>eric@jabber.cirr.com</jid>
    </admins>

    <!-- The file to log to. Leave this disabled for stdout only -->
    <debugFile>/var/log/jabberd/yahoo-transport.log</debugFile>
</pyyimt>
```

- python setup.py build
- sudo python setup.py install

Build the transports

Once all the prerequisites are built, now its time to move on to building/installing the transports themselves.

All of the transports are meant to be executed out of the extraction directory, so choose well. (The packages in pkgsrc-wip have been modified to install into a common tree, and execute there.) Thus, there is not a lot to do for building.

Configuring

All the transports use XML files for configuration, and use many of the same tags. We will start our configurations with `pyaim-t`. Change into the directory pyaim-t-0.8a and start by copying `config_example.xml` to `config.xml`. Now fire up your favorite editor on `config.xml`.

The most interesting fields to be checked and modified are: `<jid>` the `id/name` of the transport. Usually something like aim.jabber.`<domain name>` If you want off site users to be able to use your AIM transport, this name needs to exist in DNS.

- `<mainServer>` – the IP address of the jabber server
- `<mainServerJID>` – the DNS listed hostname of the jabber server
- `<secret>` – The shared secret between pyaimt and the
- jabber server (router component.)

Changing those elements will get you up and running.. Reviewing the rest of the elements may be interesting, but not essential. Configuring pymsnt is essentially identical. The example configuration file is called `config-example.xml`. Copy it to `config.xml`, and edit the `<jid>`, `<mainServer>` and `<secret>` elements to suit. Again, if you want the transport to be usable by people on other jabber servers, make sure the name specified in `<jid>` is listed in DNS.

The last transport to configure is yahoo-transport. For the yahoo-transport, the example configuration file is `config_example.xml`, and is expected to be `config.xml` in the application start up directory.

Again, the interesting elements are `<jid>`, `<mainServer>`, `<mainServerJID>`, and `<secret>`. Setting `<confjid>`, along with `<enableChatrooms/>` will set up the gateway into the Yahoo conference rooms. Once again, the name given in `<jid>` (and `<confjid>` if you want conference rooms) must be resolvable in DNS if you want off site jabber servers to be able to use it.

Starting the transports

Ok, we've got them built, and we've got them configured, hopefully. Now it is time to start the servers. Each of them was designed to run out of their source directories.

First up, make sure the user you've chosen to run the servers has write permissions in the program directories. All of the transports store their spool files and directories as sub-directories of the current directory (unless modified by the configuration file).

So, as the user you are going to run the transports as, iteratively change into each directory, and start the transport. For pyaim-t and pymsnt, it is PyAIMt.py and PyMSNt.py respectively. For yahoo-transport, it is *yahoo.py*.

`PyAIMt.py` will go into the background (become a daemon) if you specify the `-b` or `--background'` flags.

```
./PyAIMt.py -b
```

Will fire up the AIM transport. Check your log files for errors if the background program ends unexpectedly. `PyMSNt.py` acts the same as `PyAIMt.py`. Change into it is directory, and start it with the `-b` flag to make it act like a daemon. pymsnt also supports an XML element of `<background/>` to have the transport start as a daemon.

yahoo-transport is a bit different, in that it has to be explicitly be put in the background, as follows:

```
./yahoo.py &
```

Using the transports

To make use of your newly installed transports, browse your local server from your jabber client.

In Psi, right click on your account name, and select *Service Discovery* from the pop-up menu. Your newly installed transports should show up as children of the server.

To register, select the registration function in the appropriate fashion (in Psi, double clicking will do it) and then fill in the registration dialog. You will need to fill in your legacy network username and password.

Once you have registered, all of your contacts on the legacy system should start showing up in your jabber roster. Warning, you may be asked to Add/Auth a lot of users, the entire contents of your legacy system roster. Do not worry, your contacts on the legacy system won't see anything. And you will only have the annoyance once.

Congratulations, you have successfully built the transports, and used them to connect to the legacy systems. Now you can do all your instant messaging through your jabber server and your jabber client. And your frends on the legacy systems won't know the difference I have been doing just that for over 3 years.

Now, it's up to you to start enouraging them to migrate to an open-standards messaging system, XMPP/Jabber.

In the coming issues, we'll talk about setting up conference room services, file transfer proxies, and an overview of several popular Jabber capable clients. If you have any ideas for future articles, please send them to *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 prefered OS is NetBSD, which he has running on Alpha, UltraSPARC, SPARC, amd64 and i386.

Michele Cranmer is a relativity new user to UNIX and Jabber, having been basically forced into learning it when she met Eric. After having been a loyal Windows and Yahoo Messenger user for many years, she finds that she prefers the *new* systems to the others because of ease of use and reliability. Being a college student, getting her degree in Special Education, she plans on using the *new* systems in her classroom as a way of teaching the children that there are many different ways to do things other then the ``normal'' ways and those ways are no more strange or unusual then they are.

great subscriber offer

SAVE $20!

Get your copy of BSD Magazine and save $20 of the shop price

Three easy ways to order

- *visit: www.buyitpress.com/en*
- *call: 001 917 338 3631*
- *fill in the form below and post it*

Why subscribe?

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

BSD Magazine ORDER FORM

☐ **Yes**, I'd like to subscribe to
BSD Magazine from issue ☐ ☐ ☐ ☐
1 2 3 4

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 29.99€

I understand that I will receive 4 issues over the next 12 months.
Credit card:
☐ Master Card ☐ Visa ☐ JCB ☐ POLCARD
☐ DINERS CLUB

Card no. ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐
Expiry date ☐☐☐☐ Issue number ☐☐
Security number ☐☐☐

☐ I pay by transfer: Nordea Bank
IBAN: PL 49144012990000000005233698
SWIFT: NDEAPLP2

Cheque:
☐ I enclose a cheque for $ _____
(made payable to Software-Wydawnictwo Sp. z o.o.)

Signed

Terms and conditions:
Your subscription will start with the next available issue.
You will receive 4 issues a year.

* if you already are Software-Wydawnictwo Sp. z o.o. client, write your client's ID number, if not, fill in the chart above
** I enable Software-Wydawnictwo Sp. z o.o. to make an invoice

Kernel File System
Development in Userspace

Antti Kantee

As a programming and testing environment, the kernel is immensely more challenging than userspace. Therefore, kernel code is typically tested and developed in the comfort of userspace before undertaking the trial by fire in the kernel.

Previously, specially written glue code was required to make it possible to run the kernel code in userspace, but now the NetBSD *Runnable Userspace Meta Program* (`rump`) framework enables to run unmodified kernel file system code out-of-the-box in userspace and with seamless integration. It can be thought of as being a generalized superset of the functionality provided by Sun's ZFS libzpool userspace testing library.

After the developed code is dropped into the kernel, bugs are usually found in specific use cases and the code must be debugged in the kernel environment. Anyone who has ever done kernel debugging knows that it is far from the most trivial and enjoyable task in the world. As the debugging session more often than not leads to a kernel panic, two different environments are a common approach: one for running the kernel being debugged and another one for controlling the previous. There are multiple classic ways of accomplishing this: two physical machines, an emulator, or a userspace operating system.

The three ways listed above are fundamentally the same thing. Creating an alternate environment and using that for debugging. There are two common problems with this approach.

- Not enough isolation. The implementation under development still runs in the same kernel environment as the system that hosts it. For example, error path testing is difficult by introducing errors to common routines such as the buffer cache and disk drivers, since extra care must be taken to make portions of the kernel that are not under development (e.g. the root file system) not suffer from fault injection.
- Too much isolation. Repeating a bug often depends on a specific machine and application configuration. For instance, it might require a big application such as OpenOffice or Firefox, or downloading and saving a file from some specific ftp site. This environment needs to be recreated in the test setup before the problem can be repeated.

This article is a tutorial for file system development using the *Runnable Userspace Meta Program* (`rump`) facility found in NetBSD. In addition to explaining the necessary steps in a practical hands-on manner, a brief introduction of the involved technology is given.

Technology overview

There are two different technologies involved in running kernel file systems in userspace.

- Pass-to-Userspace Framework File System or puffs. puffs is the NetBSD mechanism for implementing file systems in userspace. The idea is similar to the Linux FUSE, but the interface is different and mimics the BSD file systems kernel interface enabling a more natural implementation in the kernel. puffs receives requests in the kernel, transports them to the userspace file server, waits for a result and passes it back to the caller.
- Runnable Userspace Meta Programs or rump. File systems implemented in the kernel are free to call any kernel routines. The rump shim layer makes sure these routines are available in userspace. For the most part, the routines are directly compiled from kernel source modules. Examples of these types of routines are the buffer cache routines and virtual file system subroutines. Some parts, however, must be reimplemented for userspace. Examples in the later category are the disk device driver and virtual memory subsystem code.

There are two basic choices for running kernel file system code in userspace. These are both presented in see Figure 1, in addition to a regular in-kernel file system architecture being given for comparison.

- The case with a mounted file system shows what the configuration looks like when running a kernel file system in userspace with complete application transparency. The requests are passed from the kernel to userspace and back using puffs and translated from the puffs protocol to the kernel vfs/vop interface using a helper library called `p2k` (puffs-to-kernel).
- The standalone case invokes file system operations directly. This avoids kernel involvement, but requires specially written applications against a library called ukfs (user-kernel file system). The advantage in this approach is that the application is completely disjointed from the the host kernel features, the only exceptions being a handful of common system calls such as `read()/write()`. This means that NetBSD kernel file system code can be run on virtually any platform. The ukfs interface is discussed at lenth later in this article.

File locations

All rump source code is located in the NetBSD-current source tree under `src/sys/rump`. It will be present in NetBSD 5.0 when it is released. This document is written against the status present in NetBSD-current at the end of May 2008.

The shim library is under `src/sys/rump/librump`. The kernel file systems are build as libraries under `src/sys/rump/fs/lib` while the file server binaries themselves are located in `src/sys/rump/fs/bin`. For example the efs file system's kernel portion is built into `src/sys/rump/fs/lib/efs` and the file server binary is found from `src/sys/rump/fs/bin/efs`. None of the built binaries are currently installed anywhere, so they must be run directly from the source tree.

Adding a new mountable file system: a walkthough

To add a new file server to the rump build, the kernel portion of the file server must first be built as a regular userspace library. The only difference from a normal program library is that the compilation flags used for building this library are that of the kernel. Most of the necessary steps are already automatically handled by the build framework. The user should fill in the library name, source file path, and source modules to be compiled. An example of this for the efs file system is presented in see Listing 1.

A directory called libyourfs should be created under `src/sys/rump/fs/lib` with the only content being the Makefile described above.

Additionally it might be necessary to specify file system specific compilation flags for the library. This may be done as with any other library. The following example is from libffs:

```
CPPFLAGS+=      -DFFS_NO_SNAPSHOT -
DFFS_EI
CFLAGS+=        -Wno-pointer-sign
```

Next, the file server executable for mounting the file system is required. The server daemon implementation is effectively just a matter of filling out the file system argument structure and calling `p2k` library run routine. The file system arguments depend on the file system in question, but for our efs example it is simply a matter of filling out the location of the file system image to be mounted. As the server daemon assumes this path is passed as the first parameter to the program, the following does the trick:

```
struct efs_args args;
memset(&args, 0, sizeof(args));
args.fspec = argv[0];
```

Calling the `p2k` library run routine mounts the file system and jumps to a main loop, which takes care of processing requests. The routine's signature is `p2k_run_fs` (`fs_type`, `devpath`, `mountpath`, `mountflags`, `fs_args`, `fs_args_size`, `puffs_flags`). As our example, efs is used once again: see Listing 2.

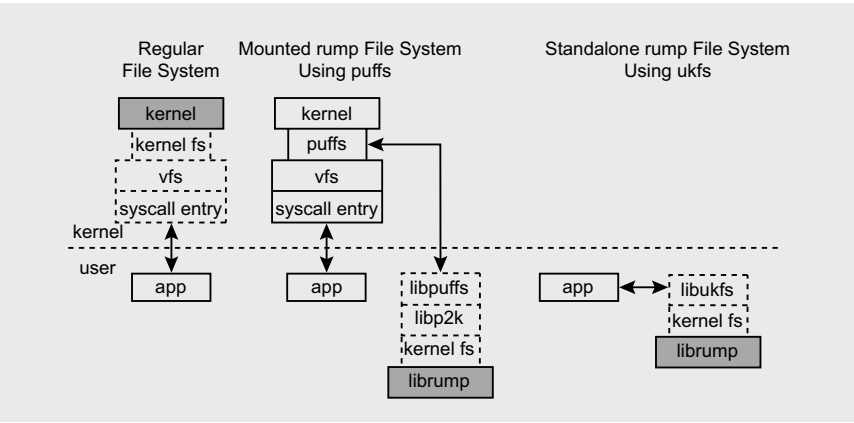


Figure 1. Kernel file system

Listing 1. Kernel fs library Makefile

```
# -      $NetBSD: Makefile,v 1.2 2007/08/07 10:16:57 pooka Exp $
#
.include <bsd.own.mk>
LIB=    efs
.PATH:  ${NETBSDSRCDIR}/sys/fs/efs
SRCS=   efs_genfs.c efs_ihash.c efs_subr.c efs_vfsops.c efs_vnops.c
.include <bsd.lib.mk>
.include <bsd.klinks.mk>
```

Listing 2. p2k_run_fs() in efs

```
   rv = p2k_run_fs(MOUNT_EFS, argv[0], argv[1], mntflags | MNT_RDONLY,
       &args, sizeof(args), pflags);
   if (rv)
      err(1, "mount");
```

The mntflags and pflags variables have been parsed earlier from command line arguments. As the kernel efs implementation is currently read-only, the readonly flag is forced. The `p2k_run_fs()` routine returns only after a fatal error or when the file system is unmounted. Unmounting can be done the normal way using umount(8), or the violent way by killing the file server. To build the file system daemon, a similar Makefile as building the kernel portion library is required. Most of the work is once again handled by existing build infrastructure magic. The daemon code should be located in `src/sys/rump/fs/bin` in a directory called yourfs. The Makefile looks a lot like a standard program BSD Makefile, with the exception that the kernel file system library gets linked in. The pathmagic for this is handled automatically by the rump build framework. See Listing 3 for an example.

Finally, the build system must be told that your file system exists. This is done by adding yourfs to the RUMPFSLIST variable in Makefile.rumpfs in the directory `src/sys/rump/fs`. Currently the relevant line looks like this:

```
RUMPFSLIST= cd9660fs efs ext2fs ffs
hfs lfs msdosfs ntfs syspuffs tmpfs
udf
```

After this, rebuild everything by typing `make` in the rump main directory. If all the above steps were done properly and rump supports all the functionality your file system uses, there will be an executable called yourfs in the object directory of `src/sys/rump/fs/bin/yourfs`. This executable can be run to mount the file system:

```
./efs ~/img/efs.img /puffs
```

As inferred by the previous example, an additional advantage of using rump is that there is no need to vnconfig file system images: they can be directly mounted as files. In case of accessing a device directly, it is recommended that the raw device is used, e.g. `/dev/rwd1e`. In case the block device node (e.g. `/dev/wd1e`) is used, all access goes through the buffer cache. Since the buffer cache is fairly small in size, this can negatively effect the performance of all other file systems on the system in case heavy file I/O is performed. The buffer cache is used by file systems only for metadata while file contents are stored in the page cache. Therefore the buffer cache is of limited size. Block device node access goes entirely through the buffer cache, therefore caching also file contents in the buffer cache. For large files, this can quickly flush everything else from the buffer cache.

After mounting it is possible to use the file system just like a regular kernel file system: see Listing 4.

Listing 3. File system server Makefile

```
#       $NetBSD: Makefile,v 1.1 2007/08/05 22:28:02 pooka Exp $
#
PROG=           efs
LDADD+=         ${RUMPFSLD_EFS}
DPADD+=         ${RUMPFSDP_EFS}
.include <bsd.prog.mk>
```

Listing 4. rump file system in mount lists

```
golem> mount | grep efs
/home/pooka/img/efs.img on /puffs type puffs|p2k|efs (read-only, nosuid,
nodev, mounted by pooka)
golem> df /puffs
Filesystem              1K-blocks       Used      Avail %Cap Mounted on
/home/pooka/img/efs.img     16214       9161       7053  56% /puffs
golem> ls /puffs
WorkSpace  debug      etc        lost+found unix
bin        dev        floppy     stand      usr
cdrom      dumpster   lib        tmp        usr2
golem>
```

The only difference to an in-kernel file system is that the file system image is being accessed in the comformt and safety of userspace.

Debugging Mounted File Systems in Userspace

All of the regular userspace debugging tricks apply to rump file systems. It is possible to single step, send signals, dump core, attach a debugger, ktrace, profile, stop and continue, add printfs, and do iterative development very quickly.

Dealing with "kernel" panics

A kernel panic in a rump file system is merely a core dump. It can be loaded into gdb like from any other userspace program and the stack backtrace and other state at the point of panic can be examined. The example below shows what happened when trying to mount a slightly corrupted FAT file system image:

```
golem> ./msdosfs ~/img/msdosfs.img
/mnt
panic: buf mem pool index 23
Abort (core dumped)
golem>
```

After examining the core dump it became clear which field caused the error. A check for a bad value added to the mount routine and now mounting of the image is politely refused instead of causing a kernel panic.

Single stepping

Single stepping rump file systems while being executed is easy, since pausing the file system does not pause the entire kernel. Only applications accessing the file system will be frozen for the duration of the debugging operation. For example, if one would like to trace/debug the execution of the ufs lookup routine, one could do the following: see Listing 5.

In addition to the small teaser presented above, the regular gdb tricks of course apply. A useful thing to note from the stack backtrace is that vnode operations go through `RUMP_VOP_OP()` instead of `VOP_OP()` as in the kernel. The former can be used to place a breakpoint for a certain operation regardless of the type of file system being debugged.

There is one catch. Since NetBSD currently has problems debugging threaded programs, as a workaround for

attaching a debugger you must compile rump so that it does not use threads. This can be done by making sure the following is set in `src/sys/rump/librump/Makefile.inc`:

```
CPPFLAGS+=    -DRUMP_WITHOUT_THREADS
```

This disables thread support completely. This means that file systems which create threads can no longer be run. It also means that system threads such as the vnode release thread will not be started. For development operations besides live program debugging, it is recommended that rump is compiled with this option commented out to better emulate a proper kernel environment. Notably, there is no problem in NetBSD with debugging core dumps created by threaded programs.

Creating code dumps

Sometimes it is useful to add clauses to the code to force a code dump if some complex set of rules it met. This can be done simply with if (conditional) panic(*hit condition*);. Taking a core dump of an already running file server is sometimes required. The standard methods of using gcore(1) to generate a live core or kill -ABRT for terminating the program and creating a core have often been found useful.

Direct access to file system code

The examples discussed so far mount the file system as part of the host system. If we recall, this means that accessing them requires control to flow through the kernel by making system calls. Accessing file system routines directly is done directly from ukfs without passing through the kernel. It can be used for developing utilities such as mtools and NetBSD makefs(8) by directly employing the kernel fs code and not requiring a separate userspace implemention. In addition, it allows the writing fine-grained test programs and the stress-test of file system code much more efficiently. Test functionality similar to Sun's ZFS ztest utility could also be written using ukfs, with the exception that it does not need to be limited to just one file system.

The main documentation for the ukfs library is currently available only in the form of a header in `src/sys/rump/fs/lib/libukfs/ukfs.h`. However, most of the routines resemble system calls, so it is easy to figure out what each ukfs call does. Calls typically take the file system context structure (struct ukfs *), a pathname, and whatever arguments are necessary. For instance:

```
ukfs_rmdir(ukfs, dirpath)
```

removes the directory dirpath, while:

```
ukfs_read(ukfs, filename, off, buf,
bufsize)
```

will read at most bufsize bytes into buf from the file filename from offset off.

To use the ukfs library, two initialization routines must be called. `ukfs_init()` initializes the global process state required for using ukfs and rump. After this, the desired file system must be mounted using `ukfs_mount(fs_type, devpath, mountpath, mountflags, fs_args, fs_args_size)`. The parameters are the same as for `p2k_run_fs()` described earlier. The mount routine returns the context structure to be passed to interface routines.

All pathnames given to the library can be relative or absolute. The current directory can be changed by calling the `ukfs_chdir()` routine. The current directory is per thread, so in case the process using ukfs has multiple threads, each thread is initialized with the current directory as the root directory and must be explicitly changed if desired.

Further information

Documentation, technical papers and examples of use for puffs and rump can be found from the NetBSD website:

- *http://www.NetBSD.org/docs/puffs/*
- *http://www.NetBSD.org/docs/puffs/rump.html*

Listing 5. using gdb on ffs

```
golem> gdb ffs
GNU gdb 6.5
[...]
This GDB was configured as "i386--netbsdelf"...
(gdb) break ufs_lookup
Breakpoint 1 at 0x80697bc: file /usr/allsrc/src/sys/ufs/ufs/ufs_
lookup.c, line 115.
(gdb) run -o ro ~/img/ffs.img /puffs
Starting program: /objs/obj/sys/rump/fs/bin/ffs/ffs -o ro ~/img/ffs.img
/puffs
rump warning: threads not enabled, not starting vrele thread
rump warning: threads not enabled, not starting namecache g/c thread

[meanwhile, cause a lookup to happen from another window]

Breakpoint 1, ufs_lookup (v=0xbfbfd1a0)
    at /usr/allsrc/src/sys/ufs/ufs/ufs_lookup.c:115

115             struct vop_lookup_args /* {
(gdb) n

120             struct vnode *vdp = ap->a_dvp;
(gdb) bt
#0  ufs_lookup (v=0xbfbfd1a0) at /usr/allsrc/src/sys/ufs/ufs/ufs_
lookup.c:115
#1  0x0807ac38 in RUMP_VOP_LOOKUP (dvp=0x8148d00, vpp=0xbfbfd1ec,
    cnp=0x80b2a20) at rumpvnode_if.c:132
#2  0x0806f725 in p2k_node_lookup (pu=0x80b7200, opc=0x8148d00,
    pni=0xbfbfd290, pcn=0xbfbfd27c) at p2k.c:327
#3  0x08076d7c in dispatch (pcc=0x80aea20) at dispatcher.c:277
[etc.]
```

Securing IM
using Jabber/XMPP and TLS

Eric Schnoebelen,
Michele Cranmer

XMPP/Jabber offers a number of features that make it different from the commercial, closed messaging systems. This month, we'll talk how to secure client to server and server to server communications.

Are your private communications vi instant messaging really as private as you think they are? This month, we will talk how to secure client to server and server to server communications.

Have you ever been chatting with a friend or family member on one of the big instant messaging services, and wondered who else might be seeing your conversation? Well, the truth is...it could be anyone! The major IM services seem to lack the mechanism for securing the communications between the client and server.

Would not you rather use a service that you operate and know is secure? One where you do not have to worry about if the things you say to your Mother about your ex, will be read by someone who knows them? That is what Jabber can give you! The security in knowing that what you chat about will be between you and the person/people you are chatting with.

In the last issue we showed you how to set up a Jabber/XMPP server, using the open source *jabberd2* server. This time we will talk about how to secure communications through that system. One of the features Jabber/XMPP offers that makes it different from the proprietary, commercial IM services is the ability to secure client to server and server to server communications. Secure server to server communications is an important feature of XMPP, and the XMPP Foundation has a goal of having most of the interconnecting (federating) jabber servers using secure channels by Jabber's 10th anniversary, 4 Jan 2009 (see *https://stpeter.im/?p=2136*).

Securing communications

First up, we are going to discuss securing communications between your Jabber/XMPP server and a client. We are going to use the jabberd2 server we built/installed last time. (although, since then versions up to 2.1.24.1 have been released, and 2.2.0 was released during the writing of this article.)

You can use either a self-signed certificate for securing your jabber server, or you can use a commercial certificate. The XMPP Foundation (*http://www.xmpp.net*) has set up an agreement with Startcom to provide every Jabber server operator with a certificate signed by a known signing authority.

We will go through the common steps for generating both a commercially signed certificate and a self-signed certificate, as they are common for most of the tasks.

Creating the certificate signing request

Some of the signing authorities, such as the one offered by *xmpp.net*, offer a web form to create the certificate signing request.

Other signing authorities will require you to create your own certificate signing request. If you are creating a self-signed certificate, you will need to create a signing request as well. XMPP certificates require a bit of additional information not required for the more common HTTP/SSL certificate signing request.

Listing 1 shows the changes/additions needed to your OpenSSL configuration file (`/etc/openssl/openssl.cnf` on NetBSD) to get the extra OID's needed for XMPP's use. (this listing can be found at *http://wiki.jabber.org/index.php/XMPP_Server_Certificates*).

Listing 2 shows the OpenSSL configuration file I used to generate signing certificates and self-signed certificates for jabber.cirr.com (along with my test jabber server, *portnoy.cirr.com*.)

Creating a self-signed certificate

Creating a self-signed certificate is fairly straight forward for anyone who has done it for web servers. Here is the command line I used:

```
openssl req -x509 -nodes -days 365 \
-config /etc/openssl/xmpp.cnf -newkey
rsa:1024 \
-keyout portnoy.cirr.com.key -out
portnoy.cirr.com.pem
```

Before installing on the jabber server, make sure to concatenate the `.key` file onto the `.pem` file.

Getting a certificate from xmpp.net

To receive a certificate from xmpp.net, you will have to register with *xmpp.net*. Follow the registration directions at *https://www.xmpp.net/account-request*.

There are two mechanisms for receiving a certificate from *xmpp.net*. The first is to use the web site to create your private key, your certificate signing request, and finally your certificate.

The second is to create your own key and signing request, and submitting it to the XMPP CA for the creation of the request.

The first two screens on both processes are the same. The first screen is selecting the request type, either letting the CA create the request, or providing your own. Select as appropriate.

The second screen is providing contact information. A street address must be provided (post office boxes are not acceptable.) The phone number provided must reverse look up to the street address provided.

Now the processes diverge.

XMPP CA generated CSR

When letting the XMPP CA generate the certificate signing, the third screen in the process will request a pass-phrase for use on your key. It must be between 10 and 32 characters long, using mixed case alphabetic letters and the digits.

The forth screen presents the private key that was generated. Copy it from the text box, and record it somewhere. Also remember to record the pass-phrase to this private key. Select *continue* to move to the next screen.

On the fifth screen, the information required for your certificate signing request will be collected. The information is your country, your `state/province`, your `city/town/locality`, you organization, and finally the hostname of the jabber server. The *top level domain* is available as a pull down.

Select *continue*, and the sixth screen appears, requesting the email address to receive the validation request, and presents the certificate signing request generated. You should save the certificate signing request.

Skip down to *Validating the request* for the rest of the process.

Self Generated CSR

When generating the CSR yourself, you can use Listing 2 as the start of a configuration file to generate your certificate signing request. Make sure the *Common Name* is the fully qualified domain name of the jabber server, as presented in the DNS SRV record or A record.

Listing 1. Lifted from http://wiki.jabber.org/index.php/XMPP_Server_Certificates

```
oid_section             = new_oids

[ new_oids ]

# RFC 3920 section 5.1.1 defines this OID

xmppAddr = 1.3.6.1.5.5.7.8.5

[ req ]

default_bits            = 1024
default_keyfile        = dotat.key
distinguished_name      = distinguished_name
req_extensions          = v3_extensions
x509_extensions         = v3_extensions

# don't ask about the DN
prompt = no

[ distinguished_name ]

countryName             = GB
stateOrProvinceName     = England
localityName            = Cambridge
organizationName        = dotat labs

commonName              = dotat.at

[ v3_extensions ]

# for certificate requests (req_extensions)
# and self-signed certificates (x509_extensions)

basicConstraints        = CA:FALSE
keyUsage                = digitalSignature,keyEncipherment
subjectAltName          = @subject_alternative_name

[ subject_alternative_name ]

DNS.0                   = dotat.at
otherName.0             = xmppAddr;UTF8:dotat.at

# Append the following for a server which handles multiple domain names:

DNS.1                   = example.org
otherName.1             = xmppAddr;UTF8:example.org
```

The following openssl command line will generate the request:

```
openssl req -new -nodes -config /etc/
openssl/xmpp.cnf \
-newkey rsa:1024 -keyout
portnoy.cirr.com.key \
-out portnoy.cirr.com.csr
```

After the certificate signing request has been generated, paste it into the text box on the XMPP CA form, and submit it. The next screen will ask for the domain administrative email (`hostmaster@`, `postmaster@`, or `webmaster@`) to receive the validation token. Patiently await it is delivery to the respective mailbox.

Listing 2. Openssl_conf

```
openssl_conf            = openssl_init
[openssl_init]
oid_section             = new_oids

[ new_oids ]

# RFC 3920 section 5.1.1 defines this OID
xmppAddr                = 1.3.6.1.5.5.7.8.5

[ req ]

default_bits            = 1024
default_keyfile        = privkey.pem
distinguished_name      = distinguished_name
req_extensions          = v3_extensions
x509_extensions         = v3_extensions
prompt = no

[ distinguished_name ]
countryName                     = US
stateOrProvinceName             = Texas
localityName                    = Plano
organizationName                = Central Iowa (Model) Railroad
commonName                      = jabber.cirr.com

[ v3_extensions ]

# for certificate requests (req_extensions)
# and self-signed certificates (x509_extensions)

basicConstraints        = CA:FALSE
keyUsage                = digitalSignature,keyEncipherment
# subjectAltName                = @subj_alt_name
subjectAltName=DNS.1:cirr.com,otherName.1:xmppAddr;UTF8:cirr.com,\
        dirName.1:distinguished_name,\
        DNS.0:jabber.cirr.com,otherName.0:xmppAddr;UTF8:jabber.cirr.com,\
        dirName.0:distinguished_name,\
        DNS.2:portnoy.cirr.com,otherName.2:xmppAddr;UTF8:
portnoy.cirr.com,\
        dirName.2:distinguished_name
```

Listing 3. jabber.cirr.com

```
    <id realm='jabber.cirr.com'
        password-change='true'
        pemfile='/etc/openssl/certs/jabber.cirr.com.pem'
        register-enable='false'>jabber.cirr.com</id>
```

Validating the request

At this point the two certificate generation paths converge.

Once the token arrives, enter it into the the field on the screen, and submit the form.

The final screen will appear with your certificate. Copy it from the web page, and also save the Certificate Authority intermediate certificates as well. Once you have got all the certificates, chaining certificates and keys, your key needs to have the pass-phrase removed (unless you want to enter your pass-phrase every time one of the component start). To remove your pass-phrase, use an openssl command line similar to the following: (replace the key file names with your key file names):

```
openssl rsa -in jabber.cirr.com.key \
-out jabber.cirr.com.key-no-passprhase
```

The certificate, chaining certificates, and your key now need to be concatenated into one large file, with the elements in the following order:

- Your certificate
- The intermediate certificate authority chain certificates
- Your key.

as an example:

```
cat jabber.cirr.com.crt
sub.class1.xmpp.ca \
jabber.cirr.com.key-no-passphrase >
jabber.cirr.com.pem
```

Congratulations, at this point you have successfully generated a certificate file for securing your XMPP communications.

Configuring jabberd2 to use the certificate

Configuring jabbed2 is pretty easy to configure to use the certificates.

Two configuration files need to be modified, and two components need to be restarted.

Configuring client-server encryption

The first of the configuration files to be modified is the `c2s.xml` configuration file (found in `/usr/pkg/etc/jabberd/c2s.xml` on `pkgsrc/NetBSD/DragonFlyBSD`, `/usr/local/etc/jabberd` on OpenBSD/FreeBSD). The stanza to be modified is `<local><id></id></local>`. You want to

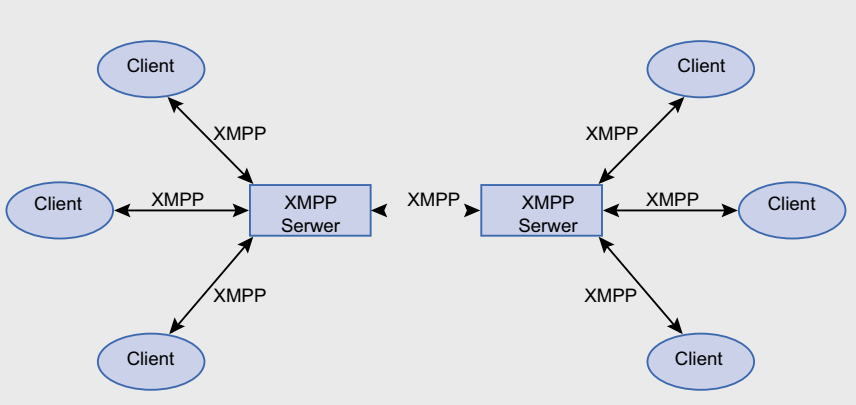


Figure 1. XMPP

add a `pemfile='<pemfile filename>'` to the `<id>` tag. In the standard `c2s.xml` file, there is a commented out stanza showing the correct syntax. Listing 3 shows the (stripped down) stanza in use on `jabber.cirr.com`.

Once you've restarted the `c2s` component, client to server communications can now be encrypted, assuming the client supports TLS authentication/encryption with the server.

Fire up your favorite TLS capable Jabber client (Psi is one such client) and look for the secured icon. Note, using a self-signed certificate may cause the client to produce a dialog about an invalid certificate authority (CA.)

Configuring server-server encryption

Configuring server to server (`s2s`) encryption is as easy as configuring client to server (`c2s`) encryption. The stanza needing attention in the `s2s.xml` is `<local><pemfile></pemfile></local>`. Uncomment that clause, and update the file path as appropriate. To verify that TLS encryption is working, verify that the s2s component started with no errors about the certificate. Then, attempt to get the presence information about someone on a TLS secured XMPP server, such as *jabber.org* or *jabber.cirr.com*. Look in your `s2s` log file, and search for a message similar to the following:

```
[7] [208.68.163.220, port=54127]
incoming route \
'portnoy.cirr.com/jabber.org' is now
valid, TLS negotiated
```

Congratulations, you have successfully secured communications between your XMPP client and your XMPP server, and between your XMPP server and other (suitably configured) XMPP servers (such as *jabber.org*, *jabber.cirr.com*, or others.)

That wraps up securing/encrypting communications between your XMPP client and your server, and between your server and others! That was relatively easy, was not it.

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 prefered OS is NetBSD, which he has running on Alpha, UltraSPARC, SPARC, amd64 and i386.

Michele Cranmer is a relativity new user to UNIX and Jabber, having been basically forced into learning it when she met Eric. After having been a loyal Windows and Yahoo Messenger user for many years, she finds that she prefers the *new* systems to the others because of ease of use and reliability. Being a college student, getting her degree in Special Education, she plans on using the *new* systems in her classroom as a way of teaching the children that there are many different ways to do things other then the *normal* ways and those ways are no more strange or unusual then they are.

OpenBSD
and making money

Girish Venkatachalam

Open Source is often alleged as being apathetic towards business and money. Corporations often accuse open source for being unable to bring in the profits that run a business. Nowadays everone knows that open source is serious and cannot be ignored.

I am going to demonstrate in this article that open source can not only mean seriuos business but also make you rich. No kidding. There are many entrepreneurs among OpenBSD developers and they use OpenBSD which has the most liberal licensing that any OS has and still interestingly they make a living out of it. I am going to show you how I use OpenBSD to make a living in Chennai, India.

We are going to be talking three different topics but related to one another in a subtle way.

Spam control

Spam control is big business in organizations. Employees having to deal with unsolicited commercial/bulk mail is something that not only reduces productivity but also eats into the company's bottomline.

Another thing that eats into the company's bottomline is the lack of productivity and disturbance caused by Microsoft Windows due to its various vulnerabilities, viruses, worms , trap doors and other malwares not to mention crashes of course. We will get to that in a minute.

First spam control.

Spam control with OpenBSD greylisting

Spam control has to invariably fall under one of the following categories.

- Bayesian filtering and contextual analysis
- Heuristical filtering based on known keywords/bad words
- CRM114 Markovian chain based filtering (related to a)
- Vipul's razor approach of DCC (Distributed checksum computation) with manual interference – gmail uses this heavily
- Greylisting to stop spam right at the MTA level
- IP address blacklisting and e-mail address whitelisting
- TMDA – cure worse than the disease (Only approved senders can send mail)
- RBL lists , spamhaus (politically sensitive spam control techniques)
- Sender Policy Framework(SPF) (not a bad idea per se) but does not work well

This is more or less it.

And most of these techniques are based on content scanning/filtering and actually reading e-mails with a computer.

Since this is an activity that requires a high end CPU and memory, spam control software and virus scanning software typically end up grinding your machines to a halt or even slow down your legitimate e-mails.

Also there is the very scary possibility of losing e-mails due to false positives.

OpenBSD's spamd uses a technique called greylisting. This is a very smart way to combat spam since it is stopped right at the MTA level. Since this never reads e-mail it is also very fast and highly efficient.

It is impossible to get a false positive here though the first mail from a domain will experience a delay.

I have seen some problem with popular mail sites like yahoo and gmail but they can be easily resolved by manual whitelisting.

Basically greylisting forces mail servers to be RFC 2821 compliant and retry mails until the receiving site is ready. This also has an added advantage of hurting spammers sometimes and also stopping the spam that is meant for some other sites.

The architecture of our solution is something like this (Listing 1). Here is a schematic to explain how OpenBSD greylisting works.

The firewall that works in the appliance redirects e-mail traffic depending on three parameters:

- Sending IP address (From IP)
- Envelope sender (who sends you mail?)
- Envelope recipient (who is mail addressed to?)

If the above `3 tuple` are seen for the first time then the mail sender is subjected to the torturous SPAMD filtering (running on port 8025 above). There is a phenomenon called *initial stuttering* that happens here. Instead of talking at full speed the MTA accepts mail one character at a time. This will piss off spammers and many go away. But legitimate senders have just one mail to send. Moreover they have to be RFC compliant. So they survive the test.

Once this process is completed, any subsequent mails from this sending IP address is assumed to be legitimate and they directly talk to the company mail server.

There are several parameters that can be tweaked here. So we can tighten the screws a bit once we observe how this comes up in production. And you don't waste your storage space and bandwidth receiving spam first and then rejecting them. Overall a very brilliant idea no doubt.

To configure spamd(8) all you have to do is enable it in /etc/rc.conf.local by adding these lines.

```
pf=YES
spamd_flags=""
spamd_black=NO
spamlogd_flags="-i fxp0"
```

I am of course assuming that your network interface is `fxp0`.

And your pf.conf should have these lines.

```
table <spamd-white> persist
no rdr on fxp0 proto tcp from <spamd-white> to any port smtp
rdr pass on fxp0 proto tcp from any to any port smtp -> 127.0.0.1 port spamd
```

Of course there is more to it than meets the eye but you get the idea.

Anyway as a bonus this also stops all sorts of irritating malware like virses, Trojans, worms and other annoyances. Such mails usually propagate with reckless abandon and my firewall running in the appliance can rate limit them.

Service Redirector

Another need the big corporates have is ensuring 100% uptime for their critical servers. This could include web servers, mail servers, database servers or anything else that forms the backbone of a company's business.

OpenBSD has two very simple ways to solve this problem – CARP and relayd.

CARP is a protocol that works at a very low level. Hence its ability to fail over is fantastic. Since it works at layer II, you can trivially fail over any service you offer since all services will be offered with an IP address.CARP configuration is brain dead simple and anyone can get it working within minutes.

If you have two OpenBSD boxes that you want to fail over in case one goes down then all you have to do is create the carp0 interface on both machines like this.

```
## Host A (MASTER)
# ifconfig carp0 create
# ifconfig carp0 192.168.1.10 vhid 1
carpdev fxp0
```

and on Host B,

```
## Host B (BACKUP)
# ifconfig carp0 create
# ifconfig carp0 192.168.1.10 vhid 1
carpdev fxp0 advskew 100
```

That is all there is to it. Now trying pinging the virtual IP 192.168.1.10 you just created from a different host.Then try something interesting. Plug out the ethernet cable from Host A. You can check which one is master with the *ifconfig* command.

You will notice that the *BACKUP* will take over within few seconds and start responding to ping requests. Once you plug the cable back in you will see that the *MASTER* and *BACKUP* roles will get interchanged automatically as per our original intention. *CARP* is really simple to get working but there is more to it. You need to allow the IP CARP protocol as well as the *PFSYNC* protocol in case you ar interested in synchronizing the firewall states before fail over. And in most real world applications you have to take care that the state of the backup is up to date with the master or at least reasonably close. For instance if you are doing a fail over of the antispam appliance then you need to ensure that the pf tables are in sync. And also the `/var/db/spamdb` database. You can easily ensure this by running a cron job to rsync or even copy it from place to another. relayd(8) is another interesting daemon introduced in OpenBSD recently that can do quite a few interesting things. We are not going to discuss most of its cool features here. We will just take a look at its potential. The gory details are in the man pages of OpenBSD as is the usual case with the OS. There is no OS that places as much emphasis on correct documentation like OpenBSD.

What does relayd do?

It is a service redirector. It is also many other things but for me it means that in case the customer runs a web server on an OS other than OpenBSD, then I can fail over the web server using relayd. But then you should remember that relayd works at a much higher layer in the OSI stack and consequently you should always try to use CARP for fail over as much as possible. Relayd can act as an SSL load balancer. This is a very useful feature since what we require is a secure connection only till the point it reaches our internal network. Beyond that we can load balance using unencrypted/unprotected sessions. So what relayd can do for us is finish the SSL handshake at the entry point to our network so that we can serve many customers even when using SSL. SSL based HTTP servers are typically highly loaded due to the crypto operations and other latency. This comes as a boon for such businesses.

Firewall

In the last issue I had covered firewalling with OpenBSD pf. pf forms such an important component of OpenBSD networking that any networking product that uses OpenBSD will invariably use it. pf can be used for NATing, blocking certain ports or redirection. It can also be used for load balancing.

We talked about the various ways in which Windows hurts a business in the beginning. OpenBSD based firewalling can be used to good effect using its ability to do passive OS fingerprinting. pf comes with an ability to detect the OS of a particular machine by inspecting its TCP SYN packet. So we can use this to make sure that Windows machines do not send malicious traffic.

Conclusion

We have very clearly seen how OpenBSD helps you succeed in business and make as much or even more money than companies that sell commercial software or hardware. The model of open source software based appliances have a great potential since most businesses are worried about support. If you can provide them support for the hardware and the open source software they will be willing to purchase your product. The reason is simple for businesses.

Open source software gives them unlimited freedom and there are no pesky limitations like number of concurrent users and other irritations like license renewals that are typically found in commercial software.

In short, OpenBSD is serious business!

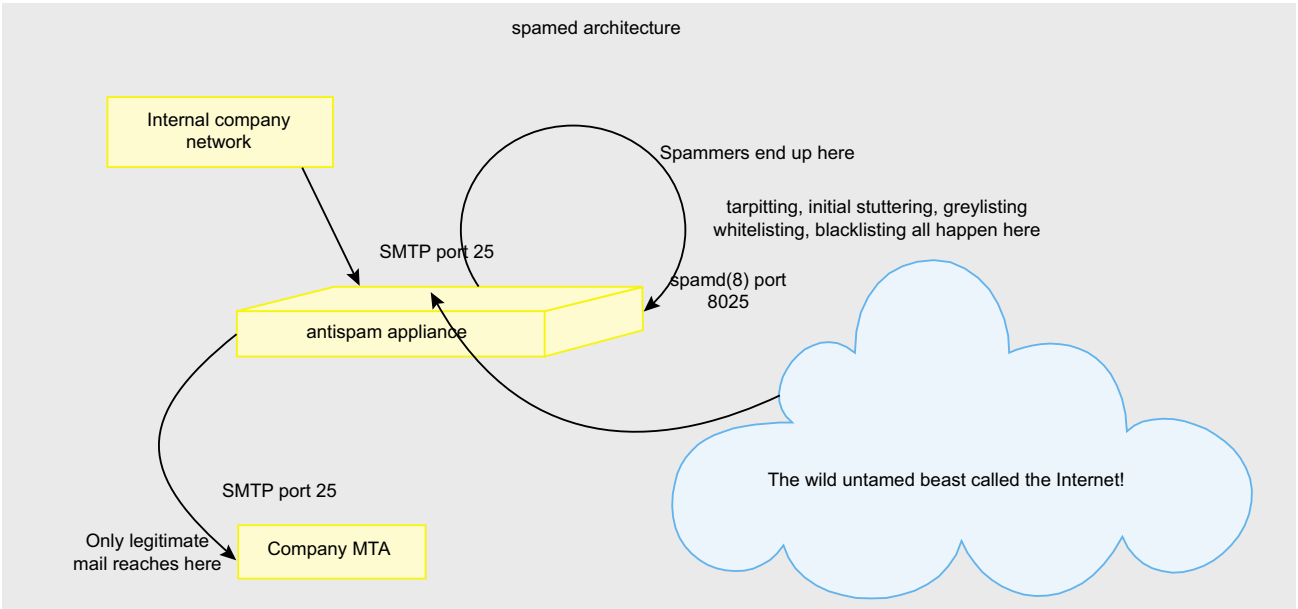
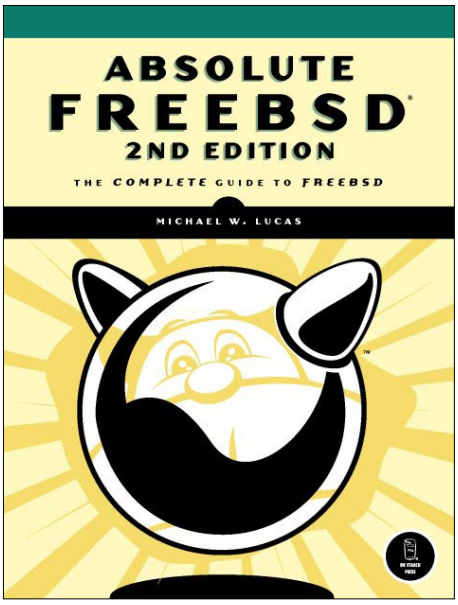


Figure 1. Spamed architecture

Absolute FreeBSD 2nd Edition



Written by MICHAEL W. LUCAS Jr (the W. in the middle is important, the appended Jr is even better) this 700 pages book is the updated opus of the famous FreeBSD bible from No Stark Press. Known for their unique books on technology, they give focus on Open Source, security, hacking, programming, alternative operating systems and Absolute FreeBSD 2nd Edition is no exception.

You will learn to manage your FreeBSD system, from installation to configuration and lot's more, like how to build your own embedded devices, how to encrypt disk partitions, how to use FreeBSD's multiprocessor features to your best advantage, how to run diskless servers, and more!

Absolute FreeBSD, 2nd Edition covers installation, networking, security, network services, system performance, kernel tweaking, filesystems, SMP, upgrading, crash debugging. It includes also a lot of tutorials and how to : Use advanced security features like packet filtering, virtual machines, host-based intrusion detection, build custom live FreeBSD CDs and bootable flash , manage network services and filesystems, use DNS and set up email, IMAP, web, and FTP services for both servers and clients, monitor your system with performance-testing and troubleshooting tools, run diskless systems, manage schedulers, remap shared libraries, optimize your system for your hardware and your workload, build custom network appliances with embedded FreeBSD, implement redundant disks, even without special hardware, integrate FreeBSD-specific SNMP into your network management system.

The first edition is 7 years old, and was a complete guide to FreeBSD 4.0 at that time. This second edition is about the last FreeBSD version 7.0, with all the tools from 4.0. Of course this book applies also to earlier version as well as for future version. Michael's coverage of GEOM, NanoBSD, FreeSBIE, journaling, memory file systems, filesystems in a file makes this book a must have even for the readers of the first edition. New readers will still get the solid introduction they need, concepts are explained clearly and with a lot of examples in this easy-to-use book. It's a great first step for those who would like to become committers or contributors in the future.

MICHAEL W. LUCAS (Jr) has been using Unix systems for more than 20 years and FreeBSD since 1995. Developer himself, and as a long term contributor of the FreeBSD system, he provides in his books a clear point of view of the operating system. Written with the help and advice of dozens of FreeBSD developers, the answers are straights, the concepts given clearly. Famous for is cool writing talent, the author of the Absolute series makes it easy to read, very lively for a system administration guide. You can make yourself an idea with the chap. 8 available for free in the editor's webpage. More than a book it's a manual aimed to the regular users who want to cleanly handle their desktop and the sysadmins who want to know how the machine thinks. Of course it's all about command line interface and configuration files, those used with GUI environments and click-here-and-then-there tutorials will discover the strength and the flexibility of Unix and how the FreeBSD system is organised.

This book covers almost everything that appears in 7.0 except too recent developments like binary updates. It is nonetheless a bible for FreeBSD users and sysadmins. Now you don't have to google for every little command or single configuration detail you're looking for.

PC-BSD
in Schools

iXsystems

Security, Stability, and Ease of Use Make PC-BSD Deployment in Poulx School District a Success. School District Deployment Sets the Tone for Future PC-BSD Deployments Throughout France

PC-BSD provided the stable and secure solution we needed for a trouble-free deployment in the Poulx School District at a negligible cost, say Marie Walrafen and Guillaume Fontaine, owners of *Chamanik.com*.

- PC-BSD is easy to install
- PC-BSD is free and open source
- PC-BSD is secure, reliable, and provides excellent content management
- PC-BSD is easy to support
- PC-BSD can handle multiple users on a small network
- PC-BSD is based on FreeBSD

Schools, businesses, and government offices have a basic set of needs when it comes to deploying a desktop operating system. They need a solution that runs smoothly and efficiently, with minimal effort on behalf of the parties involved. The solution also needs to be safe, secure, and easy to implement and maintain.

The *Poulx School District* did not have a need to run highly specialized applications. What they required was an operating system that is stable, reliable, and free of viruses. Unfortunately, hackers are continuously writing viruses for the Windows environment, and these viruses hamper the successful operation of a network. And while Linux protects against most viruses and is a low-cost open source alternative, it doesn't feature the stability and security of FreeBSD.

PC-BSD is a fully functional desktop operating system running FreeBSD 6 under the hood. Its graphical system installer makes the system installation process effortless. Its self-installing software packages make loading programs a snap. It is secure, reliable, and easy – a perfect tool for all basic needs and especially fit for use at a school, small business, or government office.

In February of 2008, Marie and Guillaume deployed PC-BSD in the Poulx School District in France.

They installed PC-BSD on a small network that had previously been running the Mandriva version of Linux.

Marie and Guillaume were already familiar with PC-BSD and FreeBSD, having deployed it for the wireless network in the city hall. They knew that the applications needed to run on the systems in the school were compatible, and that all the applications could be run with existing PBI's (push button installers) available for PC-BSD. Hundreds of easily installed PBIs are available for download from *http://www.pbidir.com*, with updates made daily. Many are also available on Disc 2 of PC-BSD. They also knew that PC-BSD can be installed very quickly and is easy to use, and can handle multiple users on a small school network. They made the recommendation to deploy PC-BSD in the schools, and have never looked back.

Marie and Guillaume downloaded PC-BSD Discs 1 and 2 free of charge from *http://www.pcbsd.org*. Marie used the Disc 1 copy as the install disk on all the machines. When each machine had completed the install process, Marie removed Disc 1 from the machine and inserted Disc 2. It took only a few minutes to install PC-BSD on each computer.

The final steps of the deployment process took about half an hour to complete. Marie configured the internet access for the school network and installed the French language files from the second CD. She also installed the PBIs for critical applications needed by the school. Through the use of the PBI software Marie was quickly able to install Gimp, Planetarium, and various educational games.

The school's requirement for preventing inappropriate site content from being accessed by students resulted in the need to set up a proxy server as a filter. Methods and protocols were established so that teachers were able to log in and connect to the internet without going through the proxy server for unrestricted searches and research. The systems were also set up so that the teachers could boot from their individual computers, instead of having to boot from the general server.

Marie set up an individual profile for each pupil on the school network, which would allow documents saved on the network to be accessed by students using any computer within the network. All software needs were accommodated by existing PBIs.

All in all, the deployment process was highly successful. Marie just laughed when asked to describe a technical problem she had had during the deployment, as there were none. Support issues since the deployment have been minimal as well, consisting primarily of hardware upgrades and other issues not related to PC-BSD.

The teachers are very comfortable using PC-BSD and appreciate its ease of use and trouble-free administration. They have forgotten all about Mandriva and Windows XP (which they were using before Mandriva). The students have been able to access their files with ease, and some of them are enjoying PC-BSD so much that they have asked Marie and Guillaume how to install it on their desktop at home. They appreciate the possibility that there is an available alternative to Windows, and even to Linux.

The solution deployed by Marie and Guillaume in the school can be easily replicated in an academic, government, or small business environment. Marie and Guillaume are in the process of setting up other deployment contracts within the Poulx school district, as well as throughout France. It is easy to sell the PC-BSD implementation solution to other entities given PC-BSD's stability, reliability, and trouble-free system administration. Marie says that even though she is the *technically ignorant half* of the partnership with Guillaume, she was able to get up to speed on installing and using PC-BSD in no time. PC-BSD is also significantly more cost-effective than its closest non-open source competitor, which costs upwards of $200 per copy for the full version of the operating system.

Marie and Guillaume are also taking their solution to the Poulx City Hall, which previously contracted them to set up the city's wireless network. City Hall is currently running Windows on 8 of the 12 available computers but has agreed to gradually switch the remaining computers over to PC-BSD. Marie and Guillaume are confident that the software used to run city hall's administrative functions can be made to work on WINE (a compatibility layer for running Windows programs on top of UNIX). They intend to eventually develop their own solution that does not need WINE. Once the switch-over is complete Poulx will have the unofficial title of *FreeBSD City* bestowed upon it by its Mayor.

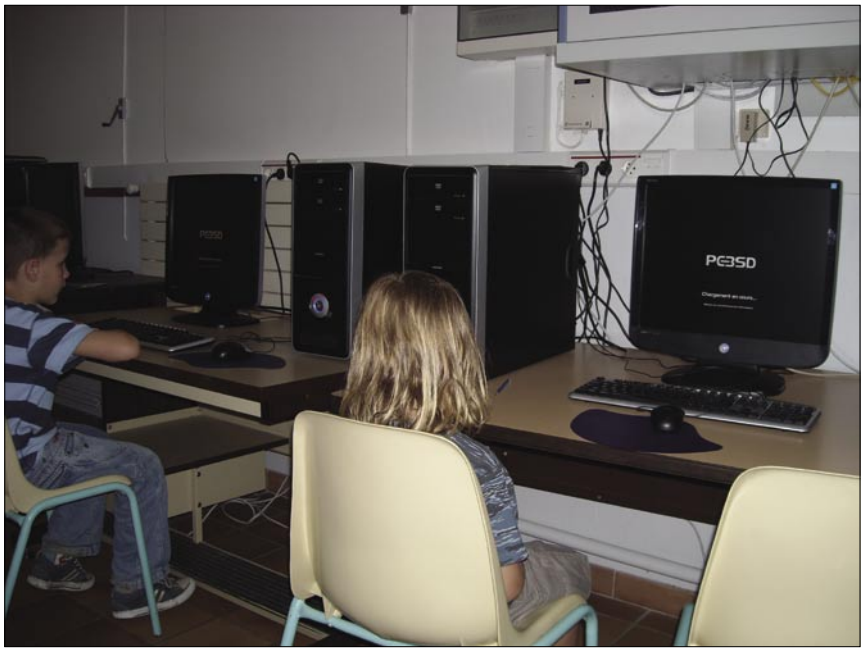


Figure 1. Poulx School District

General Advantages of PC-BSD

In addition to some of the items listed above, there are a number of reasons to deploy a FreeBSD-based solution when designing a network architecture. Because the underlying OS for PC-BSD is FreeBSD, these advantages apply to PC-BSD as well.

First of all, the FreeBSD license is unrestrictive and user-friendly, and consists of only a couple of clauses. It does not require people to make their code changes public, which means that you can take BSD licensed code, change it, and sell it as closed source software. The same is not true for Linux, another popular open source OS, which is released under the GPL (GNU Public License) and requires that changes be contributed back to the source code. As a result, when Linux code is modified, these changes are not proprietary.

Furthermore, FreeBSD eliminates most dependency issues through the FreeBSD Ports System. The Ports System is a software management infrastructure for easily installing, upgrading, and maintaining software on the system.

With PC-BSD the PBI's can be installed in addition to the over 18,000 ports of available applications. PBI's are not part of the centralized repository system. While the PC-BSD Project hosts and maintains many popular programs in PBI format, users can download programs from anyone who has a PBI, and anyone can build PBIs and host them. This is different from Linux, where software availability is mostly controlled by the distro manufacturer.

Finally, FreeBSD is a centrally developed and maintained operating system, whereas Linux is a kernel wrapped in mostly GNU userland utilities. This means that with FreeBSD, a single project comprised of various teams is responsible for the kernel AND userland while in Linux, userland utilities and kernel versions are different from distribution to distribution.

Interview with Damien Bergamini

OpenBSD developer

One of the most requested features for wireless networking should be part of OpenBSD 4.4. I am talking about WPA, and I had the pleasure to interview Damien Bergamini, the developer who made a huge work for OpenBSD wireless subsystem.

Damien worked on the drivers, reverse engineering and building some of the code that can now be found in most free OSes, even OpenSolaris!

The work he did on the WPA implementation follow a different design, as the code runs in the kernel, and provide a very clear way of configuration: ifconfig.

You could setup WPA-PSK in station mode with a simple line:

```
# ifconfig ral0 wpa wpapsk \
0x0e8de50e2a614dbd83df61db3e042b39617
7e8cc8ef7e1f2e83e158a19ba5ea3
```

or a WPA2-PSK setup for access point mode with:

```
# ifconfig ral0 mediaopt hostap nwid
openbsd_ap chan 5 \
wpa wpaprotos wpa2 wpaciphers ccmp
wpagroupcipher ccmp wpapsk \
0x0e8de50e2a614dbd83df61db3e042b39617
7e8cc8ef7e1f2e83e158a19ba5ea3
```

Keep reading for the other cool details!

Could you introduce yourself?

I am French, I'm 28 years old. I'm an OpenBSD developer since 2004. I have written numerous drivers for 802.11 wireless devices, and lately, I added support for WPA-PSK (Wi-Fi Protected Access using pre-shared keys) to our generic 802.11 layer.

What type of difficulties did you have to overcome to implement WPA/WPA2?

The reason it took a long time to implement WPA in OpenBSD is that the various standards that make WPA are fairly complicated. It's a steep learning curve.

Of course we could have thrown in whatever existing WPA implementation that would have made the trick but this is not the way we operate in OpenBSD.

OpenBSD tends to be more quality-driven than feature-driven. Before we import a large piece of code in the base system, we must make sure someone in OpenBSD can maintain that code and can fix it should it break. This means at least one developer must fully master that code and be very comfortable with it. We prefer to not support a feature rather than import code we cannot maintain. Although this may be frustrated for our users sometimes, this is a winning strategy in the end.

Before beginning my work on WPA, I studied various existing WPA implementations (mostly *wpa_supplicant*, hostapd and xsupplicant) but I did not like their design so I decided to write my own implementation from scratch, taking a very different approach.

What differences do you see in OpenBSD's WPA implementation compared with other BSDs' ones?

Other BSDs use *wpa_supplicant* for client mode and hostapd for AP mode.

The reason I chose to not go that road is that *wpa_supplicant* and hostapd are rather huge (in terms of lines of code) and that they try to implement too many things at the same time (`802.1X`, `802.11i`, EAPs).

I particularly did not like the way those tools were reimplementing parts

of the 802.11 management entity (MLME) in userspace. This is very redundant with what we already do in the kernel, and it requires that the kernel implement hooks to let the userspace play with the 802.11 management state machine.

In OpenBSD, support for 802.11i is fully implemented in the kernel (in our generic 802.11 layer) because this is the natural place to do it (this is where we keep all the information and states about APs and stations.) As a result, you can setup a WPA-PSK network (AP or client mode) without running any external daemon.

You only need to know one command: *ifconfig*.

However, in OpenBSD, we do not support WPA-Enterprise yet, while other BSDs support it. But this is something I'm actively working on.

I did like to implement the 802.1X PACP protocol in the kernel (both supplicant and authenticator state machines) for both wired and wireless interfaces. Then I will implement some of the most used EAPs.

Does running WPA in the kernel increase the security risk?

Not at all. In this particular case, I would say quite the opposite because implementing the 4-way handshake and group key handshake in userspace require that you to let the userspace control the 802.11 kernel state machine which is very error-prone given that the 802.11 state machine is quite complicated and that not all drivers handle all the possible state transitions properly, especially those that implement the 802.11 state machine in firmware.

Considering that your implementation runs in the kernel, do you see any performance advantage over the other implementations?

No. Except for software encryption/decryption (that other OSes do in the kernel too), WPA is not performance critical.

It consists in the exchange of a small number of packets (4 for the 4-way handshake) between the supplicant (the client) and the authenticator (the access point). This does not require any special optimization.

Is there any work on performance improvements or power saving for wifi drivers?

I'm currently adding hardware crypto support for more chipsets. This should help a bit performance-wise. I'm also working on supporting stations in power-save mode when operating as an access point.

I remember that you used only software crypto for WEP, instead of the features included in some chips. Is this still true? What about modern WPA-compliant chips? What advantages do you have using software crypto and opensource drivers?

That is not exactly true. Some drivers were already doing WEP in hardware, however, because CCMP is more costly to do in software, it will become critical to support hardware crypto for more devices. I have already implemented hardware crypto for TKIP and CCMP in the Ralink RT2860 driver to make sure our net80211 design was clean enough to allow for both types of crypto.

I am now working on other drivers, like wpi(4) and iwn(4). Some crypto engines are so badly designed though that supporting them will offer little to no performance benefit (because, for instance, even if the device supports scatter/gather, the crypto engine does not, and you have to copy every outgoing packet). For these devices we will continue to use the software crypto code.

OpenBSD developed a lot of drivers for wireless chips using reverse engineering. We saw some exploits for closed-source drivers provided by vendors. Were your drivers vulnerable? What type of measures did you adopt to improve wireless drivers security?

Offering open-source drivers does not guarantee that no vulnerability will ever be found. However, you do not need to wait for the vendor (or the developer that wrote the driver under an NDA) to fix that vulnerability.

How are your relationships with vendors? Do they offer you access to datasheets and specs without NDA agreements? Do they let you redistribute their firmwares?

Only a few vendors provide datasheets without NDAs. Ralink is one of them. Zydas also provided some documentation for their USB chipsets before they got bought by Atheros.

There was some documentation available for the earliest Realtek chipsets too, but I'm not sure it's still the case for their latest chipsets. Some vendors, like Intel or Marvell, provide open-source Linux drivers but no documentation. The worst players are Atheros and Broadcom, though things may change with Atheros in the future.

From a security point of view what setup would you suggest for a wireless network?

For a home network, WPA2-PSK (with 256-bit AES) is a good compromise between security and ease of configuration. WPA2-Enterprise or IPSEC are equally good solutions for enterprise networks.

What reasons do you see to deploy an OpenBSD based access point instead of using one of those cheap little boxes?

Of course, you can always use a classical access point as a bridge if you want, but it is a bit of an overkill if you want to build something small. With the support of more embedded systems in OpenBSD (armish, socppc ports), it becomes even more important to have a good support for AP mode. This way you can for example setup a smaller NAS with Wi-Fi support, and all the good things that OpenBSD brings to you (pf, etc).

Any thought on 802.11n?

802.11n is not yet standardized at the time of this writing [May 2008]. It is not yet supported in OpenBSD.

Although we already have drivers for 802.11n devices, they only support 802.11g mode for now. Some parts of the 802.11n specification are very complicated to implement (like block ACK sessions) while the performance gain in a real-life setup is not clear at all.

I don't buy the argument about the improved speed in 802.11n at all. Anyway, I'm planning to work on 802.11n at some point, but there are more important things to do first, like multi-bss support and improved power management.

by Federico Biancuzzi ed@bsd.it

Mac OS X the other BSD

Mikel King

Apple's emergence into the BSD Community has been a long and storied one. While they are quick to claim membership as they have derived much of Mac OS X from various points, most notably from FreeBSD 5, I often wonder how much have they returned. Granted, there are not any requirements for such participation in the community, which is a major facet of BSD licensing as a whole. Still, it would be nice to cite some examples of their contributions, and respectfully offer some suggestions.

One of the most often overlooked aspects of Apple's BSD lineage is the fact that, as far as Open Source is concerned, they single handedly launched FreeBSD into the stratosphere, numbers-wise. BSD can accordingly claim more desktop installations than any other freely available OS, including all of the Linuxes combined. However, I am still left wondering, "Is this enough?". This is especially so since Mac OS X, like DragonFlyBSD, is a fork off of FreeBSD 5, which has been officially deprecated as of the release of FreeBSD 7. I am not saying that either of these products are flawed, just that I have to wonder what Apple's game plan is.

When Apple made the shift to FreeBSD 5 as the base of their OS, many pondered the possibility that Apple would simply evolve their product along the line as FreeBSD itself evolved. Considering that they use their own version of the Mach kernel, there may be little benefit for them to incorporate FreeBSD's major evolutions into their product. Yet it would seem that with the switch to an Intel-based architecture, it would be possible one day to run FreeBSD with an Apple UI; and that truly would be interesting. Considering that they were a bit tardy with the Leopard release, it certainly might help reduce their overhead if they were to adopt this approach.

Another interesting point would be to fully incorporate the MacPorts into the base OS right from the installation. This could be especially true on their server version of the product, where it should be a trivial matter to update the installed version of, say PHP, to add a new feature not bundled in the original installation. Do not even get me started on sed, which is version 0.1 from 1987. While Mac OS X updates fix the items they have added to the OS and eventually tie up the loose ends in security issues, they typically do not address that lagging UNIX under-belly.

Personally I would prefer the FreeBSD model where you install the OS bare-bones. Then install things like Apache from the ports rather than have them installed by default, as you would on other overly bloated operating systems. The obvious benefits of this approach are well-documented and are discussed to death on the various FreeBSD mailing lists and forums.

Another Open Source project to feel the touch of Apple's broad borrowing is the KDE project, as they have adopted the KHTML engine, which is the basis of the Konqueror and their Safari browser. Here again I can not find a direct example where Apple has done anything more than tell the world *Hey we use KHTML as the basis for our browser* thus drawing attention to the project that it would not have otherwise garnered on its own. What is truly interesting here is that a tangent of the KDE project has devoted itself to a natively deployable version on Mac OS X without the requirement of X11 at all.

This, of course, leads me to Apple's touting the ability of running thousands of ready made applications available from the X community. With Leopard Apple made the shift from XFree86 to X.org which was not a happy transition, to say the least. Here again Apple does not treat X as a part of the OS with regards to updates, and users had to wait for quite a while before the version supplied with Leopard was stable. Fact of the matter is numerous users installed the version found in Tiger in lieu of the latter version so that they could continue to run their favorite applications. This is yet another example why Apple should just incorporate the ports directly into the OS. Were they to provide the necessary libraries, components and patches, users could keep their systems up to date without issue.

To be fair, I have read that Apple has been kind enough to donate hardware on occasion to Open Source projects. However, I must admit they do not make that list of recipients well known. To sum up their involvement in the Open Source community, it appears to be little more than a marketing ploy, which is truly sad.

If you compare their involvement to that of IBM or NOVELL, who both have a clear track record, Apple would look more like a SUN rather than a true Open Source contributor. Sun has eked ahead only slightly with the recent purchase of MySQL and the decision to keep it open (for now). Finally, sad as it is to say, Microsoft has a more clearly defined stance on Open Source; they made no bones about using FreeBSD's TCP/IP networking stack for years without any intentions of giving anything back to the community.

All in all, I must ponder what sorts of leaps and bounds could be made if Apple worked more closely with the community.

Community membership application status: Probationary Approval

In the next issue:

A complete guide to PC-BSD

- Enable prisons in PCBSD for maximum horsepower
- Virtualisation in PC-BSD

and much more...

Next issue of BSD magazine available in December !

PC-BSD 7.0 Fibonacci Edition

Perfect for Your Server and Your Desktop

PC-BSD is the operating system of the future. A fully functional desktop operating system running FreeBSD 7 under the hood. Less vulnerable to viruses, spyware, and crashes that plague other systems thanks to the stability and security that only a BSD-based operating system can bring. Easy to use thanks to a graphical system installer and helpful utilities that make installing and using PC-BSD effortless.

PC-BSD Fibonacci Edition takes many of the powerful features inherent in FreeBSD and makes them easier to use in PC-BSD. In addition to the powerful FreeBSD command line, most common tasks can be performed via the optimized and tuned KDE interface. For those who prefer a regular console OS, modify one file to turn off the GUI and disable the X server entirely.

In addition to the FreeBSD Ports and Package Management Systems for software, PC-BSD can install applications via the Push Button Installer (PBI), a graphical utility to remove and install software in a simple to use, self-contained format. With PC-BSD's PBI system hundreds of great programs are readily available and can be downloaded from http://www.pbidir.com, with updates made daily. Many PBI's are also available on Disc 2 of PC-BSD. Once downloaded, these PBIs install with their own libraries, eliminating the problem of shared dependencies. Installing one program does not necessarily mean breaking another as it often does with Linux. ;)

New server tools and enhancements featured in PC-BSD Fibonacci include speed improvements with the ULE Scheduler, experimental ZFS support during install, and UFS Journaling through GEOM. Furthermore, the online update manager can be customized to provide manual updates to individual servers or groups of servers automatically.

Desktop users will appreciate the new searchable kmenu using KBFX, enhanced WiFi compatibility including 802.11n support, and Improved Wine stability. The new Xorg graphical configuration tool allows for easy dual-head monitor set up right out of the box. These features are sure to make any user experience more rewarding and productive. PC-BSD is designed to meet the needs of every user from beginner to expert.