MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

ROLLING YOUR OWN KERNEL

INSIDE

VOL.4 NO.12
ISSUE 12/2011(29)
1898-9144

TrueNAS™ Storage Appliance: You are the Cloud

With a rock-solid FreeBSD® base, Zettabyte File System support, and a powerful Web GUI, TrueNAS™ pairs easy-to-manage software with world-class hardware for an unbeatable storage solution.

Expansion Shelves Available

TrueNAS™ 2U System

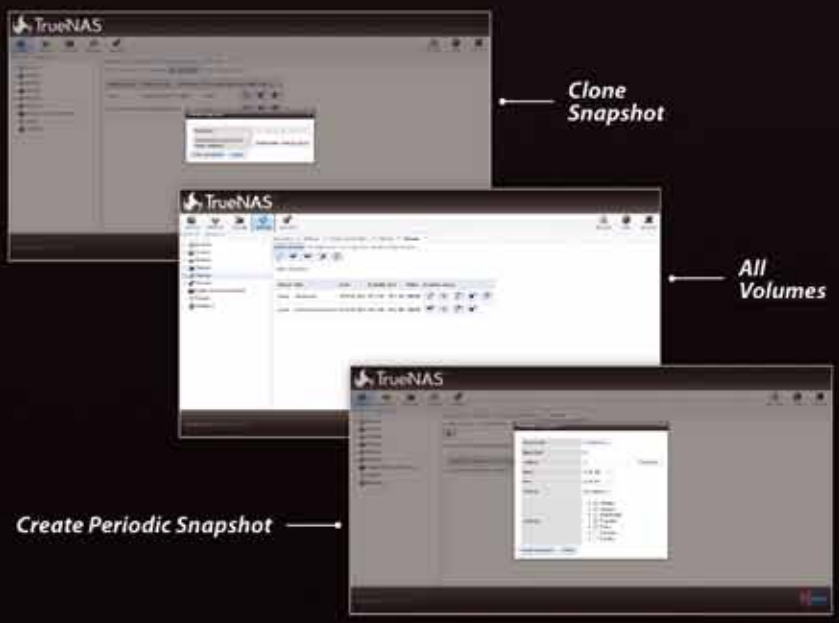
TrueNAS™ 4U System

FreeNAS 8 POWERED

Storage. Speed. Stability.

In order to achieve maximum performance, the TrueNAS™ 2U and 4U Systems, equipped with the Intel® Xeon® Processor 5600 Series, support Fusion-io's Flash Memory Cards and 10GbE Network Cards. Titan TrueNAS™ 2U and 4U Appliances are an excellent storage solution for video streaming, file hosting, virtualization, and more. Paired with optional JBOD expansion units, the TrueNAS™ Systems offer excellent capacity at an affordable price.

For more information on the **TrueNAS™ 2U** and **TrueNAS™ 4U**, or to request a quote, visit: **http://www.iXsystems.com/TrueNAS.**



TrueNAS™ 2U KEY FEATURES

- Supports One or Two Quad-Core or Six-Core, Intel® Xeon® Processor 5600 Series
- 12 Hot-Swap Drive Bays - Up to 36TB of Data Storage Capacity*
- Periodic Snapshots Feature Allows You to Restore Data from a Previously Generated Snapshot
- Remote Replication Allows You to Copy a Snapshot to an Offsite Server, for Maximum Data Security
- Software RAID-Z with up to triple parity
- 2 x 1GbE Network Interface (Onboard) + Up to 4 Additional 1GbE Ports or Single/Dual Port 10GbE Network Cards

TrueNAS™ 4U KEY FEATURES

- Supports One or Two Quad-Core or Six-Core, Intel® Xeon® Processor 5600 Series
- 24 or 36 Hot-Swap Drive Bays - Up to 108TB of Data Storage Capacity*
- Periodic Snapshots Feature Allows You to Restore Data from a Previously Generated Snapshot
- Remote Replication Allows You to Copy a Snapshot to an Offsite Server, for Maximum Data Security
- Software RAID-Z with up to triple parity
- 2 x 1GbE Network Interface (Onboard) + Up to 4 Additional 1GbE Ports or Single/Dual Port 10GbE Network Cards

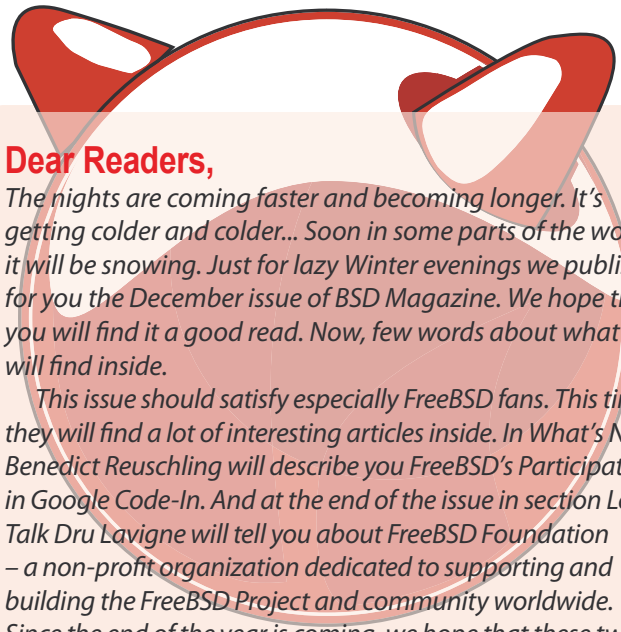
JBOD expansion is available on the 2U and 4U Systems

** 2.5" drive options available; please consult with your Account Manager*

Call iXsystems toll free or visit our website today!

1-855-GREP-4-IX | www.iXsystems.com

Powerful. Intelligent.



Dear Readers,

The nights are coming faster and becoming longer. It's getting colder and colder... Soon in some parts of the world it will be snowing. Just for lazy Winter evenings we publish for you the December issue of BSD Magazine. We hope that you will find it a good read. Now, few words about what you will find inside.

This issue should satisfy especially FreeBSD fans. This time they will find a lot of interesting articles inside. In What's New Benedict Reuschling will describe you FreeBSD's Participation in Google Code-In. And at the end of the issue in section Let's Talk Dru Lavigne will tell you about FreeBSD Foundation – a non-profit organization dedicated to supporting and building the FreeBSD Project and community worldwide. Since the end of the year is coming, we hope that these two articles will encourage you to support or participate in the future projects dedicated to FreeBSD.

The author of Rolling Your Own FreeBSD Kernel in his article will lead you step by step how to compile a custom kernel. He will also list you the advantages and disadvantage of having your own kernel. The last article dedicated to FreeBSD enthusiasts is Anatomy of a FreeBSD Compromise by Rob Somerville. In November issue you had an opportunity to read his last text from GIS series, now he begins the new series dedicated to security for admins. If you have any comments or expectations don't hesitate to write him or us and send the feedback!

The OpenBSD users won't be bored as well. First, in Developers Corner they will find out how to keep configuration files shiny as new by using sysmerge(8). Than, they will learn How To get a basic Cacti server running and monitor OpenBSD server with Symon.

To whole BSD users we can recommend the Michael Shirk's article – Hardening BSD with Security Levels which covers the configuration of security levels via securelevel for OpenBSD, FreeBSD, NetBSD and DragonFlyBSD. And Robert Fekete's text about Extracting Useful Information From Log Messages.

One last not mention yet is a great text by Kris Moore, who shows how to setup the Mac for dual-booting, and perform the installation. Short, entertaining and very practical text about Installing PC-BSD on a Mac.

There is nothing more left than to wish you enjoy the reading! Don't forget that by sending your feedback you help us to do a better job and get a content you want to read :)

Patrycja Przybyłowicz
& BSD Team

MAGAZINE BSD

Editor in Chief:
Patrycja Przybyłowicz
patrycja.przybylowicz@software.com.pl

Contributing:
Dru Lavigne, Kris Moore, Antione Jacoutot, Toby Richards, Paul Ammann, Rob Sommerville, Michael Shirk, Benedict Reuschling, Robert Fekete

Proofreaders:
Jeff Loupe, Tristan Karstens, Barry Grumbine, Sander Reiche, Christopher J. Umina

Special Thanks:
Denise Ebery, Dru Lavigne

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski

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

CEO:
Ewa Dudzic
ewa.dudzic@software.com.pl

Production Director:
Andrzej Kuca
andrzej.kuca@software.com.pl

Executive Ad Consultant:
Ewa Dudzic
ewa.dudzic@software.com.pl

Advertising Sales:
Patrycja Przybyłowicz
patrycja.przybylowicz@software.com.pl

Publisher :
Software Press Sp. z o.o. SK
ul. Bokserska 1, 02-682 Warszawa
Poland
worldwide publishing
tel: 1 917 338 36 31
www.bsdmag.org

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

What's New

Developer's Corner

How To

Security

Let's Talk

Google Code-In

and FreeBSD's participation

For the first time, the FreeBSD project is participating in another program run by Google Inc. to encourage student participation in open source projects – Google Code-In (often abbreviated to GCIN).

What you will learn…

- what the Google Code-In program is,
- what the benefits for open source projects as mentoring organizations are,
- the current state of progress for tasks by the FreeBSD Project.

While being similar to *Google Summer of Code* (GSoC), which FreeBSD has been participating in for a number of consecutive years, some aspects are quite different. This article will explain the program from a participating organizations point of view and what it's current progress looks like.

What Google Code-In is

Google Code-In is meant for pre-university students between thirteen and seventeen years who want to gain their first experiences contributing to an open source project. The program officially opened for students worldwide on November 21, 2011 and will run until January 16, 2012. Despite of what the Code-In part of the program's title suggests, the tasks that students are working on are not limited to writing code. Things like documentation, quality assurance, research, training, translation, user interface design or even outreach tasks to promote the project are activities to get a foot in the door of an open source project. Each project creates a number of tasks from each category (if applicable) totalling at least fourty tasks that the students can work on. More tasks will be added later as more and more tasks get finished by the students. Tasks usually have a description and a goal that must be met (for example, a document was translated or a bug was fixed) in order to mark the task as completed. For each task, the participants receive points – one for easy tasks, two for medium tasks and four for tasks rated as hard. The participating organizations are tasked with calibrating each task and to put at least one mentor behind each task to help students with questions they might have or general help getting started. Also, it's the mentors who decide whether a submitted task is considered finished and students will receive points for it. After the contest is finished, the ten students with the highest number of points earned will win a trip to Google's Mountain View, California campus for themselves and a parent or legal guardian. Unlike Google Summer of Code, there is no monetary incentive for open source projects engaging in Google Code-In. However, there are other benefits that make it palatable to mentor these students.

Why Google Code-In Participation is Important to FreeBSD

FreeBSD benefits in a number of ways from participating in the program. First of all, we gain valuable experience in the program itself (being the first time we ever took part in it). Google Code-In is different from Google Summer of Code in that it requires more active mentoring for the young participants. The turnaround time for the tasks which can be claimed and assigned to a single student at any time is also higher, requiring a certain flexibility on our part. Also, we cannot expect that the participants know how to properly interact with other people in an

open source project. Google warned prospective mentors that students might use excessive capitalization in their communications and other forms which are normally considered inappropriate on the internet. However, such behaviour has not surfaced yet for those who worked on tasks that the FreeBSD project has put up. Instead, the students we came into contact with were not that different in their communication style from other members of our community who post regularly on mailing-lists, the FreeBSD forums or on IRC channels. Of course, finishing as many tasks as possible from our list is one of the main goals that we try to accomplish with our participation as a mentoring organization. But besides that, we were also interested to see how much prior experience the students had with FreeBSD itself. Another bonus of participation that we hope will benefit us would be that students might continue to do work long after the Google Code-In contest has finished. Even if some of them do not become active committers, they had contact with our community and will hopefully spread the word that this has been a nice experience for them. And last but not least, the variety of areas that the students could engage in other than writing code (which nevertheless is a worthwhile thing to do) attracts different kinds of people to the project who would otherwise not consider a contribution. Imagine that a student may have discovered early in life to have an artistical talent. Illustrating certain aspects of the system or creating artwork to promote the project is something that may have an enormous impact on other people who might then better understanding what the FreeBSD project is all about. Reaching out to a broad range of people and to make use of their skills is a good way to ensure growing a community of users as well as contributors.

Preparations to Become a Mentoring Organization

The primary preparations like registering FreeBSD as a mentoring organization, creating wiki pages for tasks and calling for mentors were done by Wojciech A. Koszek. That's when I first heard of the contest. The main goal for being accepted as a mentoring organization was to put up fourty tasks for students to work on during the time

Glossary

- GCIN – Abbrv. For Google Code-In, a contest run by Google Inc. to encourage contributions to open source projects.
- GSoC – Abbrv. For Google Summer of Code, another contest run by Google Inc. to encourage contributions to open source projects by offering students to work for money during the summer on a software project.

On the Web

- *http://www.google-melange.com/gci/homepage/google/gci2011* – Google Code-In Homepage,
- *http://wiki.freebsd.org/GoogleCodeIn/2011* – FreeBSD's tasks for Google Code-In 2011.

Google Code-In is being held. Being interested in creating, updating and enhancing the FreeBSD documentation set of documents, I started working through the open task list that we put up in the FreeBSD wiki. Creating tasks took a little longer than initially expected. Additional explanations were necessary because people external to the project are usually not familiar with the tools that we use (or even the operating system itself) and might struggle with even the basic tasks. A simple documentation task soon looked like a huge amount of work due to the size of its task description, but at the core, it contained all the required information. Of course, tasks can be edited afterwards and that proved to be very helpful to us. Especially when it comes to the available time a student gets to work on a task exclusively, we discovered that more time is needed than a normal committer might require to do it. After all fourty tasks were created (not all of them by me) and many FreeBSD committers agreed to take up the mentorship for some of them, we decided that we need more ways to interact with students seeking help. So we made use of the IRC channel #freebsd-soc that we normally employ when Google Summer of Code is running and set up email contact information to reach the mentors with questions. We even created a VirtualBox image with a basic FreeBSD system containing checkouts of the documentation source trees so that students can start with main objective of the tasks rather than spending time with setting up a system for themselves.

What We've Learned so Far

While the contest is still running, a few early findings can be gathered already that may help when this contest runs next time and we participate again. Fears that students needed parenting or do not know the basics of writing properly formatted email or do now know basic IRC etiquette have not come up thus far. Students were not shy to ask when certain aspects of a task were not clear to them or if they struggled with things they did not know. Students who worked and finished a task are very likely to pick a similar task from the same project. For example, once someone knew how to write an article in the DocBook SGML notation that FreeBSD uses, it is likely that another article task is being claimed. Most of the students have never come into contact with FreeBSD or a UNIX-like operating system before. Only a few who might have had prior interactions with the project could start right away on the more easier tasks. Giving students enough time for a task is also important as they might ask more questions in between which takes away valuable time. When we saw that a student is interested in finishing the task, we generously extended the time available to let them finish the task without additional agitation. For me personally it was very rewarding to see how many committers rallied to help mentor the tasks that I put up. This not only meant that the tasks are still relevant for others, but also that they want to spend some time helping out newcomers apart from their usually high workloads in the FreeBSD project. I hope that the students keep up the pace with finishing these tasks (11 of 72 at the time of this writing) and that they not just see the tasks as milestones towards the contest goal, but actually as valuable work that helps the FreeBSD project and its userbase immensely.

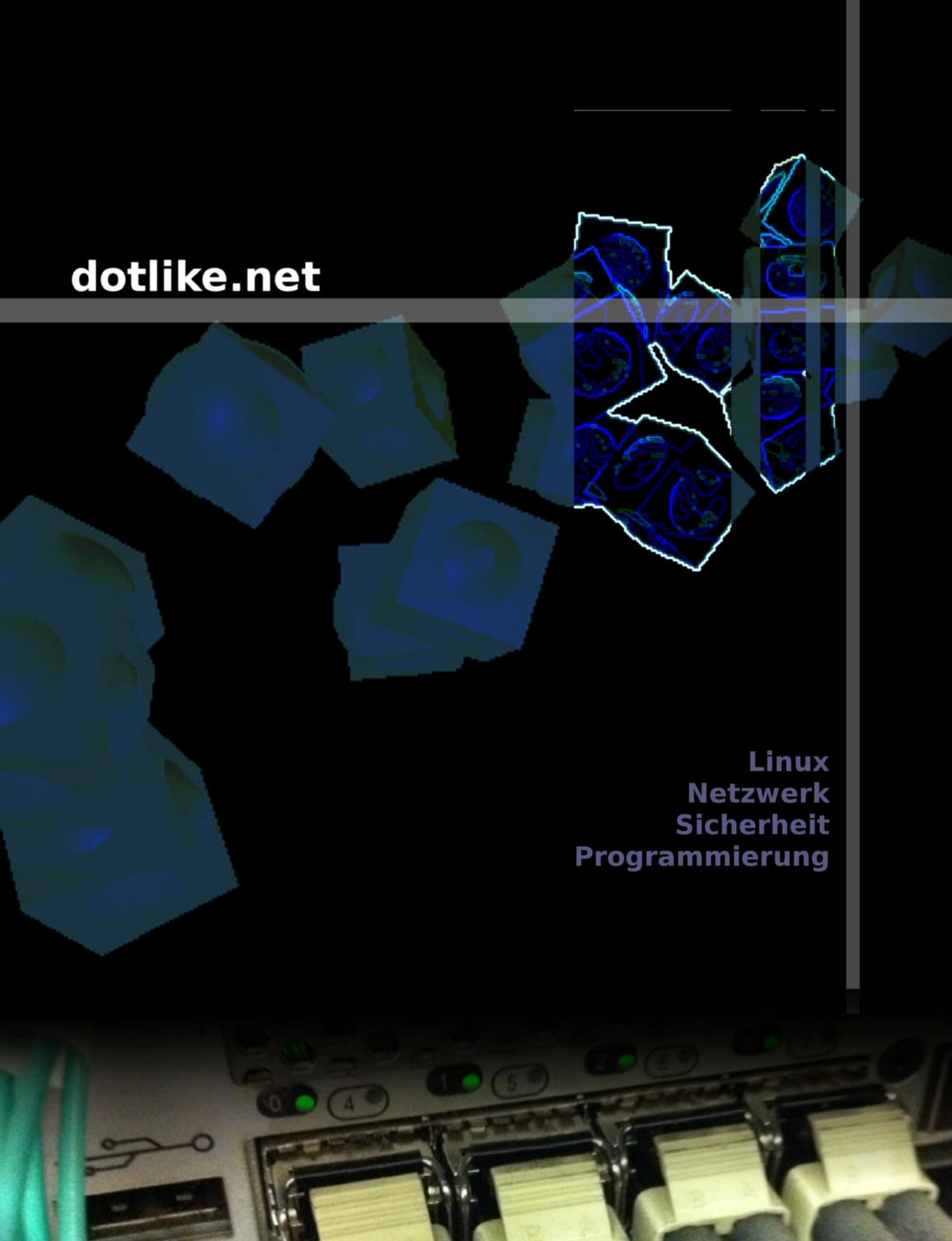
Summary

It has been (and still is) a good experience participating in Google Code-In as a mentoring organization despite the work associated with it. The benefits of getting additional help in many areas that usually do not get that much attention but are otherwise important is very rewarding. I wish all participating students good luck in finishing enough tasks and hope to see them join the projects as contributing members when the contest is over. If you have a good idea for a FreeBSD task, feel free to add it to the FreeBSD wiki or contact one of the mentors.

BENEDICT REUSCHLING

Benedict Reuschling has been using FreeBSD since 5.2.1-RELEASE. He was lurking quietly on the FreeBSD mailinglists until a blog post in 2008 made him join the FreeBSD German Documentation team. After having received his commit bit for the FreeBSD project's documentation set as well, he's been involved in many aspects of documentation around FreeBSD. Having worked in the private sector at a number of companies during his time as a student, he now enjoys teaching students at the Department of Computer Science where he spent so much time of his education. Learning and practicing the Tai Chi Yang style is helping him switch his internal processor off during spare times.

dotlike.net



Linux
Netzwerk
Sicherheit
Programmierung

Installing PC-BSD on a Mac

Starting with PC-BSD 9.0-RC1, it is now possible to easily install directly to a Mac or MacBook BootCamp partition. In this article we will take a quick look at how to setup the Mac for dual-booting, and perform the installation.

To begin the process of configuring your Mac for dual-booting, you must first ensure that you are on an Intel-based Mac, running OS X 10.5 (Leopard) or later. Next, click the search button in the top right corner of the system and type *boot*. Locate and click on the *Boot Camp Assistant* program. (It can also be found in *Applications › Utilities › Boot Camp Assistant*).

After starting the Boot Camp program, click continue and select *Create or remove a windows partition*.

Next you need to allocate some disk space for your new PC-BSD partition. Drag the circle divider to create a windows partition of the desired size. When finished click the *Partition* button.

After the partitioning is finished, click the *Quit & Install Later* button to exit the boot-camp utility.

With your partitioning finished, the next step is to insert your PC-BSD DVD/CD and reboot your system. Press and hold the *C* key to boot from the CD/DVD drive. Once you boot into the PC-BSD installer, you can proceed with a normal installation until you reach the disk setup screen. On the disk screen, it is important to select the correct Boot Camp partition to do the installation. On a Mac with a single volume and the Boot Camp partition, this will show up as *ada0p3 – linux-data*. If in doubt, confirm that the partition sizes match what you just created in Boot Camp. With that partition selected, you can simply click

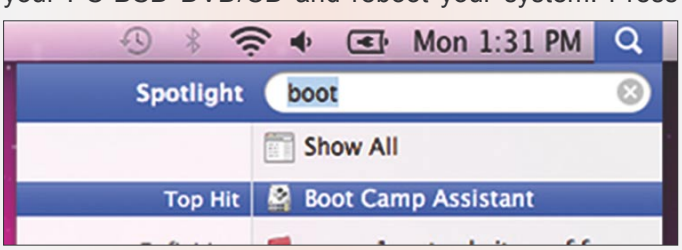


Figure 1. *Starting BootCamp From OSX*

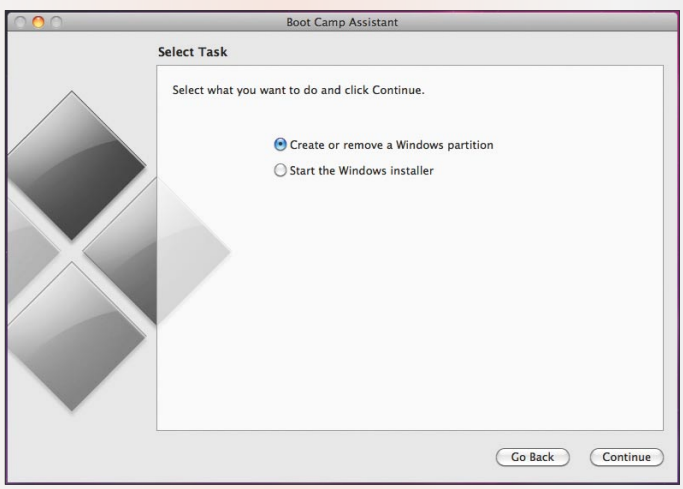


Figure 2. *Creating a New BootCamp Partition*

Next to continue with the installation. After selecting the rest of your installation options and finishing the PC-BSD setup, you can click *Finish* to reboot the system. Before the Apple logo shows up, press and hold the *Option* key to bring up the Mac *Startup Manager*. From this screen you can select the new BSD partition to boot into your desktop.

If the *Startup Manager* does not see your BSD partition or you don't want to remember to hold the *Option* key during startup to boot into PC-BSD, you may need to install a custom boot manager such as rEFIt. rEFIt can be freely downloaded from their website and installed on your system replacing the default boot manager. Once rEFIt is installed, at bootup time you will be presented with icons to select the system volume you wish to load. Choose the *BSD* volume to begin booting into the PC-BSD desktop.

If you later need to remove the PC-BSD Boot Camp partition and return the disk space to your OSX volume, there are only a few steps to take. First you will need to

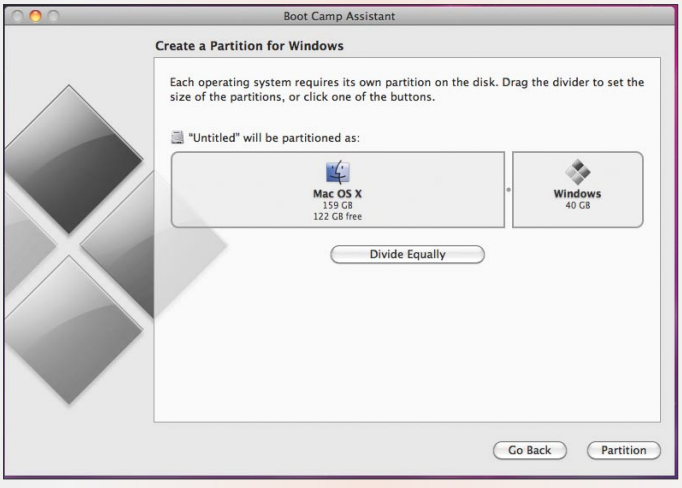


Figure 3. *Setting The Size For The Partition*

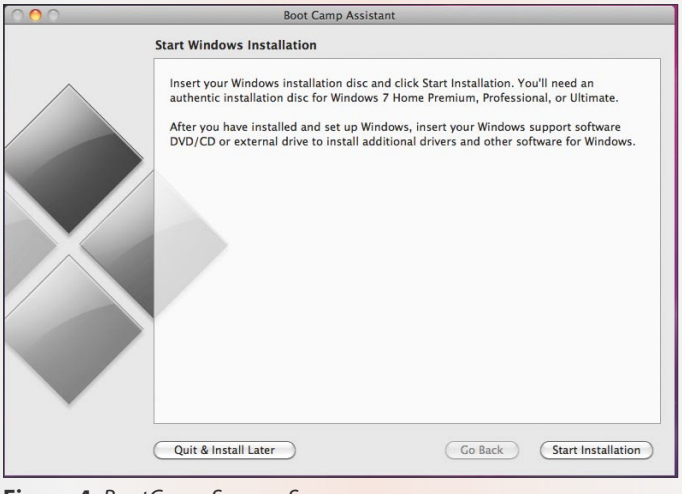


Figure 4. *BootCamp Success Screen*

More Information on the Web

- PC-BSD Installation Guide: *http://wiki.pcbsd.org/index.php/Installing_PC-BSD*
- rEFIt Homepage: *http://refit.sourceforge.net*

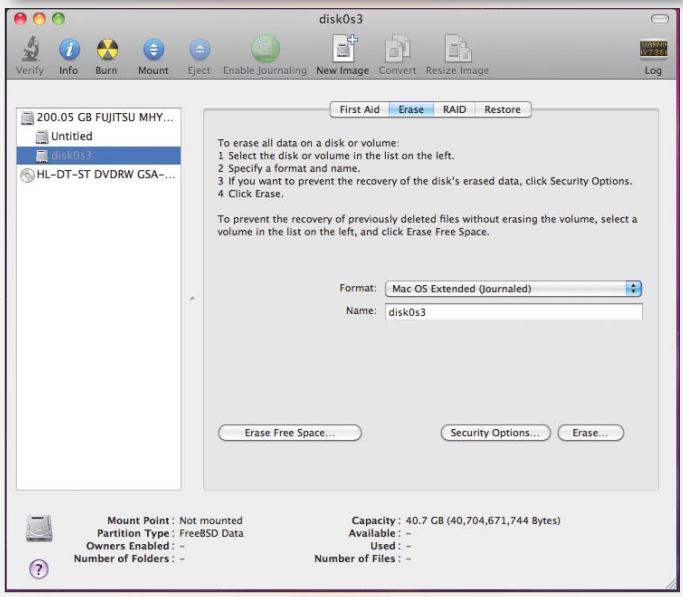


Figure 5. *Erasing The BootCamp Partition*

bring up the *Disk Utility* under *Applications › Accessories › Disk Utility*. Locate the PC-BSD / Boot Camp partition, and select the *Erase* tab. On this screen change the *Format* option to *MS-DOS (FAT)* and click *Erase*.

Now re-run the Boot Camp Assistant program. It will then walk you through the process of removing the partition and restoring the disk space to your existing OS X volume.

KRIS MOORE

Kris Moore is the founder and lead developer of PC-BSD. He lives with his wife and four children in East Tennessee (USA), and enjoys building custom PC's and gaming in his (limited) spare time. kris@pcbsd.org

Keeping Your Configuration

Files Shiny As New Using sysmerge(8)

"I've just upgraded to a new release... now what?"

What you will learn...

- sysmerge(8) history and internals
- sysmerge(8) usage and best practices

What you should know...

- OpenBSD upgrade mechanism
- Bourne shell

Unlike most Linux distributions and similarly to other BSD systems, OpenBSD is not built from a collection of packages. Instead, it is bundled as a coherent whole, providing a complete Operating System extracted from a handful of tarballs. That makes it a very easy system to upgrade (most of the time, it's just a matter of extracting the sets). However this works fine for binaries, libraries, man pages... but one does not want to just extract the default configuration components (i.e. *etcXX.tgz* and *xetcXX.tgz*) as it would obviously overwrite any local change(s).

In the past, updating configuration files would either require a *patch* file (provided on the *OpenBSD upgradeXX.html* page) which would update some of the files that would usually not be modified by the admin, or one would have to manually merge the changes between the old and new versions... which was cumbersome.

Having a tool that would help the administrator update his configuration in a fast, easy and thorough way after upgrading from one release to another didn't exist at that time and it was the reason sysmerge(8) was created.

Note that sysmerge(8) only deals with the *base system* files, it does not touch any third parties files like the ones coming from ports(7)/packages(7).

Past

When sysmerge(8) was introduced, several people where using the mergemaster(8) package (which was ported over from FreeBSD) as a tool to update files under `/etc` and `/var` (some configuration files for chrooted daemons like named(8) or httpd(8) are located under `/var`).

It worked quite well but had some drawbacks: it was targeted at FreeBSD (obviously), had a huge amount of options, was very big script and was not part of the base system.

So it was decided not to use mergemaster(8) but rather write a replacement that would somehow work in the same way but better suits our needs and requirements.

sysmerge(8) was initially added during the 4.4 release development (about 3.5 years ago).

At that time it was mostly a simplified mergemaster(8) only able to diff files and present you with an sdiff(1) interface when they mismatched.

Nowadays, we are still using some important parts from mergemaster(8) like the compare and diff loops.

sysmerge(8) also took some inspiration from tools like etcupdate from NetBSD or *etc-update* from Gentoo.

Present

At the time of this article, sysmerge(8) is exactly *714 lines longs* (mergemaster(8) is about the double). Both utilities share most of the same features but mergemaster(8) makes a lot of them optional using knobs while we prefer sysmerge(8) to be as simple as possible without having to deal with an insane amount of options (Listing 1).

sysmerge(8) does all of its work under WRKDIR, which default to `/var/tmp/sysmerge.XXXXXXXXXX`.

The new reference files are installed under `TMPDIR/temproot/` and a backup of all replaced files is stored under `TMPDIR/backups/`.

A summary log file is available under `WRKDIR/sysmerge.log`.

While all options are detailed in the sysmerge(8) manual, let's talk about some common usages. There are 3 ways of running sysmerge(8): without any option (the *default* mode), in *batch* mode (`-b`) or in full *diff* mode (`-d`). Although the batch mode can be combined with the full diff one, it wouldn't make much sense as we will see below.

Comparison is made between the currently installed files and what are called the reference ones (the (x)etcXX sets or the OpenBSD source directory of the new release).

So, each of the modes will run against a defined set of reference files.

- `-s` is used to specify the path to either the *etcXX.tgz* set or the OpenBSD source tree (usually `/usr/src`)
- `-x` is used to specify the path to the *xetcXX.tgz* set

These switches are optional: sysmerge(8) will default to use `/usr/src` if they are not set (i.e. it will run `` `sysmerge /usr/src` ``).

Sanity checks are performed to make sure that ownerships and permissions of updated and/or newly installed files and links match the reference ones.

At the end of any sysmerge(8) invocation, mtree(8) is run to make sure no directories are missing and that they all have correct permissions.

Note that due to the specific nature of the following files, they will always be ignored from comparison:

- `/etc/*.db` – passwd(1) and `login.conf(5)` DB files created from corresponding utilities
- `/etc/mail/*.db` – sendmail(8) DB files created from flat text files by makemap(8)
- `/etc/passwd` – generated by `pwd_mkdb(8)` from `master.passwd(5)`
- `/etc/motd` – no point in touching this file
- `/etc/myname` – FQDN of the machine, no need to overwrite this
- `/var/db/locate.database` – locate(1) DB, generated by locate.updatedb(8)
- `/var/db/sysmerge/{etc,src,xetc}sum` – checksums created and used by sysmerge(8)
- `/var/games/tetris.scores` – no need to overwrite high-scores ;-)
- `/var/mail/root` – post-installation mail to root

Some files and/or directories can be added to this *ignore* list. Adding an entry to the `/etc/sysmerge.ignore` file (one per line) will make sysmerge(8) skip it from comparison which may be handy on some setups (`/etc/pf.conf` could be a candidate).

When the following files are installed and/or updated by sysmerge(8), their corresponding utility is instantly triggered so that values between the flat files and the DB stay consistent:

- `/dev/MAKEDEV` – MAKEDEV(8)
- `/etc/login.conf` – cap_mkdb(1) (*only* when `/etc/login.conf.db` already exists)
- `/etc/mail/access`, `genericstable`, `mailertable`, `virtusertable` – makemap(8)
- `/etc/mail/aliases` – newaliases(8)
- `/etc/master.passwd` – pwd_mkdb(8)

Running in default mode

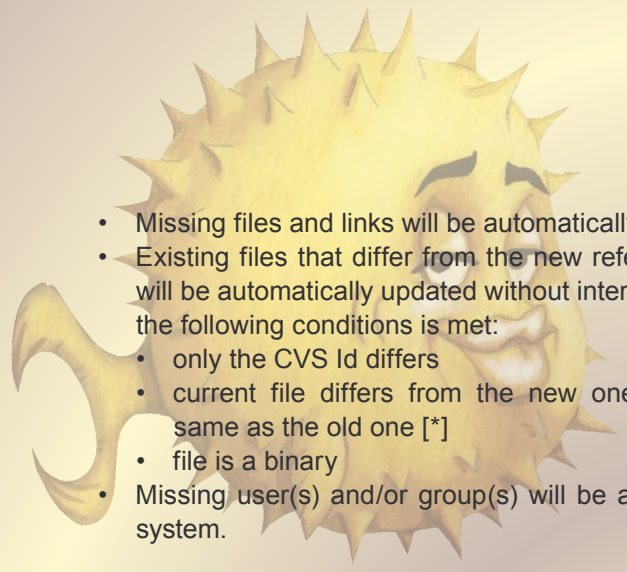
This is what people will want to run most of the time. In this mode, sysmerge(8) does a lot of behind the scenes actions to help update the system without the need of too much interaction with the administrator:

- Files that have the same CVS Id will be skipped from comparison (i.e. added to the *ignore* list).
- There is a special check for sendmail(8) configuration files (i.e. dot.cf files) because they include their build date and directory which mean they will always be different than the one installed on the currently running system; so sysmerge(8) will add them to the *ignore* list unless they *really differ.*

Listing 1. *Comparison of sysmerge and mergemaster usage*

```
Usage: mergemaster [-scrvhpCP] [-a|[-iFU]] [--run-updates=always|never]
    [-m /path] [-t /path] [-d] [-u N] [-w N] [-A arch] [-D /path]

usage: sysmerge [-bd] [-s [src | etcXX.tgz]] [-x xetcXX.tgz]
```



- Missing files and links will be automatically installed.
- Existing files that differ from the new reference ones will be automatically updated without interaction if one the following conditions is met:
  - only the CVS Id differs
  - current file differs from the new one but is the same as the old one [*]
  - file is a binary
- Missing user(s) and/or group(s) will be added to the system.

[*] this is done using checksums; currently installed files have they checksums stored under /var/db/sysmerge/{etc,src,xetc}sum and these are compared against the new reference ones under WRKDIR; this idea originally came from mergemaster(8).

If an automatic decision cannot be made, sysmerge(8) will present the admin with an sdiff(1) interface so that manual merging can be done (sdiff(1) can also be skipped for the administrator to manually merge files later; this is true in all modes).

Running in batch mode

In this mode, sysmerge(8) will behave like in default mode (unless -d is also specified) except that interactive merging (i.e. sdiff(1)) will be skipped. An administrator can then manually merge these files later, they can be found under the TEMPROOT. This mode is particularly useful when running unattended updates on lots of machines.

Running in full diff mode

In this mode, sysmerge(8) makes no automatic action and is fully interactive. Missing files will not be installed without explicit consent from the administrator and any differing file will need to be merged interactively.

While most of the users would not need to do this, running sysmerge(8) this way can be useful to make the system configuration as close as a new installation would.

Usage

sysmerge(8) is meant to be run after each upgrade of the system; from release X.Y to release X.Z or from one snapshot to the other.

It is very important to give it the corresponding reference set according to the way the system was upgraded:

- upgrade was done by rebuilding from source › run sysmerge(8) against /usr/src
- upgrade was done using sets (bsd.rd) › run sysmerge(8) against (x)etcXX sets

e.g. after upgrading your system from OpenBSD 4.9 to 5.0, reboot the machine, get the *(x)etc50.tgz* sets from your favorite mirror then:

```
$ sudo sysmerge -s /path/to/etc50.tgz -x /path/to/xetc50.tgz
```

Listing 2. *Typical sysmerge output*

```
# sysmerge -s etc50.tgz -x xetc50.tgz
===> Populating temporary root under /var/tmp/
                    sysmerge.PLMRVMNTiT/temproot
===> Starting comparison
===> Updating /etc/changelist
===> Updating /etc/login.conf
===> Updating /etc/moduli
===> Updating /etc/netstart
===> Updating /etc/protocols
===> Updating /etc/rc
===> Updating /etc/rc.conf
===> Updating /etc/rc.d/amd
===> Updating /etc/rc.d/ftpd
===> Updating /etc/rc.d/identd
===> Updating /etc/rc.d/ldapd
===> Updating /etc/rc.d/mopd
===> Updating /etc/rc.d/ntpd
===> Updating /etc/rc.d/rarpd
===> Updating /etc/rc.d/rc.subr
===> Updating /etc/rc.d/smtpd
===> Updating /etc/rc.d/ypbind
===> Updating /etc/rc.d/ypldap
===> Updating /etc/rc.d/ypserv
===> Updating /etc/services
===> Updating /etc/ypldap.conf
===> Updating /etc/X11/app-defaults/XTerm
===> Updating /etc/X11/xdm/Xaccess
===> Updating /etc/X11/xdm/Xresources
===> Updating /etc/X11/xdm/Xsession
===> Updating /etc/X11/xdm/Xstartup
===> Updating /etc/X11/xdm/xdm-config
===> Comparison complete
===> Checking directory hierarchy permissions (running
                    mtree(8))
===> Output log available at /var/tmp/
                    sysmerge.PLMRVMNTiT/sysmerge.log
        *** WARNING: some new/updated file(s) may
                    require a reboot
```

Of course one would only need to append *xetc50.tgz* if the X(7) sets are installed on the machine.

It can also work against reference files hosted remotely.

```
$ sudo sysmerge -s http://ftp.fr.openbsd.org/pub/
   OpenBSD/5.0/i386/etc50.tgz
```

Upgrade example

So now let's see a typical real world example and let's run sysmerge(8) after we upgraded a machine from release 5.0 to 5.0-current. We will be using the *etc50.tgz* and *xetc50.tgz* sets downloaded from the *snapshots* directory of an OpenBSD mirror (Listing 2).

Pretty easy since no manual merging was needed :-)

Listing 3. *Content of the sysmerge log file*

```
===> Automatically installed file(s)
/etc/changelist
/etc/login.conf
/etc/moduli
/etc/netstart
/etc/protocols

/etc/rc.conf
/etc/rc.d/amd
/etc/rc.d/ftpd
/etc/rc.d/identd
/etc/rc.d/ldapd
/etc/rc.d/mopd
/etc/rc.d/ntpd
/etc/rc.d/rarpd
/etc/rc.d/rc.subr
/etc/rc.d/smtpd
/etc/rc.d/ypbind
/etc/rc.d/ypldap
/etc/rc.d/ypserv
/etc/services
/etc/ypldap.conf
/etc/X11/app-defaults/XTerm
/etc/X11/xdm/Xaccess
/etc/X11/xdm/Xresources
/etc/X11/xdm/Xsession
/etc/X11/xdm/Xstartup
/etc/X11/xdm/xdm-config

===> Backup of replaced file(s) can be found under
/var/tmp/sysmerge.PLMRVMNTiT/backups
```

Sometimes, like this is the case here, `sysmerge(8)` will warn you that you should reboot. This will appears when `/dev/MAKEDEV` and/or `/etc/login.conf` have been updated. The reason is that running MAKEDEV(8) will reset some TTY permissions which may lead to strange behavior on a running system. When `login.conf(5)` is modified it is usually easier to just reboot the machine to make sure all processes get their new limits.

Now, let's have a look at the log file (this file is very handy when running in batch mode where console output might not be available; Listing 3).

This is a summary log file that records which actions where taken by sysmerge(8). It also gives you the location of the *backups* directory; one can always make a mistake and overwrite a file that wasn't supposed to be.

From there, you are finished and are the owner of a brand new and shinny configuration :-)

Future

While sysmerge(8) is now fully featured, development hasn't stopped. Several enhancements are under considerations to make things even less interactive.

One particular area that is currently investigated is the possibility of using a 3-way merge, using the old sets, the new ones and the currently installed files. This would allow to update locally modified files automatically but triggers new challenges, as we want to make sure no local modifications nor existing behavior of the corresponding applications gets changed. There is no code at the moment, barely some notes and ideas...

Another area of improvement that was committed then disabled is to integrate sysmerge(8) directly into the OpenBSD installer. That would allow the administrator to merge his configuration files right after the binary upgrade, saving an extra reboot of the machine.

This change was successfully tested but eventually got reverted because of some display issues on a couple of legacy architectures when using a serial console. Hopefully this will be worked on soon and enabled again.

On and on I'm very happy with the current state of sysmerge(8), it is a small yet very handy tool and most of the old-timers that used to run their own upgrade script finally came into using it.

ANTOINE JACOUTOT

Antoine Jacoutot is an OpenBSD developer living in Paris, France. He is responsible for more than 300 packages, wrote the sysmerge(8) utility and was part the OpenBSD rc.d(8) framework development. He is also a GNOME committer and member of the GNOME foundation. He runs OpenBSD for pretty much everything.

Rolling Your Own FreeBSD Kernel

Compiling a custom kernel has its own advantages or disadvantages. However, new users may find it difficult to compile a FreeBSD kernel.

What you will learn…

- Why you would want to build your kernel.
- How to build a FreeBSD kernel.
- How to create a BSD configuration file.
- How to tune a FreeBSD kernel.
- How to add a FreeBSD device driver.
- Brief discussion on loadable kernel modules in FreeBSD.

What you should know…

- Basic knowledge of FreeBSD.

When compiling a kernel, you need to understand a few things other then just typing a couple of commands. In this article, I cover the nuts-and-bolts of compiling a FreeBSD kernel.

Why Configure the Kernel?

When the system is installed, it comes with a generic kernel that's designed to run on most any hardware. The generic kernel includes many different device drivers and option packages. Since the kernel only needs to run on your particular system, it's a good idea to reconfigure it to get the modules you won't be using and to turn off options that don't interest you.

Building a kernel tailored to the system is a good habit to get into. Your well-tuned configuration, once attained, can serve as a reference guide for the system's hardware. Although unused features and drivers might not interfere directly with the operation of the system, they can still consume memory.

Modern kernels are better than their predecessors at flushing unwanted drivers from memory, but compiled-in options will always be turned on.

Although reconfiguring the kernel for efficiency reasons is less important than it used to be, a good case can still be made for doing so.

Another reason to configure the kernel is to add support for new types of devices (i.e., add new device drivers). The driver code can't just be mooshed onto the kernel like a gob of Play-Doh; it has to be integrated into the kernel's data structures and tables. On some systems, this procedure may require that you go back to the configuration files for the kernel and add in the new device, rebuilding the kernel from scratch. On other systems, you may only need to run a program designed to make these configuration changes for you.

Some systems include the concept of a *loadable* device driver, in most cases implying that new code can be loaded into the kernel while it is running. A good human analogy might be having brain surgery while operating heavy machinery.

Building a kernel is not difficult; it's just difficult to fix when you break it.

Building a FreeBSD Kernel

Although the examples in this article are specifically for FreeBSD, configuration for NetBSD is similar. OpenBSD has provided some valid arguments on why you should or should not customize your kernel. Instead of listing them here, please refer to: *http://www.openbsd.org/faq/faq5.html#Why*.

BSD kernels each have a name that is used through the configuration process. The kernel name can be anything you like, but it should be descriptive of the system or systems on which the kernel is to run. If the kernel is being

built for one particular machine, that machine's hostname makes a good kernel name.

To build a FreeBSD kernel, you first create a configuration file that lists the parameters of the new kernel. You then run the `config` command to build a kernel compilation directory as specified in your config file. The name you give the configuration file becomes the name of the compilation directory and ultimately the kernel.

The files needed to build a BSD kernel reside in `/usr/src/sys`, which is usually symbolically linked to `/sys`. In the following discussion, I will use the uppercase name *SYS* to refer to this directory, just to emphasize that it doesn't really matter where it's located. If you cannot find the kernel configuration directory for your machine, consultant your distribution's website.

Here is an `ls -F` of the *SYS* directory for FreeBSD: Listing 1.

The `i386` directory contains architecture-specific modules: Listing 2.

Another important directory in the *SYS* area is `SYS/arch/conf`, where the kernel configuration files are stored; each file corresponds to one kernel. In this article I will assume that you are using the Intel 386 architecture, although FreeBSD also supports the additional architectures such as amd64, ia64, powerpc, sparc64 to name a few. `config` reads a configuration file from `SYS/arch/conf` and creates the corresponding compilation directory in `SYS/compile/KERNEL_NAME`.

For example, when the system is first installed, it comes with a generic kernel named GENERIC. The default kernel configuration file is `SYS/i386/conf/GENERIC`, so that default compilation directory would be `SYS/compile/GENERIC`.

The rest of the directories in `SYS` contain various parts of the kernel that are assembled to create the executable image. The exact files and subdirectories vary widely among BSD systems.

The Master Recipe for Building a Kernel

The following list details the eight steps involved in build a kernel. We take up each of these steps in the subsequent sections.

- Audit the system's hardware.
- Create and edit the kernel's configuration file in `SYS/i386/conf`.
- Run the `config` program from the `conf` diretory.
- Run *make depend* in the compilation directory.
- Build the kernel with *make*.
- Archive the old kernel and install the new one.
- Test and debug the new kernel.
- Document the new kernel.

Audit the System's Hardware

Before you can configure a kernel, you need to know what devices it must handle. Start by taking a hardware inventory of your system. Make a list of all the devices connected to the computer, including:

- Disks and CD-ROM drives, and their controllers
- Network interfaces
- Specialty hardware
- The keyboard and mouse

This hardware audit can be a grueling task in the PC world. PC manufacturers often just give you a packaged machine and don't tell you what kind of hardware is inside; *an Ethernet card* is not enough. There are hundreds of different PC cards marketed under dozens of names. Frequently, the only way to do the audit is to open your machine and look at what you've got. If you think know the name of the device driver, you can look at the man page for that driver. Unfortunately, FreeBSD provides little documentation of the exact devices that a driver supports.

Remember to check what the generic kernel reports about your hardware at boot time; its output can give you hints as to which drivers you should keep in your kernel. You can check the current kernel's idea of your hardware

Listing 1. *Listing of the SYS directory*

```
# ls -F
Makefile   ddb/      libkern/  netnatm/  rpc/
amd64/ dev/      mips/     netncp/      security/
arm/       fs/    modules/  netsmb/   sparc64/
boot/      gdb/      net/      nfs/      sun4v/
bsm/       geom/     net80211/ nfsclient/   sys/
cam/       gnu/      netatalk/ nfsserver/   tools/
cddl/      i386/     netgraph/ nlm/      ufs/
compat/    ia64/     netinet/  opencrypto/  vm/
conf/      isa/      netinet6/ pc98/     x86/
contrib/   kern/     netipsec/ pci/      xdr/
crypto/       kgssapi/  netipx/       powerpc/      xen/
```

Listing 2. *The i386 directory*

```
# ls -F i386
Makefile   compile/  ibcs2/    linux/    xbox/
acpica/       conf/     include/  pci/      xen/
bios/      i386/     isa/      svr4/
```

with the `dmesg` command. You can also use the `SYS/i386/conf/LINT` file as an extra reference.

Create a Configuration File in SYS/i386/conf

Once you know how you want your kernel configured, you must put this information into a form that `config` can understand. To do this, you create a configuration file in `SYS/i386/conf`. The name can be any valid filename, but it should be descriptive enough that a stranger to your *SYS* directory can tell what each kernel is for.

Don't create the configuration file from scratch. Instead, copy the *GENERIC* configuration and delete the parts you don't want. If you get stuck on something related to the configuration file and can't figure it out from the material here, refer to the documentation for *config*. The man pages for individual device drivers are also a good source of information and usually show any kernel config lines you might need. For example, the man pages for `de(4)` begins with

```
SYNOPSIS

device de
```

which is the exact line you need to put in the kernel config file to include that device. (Of course, this method is still a bit backward because you need to know the name of the device driver before you can look up the man page. `man -k` is your friend.)

The format of a kernel configuration file requires quite a few pages to describe, so instead of interrupting our overview of the kernel building process with a complete discussion, I'll defer the details until the next section.

Run config

You must `cd` to `SYS/i386/conf` before running `config`; it expects to find the configuration file specified on the command line in the current directory. Simple versions of `config` take the name of the configuration file as their only argument. Fancier versions support a number of options. To set up the compilation directory for the kernel described in `SYS/i386/conf/EXAMPLE`, we would use the following commands:

```
# cd SYS/i386/conf
# config EXAMPLE
```

If the `config` command produces error messages, you must go back and fix your configuration file before continuing. If you get through `config` without any errors, you can assume that your configuration was at least syntactically valid and that the kernel compile can proceed.

Run make depend

After `config` finishes, change your working directory to the new kernel's compilation directory (`cd ../../compile/EXAMPLE`) and do an `ls`. You should see lots and lots of files. Don't worry about their contents; `config` knows what it's doing.

Now run *make depend* inside the compilation directory. This command initializes the file dependency information used by *make*. *make depend* may produce voluminous output.

Build the Kernel

In the compilation directory, simply type *make*. You must watch carefully for error messages during the compilation. *make* will usually detect errors and abort the compilation, but it always helps to be alert. For extra protection, use the `tee` command to keep a record of everything that gets sent to your screen.

```
# make |& tee ERRS.LOG
```

The `&` behind the vertical bar ensures that both error messages and status messages will be directed through the pipe. Bourne shell users should use

```
# make 2>&1 | tee ERRS.LOG
```

If an error occurs during compilation, you should first suspect your configuration file. If you get messages about missing files or undefined routines, you have probably left something out of the config file. If you get messages complaining about syntax errors, the fault may be with your configuration file or with the system, although the latter is not likely.

Install the New Kernel

Before you boot a new kernel, make sure you can recover your system if the new kernel doesn't work. Never replace the old kernel directly with a new one, because you will then have nothing to boot from in the event of a catastrophe. Traditionally, kernels have been called `/vmunix`, but every OS seems to call them something different these days. Under FreeBSD, the kernel is `/kernel`.

You should back your old kernel by moving `/kernel` to `/kernel.works`. All systems provide some way to keep an old kernel bootable while you test a new one. You could do this with boot loaders, but that's a topic for a different article.

`/kernel` can be a hard link to some other filename, so you can just make a link to your new kernel rather than copying it. If the kernel is not called `/kernel` and you don't make this link, the boot loader will have difficulty finding it.

Test the New Kernel

If the system boots successfully, you are probably in good shape. However, you should try a few checks just to make sure. Run `ls` on at least one directory in each filesystem. Success indicates that the filesystem is functioning correctly. *ping* another machine on your network to see if your network device is working properly.

Document the New Kernel

Before washing your hands of this whole sordid kernel business, go back to your original `SYS/i386/conf/KERNEL_NAME` file and put in copious comments so that you will understand what you have done when you come back to read it six months or a year later.

If you have lots of free space, you can preserve the `SYS/compile/KERNEL_NAME` directory to speed up subsequent alterations. If you're tight on space, just delete it; everything it contains can be regenerated with `config`.

Creating a BSD Configuration File

Creating the configuration file (under `SYS/i386/conf`) is the hardest part of building a BSD kernel; the rest of the process quite mechanical.

A configuration file is a list of control phrases, one per line. Any line beginning with a tab character is considered a continuation of the previous line. Anything between a pound sign (`#`) and the end of a line is considered a comment, and blank lines are ignored. Keywords must be separated by whitespace, but except for this and the special meaning of tabs as continuation characters, spaces and tabs are ignored.

Table 1. *Keywords used in BSD configuration files*

Keyword	Function
machine	Sets the machine type
cpu	Sets the CPU type
ident	Sets the name of the kernel
maxusers	Sets the kernel's table sizes
options	Sets various compile-time options
config	Assigns the root and swap areas
controller	Declares a disk or tape controller
disk	Declares a disk connected to a controller
tape	Declares a tape connected to a controller
device	Declares devices without controllers
pseudo-device	Declares pseudo-devices

Integers in the configuration file can be entered in hexadecimal, octal, or decimal form. Octal numbers are identified by a leading zero, and hexadecimal numbers by a leading `0x`. Strings must be double quoted if they contain numbers used as text.

A control phrase begins with a single keyword that indicates how the remainder of the line is to be interpreted. The rest of the line provides the keyword's arguments. Some keywords can accept a list of arguments separated by spaces or commas, but it's wise to use only one argument per line. Most keywords that can accept multiple arguments can also have arbitrarily many control lines.

The order in which control phrases appear is usually not important; Table 1 shows the traditional order: Table 1.

The maxusers Keyword

The *maxusers* keyword sets the size of several important system tables. As its name suggests, the argument to *maxusers* is roughly the maximum number of simultaneous users that the system is expected to support (though most versions of UNIX don't actually enforce a limit on the number of users per se). If you want to tune this value yourself, you should increase it by 1 for each expected simultaneous user and, if you are configuring the kernel for an NFS server, by 1 for each client machine. Add 8 for each frame buffer on which a window system can be run.

The *maxusers* number affects the values of several other kernel parameters, such as the maximum number of processes, the number of file table entries, and the number of buffers for terminal I/O. The most important of these is the maximum number of processes on the system. Here's the formula:

```
Maximum processes = 20 + 16 * maxusers
```

This maximum process count includes the 18 or so processes that start when the system is booted.

The options Keyword

An *options* directive defines variables for the C preprocessor during compilation of the kernel. There are two different forms of the *options* statement.

In the first form, tokens are defined but given no particular value. Such tokens specify whether an option

is on or off using the preprocessor directives `#ifdef` and `#ifndef`. When a token is supplied as an argument to an *options* statement, the corresponding preprocessor symbol is defined and the option is enabled. For example, the phrase to include NFS in the kernel is

```
options      NFS
```

Note that with FreeBSD, any string in the config file containing both letters and numbers needs to be quoted. For example the ISO-9660 filesystem used on CD-ROM is enabled with the following line:

```
options      "CD9660"
```

The second form of *options* statement not only defines a symbol but also gives it a specific value. The kernel code uses the symbol as if it were a constant, and the C preprocessors makes an appropriate substitution wherever the symbol appears. This type of symbol is declared with the syntax:

```
options      symbol="value"
```

For example, to modify the value of the *MAXDSIZ* option, which sets the maximum amount of virtual memory that can be allocated to the data segment of a single process, you would use a line such as:

```
options      MAXDSIZ="(64*1024*1024)"
```

This example sets the value to 64 megabytes.

The most common options are listed below. None of these options take a value. See your vendor's documentation for a complete list.

- `INET` – This option includes networking support. Networking has become so persuasive that a lot of software is likely to break if you don't include it; it's an option in name only. When you enable INET, you should also include the pseudo-device loop. The INET option includes only software-side networking support. Network hardware is declared later in the config file.
- `FFS` – This option allows local disks to be attached to the machine. It's omitted only when an extremely lean kernel for a diskless client or an embedded device is set up.
- `NFS` – This option includes NFS support in the kernel. It's required for both NFS clients and servers.
- `GATEWAY` – This option is for use on machines that have more than one network interface and are intended to perform Internet routing and forwarding functions. This option currently has only minor ramifications: it increases the size of some kernel data structures to cope with the expected load and provides for special network behavior if one of the interfaces goes down.

The config Keyword

The `config` keyword specifies the location of the root partition on the system's disks.

The root partition is the topmost component of the filesystem. It contains the directory root (`/`) and several other important files and subdirectories. Information about how to mount filesystems is kept in the `/etc/fstab` file, but the kernel can't see this file until the root partition has been mounted.

To bootstrap the filesystem, information about the partition that holds the root must either be compiled into the kernel or, on some systems, passed to the kernel by the bootstrap loader. The situation for swapping is not as dire, since it's unlikely that any swapping will occur until the `/etc/rc*` scripts run the `swapon` command.

A config line has the form

```
config kernel_name root on partition
```

The `kernel_name` parameters sets the filename under which the compiled kernel will be stored. FreeBSD kernels are named *kernel*; alternates are often named to identify the disk they use for the root partition (e.g., *dakernel*).

The *partition* parameter tells which partition the root filesystem is located on. The partition is typically `wd0` for IDE systems and `da0` for SCSI systems.

Here's a complete example:

```
config kernel root on wd0
```

The ability to build variant kernels is useful for disaster planning. An alternate root partition equipped with its own kernel can be of great help when your main root partition is damaged. If the alternate root is on the same disk drive or controller as the one that got trashed, be sure to verify the stability of the hardware before rebooting. Otherwise, you run the risk of destroying the alternate root, too.

On some systems, a CD-ROM or USB drive controls the boot procedures. The CD-ROM or USB drive knows the location of the kernel and control its invocation. If you maintain an alternate root partition, you may have to build a new boot CD-ROM or USB drive that uses it.

Hardware Devices

The syntax for declaring devices is confusing, and building the basic entries required to make the system run vary from machine to machine. Section 4 of the BSD manuals covers devices. Most man pages for device drivers list an example config line you can include in the kernel.

Take the following instructions with a grain of salt. I discuss the general syntax, but since I expect that you will mostly be paring down your system's generic configuration, I don't talk about how to write your own device specifications from scratch.

The basic form of a declaration is:

```
device-type device-name at connection-info port address
                [device-class] irq interrupt
```

Not all clauses are applicable to all devices.

device-type is the type of device you are declaring. A few types of devices, such as *controller* and *disk*, have special keywords. Others use the generic keyword *device*.

device-name is the standard name of the device (or more accurately, the name of the device driver), plus the logical unit number. For example, the name for the first IDE controller is `wdc0`. As you wade through the generic configuration, you can look up each device in section 4 of the manuals to find out what it is and whether it applies to you. Note that the logical unit number has no relationship to any hardware-specified selection number of the device.

The *connection-info* for a device tells the kernel where to find the device and what kind of device it is. For disk and tape drives, this connection info is usually the name of a controller. For controllers and devices, it's the name of a bus or bus controller. For example, the following lines define the system's ISA bus, an IDE controller that's attached to it, and an IDE disk that's attached to the IDE controller:

```
controller   isa0
controller   wdc0   at isa? port "IO_WD1" bio irq 14
controller   wd0 at wdc0 drive 0
```

It is usually sufficient to state that a device is connected to a particular type of controller without specifying which one. For example, the location of the `wdc0` IDE controller is indicated above not as `isa0` or `isa1`, but as the more generic *isa*?.

The `address` parameter, the argument to the `port` keyword, represents the location of the device's command and status registers in the address space of the bus or backplane to which it is connected. Controllers and devices connected directly to a bus often have this parameter filled in. Each kind of device has a certain number of address locations that it occupies in the bus's address space. The values only need to be specified for ISA or EISA devices; PCI drivers can dynamically determine the address range a device is using.

Set the *interrupt* to the interrupt request (IRQ) the device has been configured to use. This parameter only needs to be specified for ISA and EISA devices. PCI drivers can dynamically determine the interrupt a device is using.

For some device drivers, you must specify a *device-class*. This parameter is mainly used for network devices and some controllers. To see if a specific device needs a device class, refer to its man page.

Here's the config line for an ISA NE200 network card that uses most of these options:

```
device ed0 at isa? port 0x360 net irq 10
```

This line says to locate the device `ed0` on the ISA bus at I/O address `0x360`. It uses interrupt 10. It's more typical that some keywords can be left out. Here's another Ethernet card, this time on the PCI bus:

```
device de0
```

Thanks to the wonders of PCI, we are not required to specify all the gory details.

The most effective way to organize your declarations is to pair related devices. If you have a controller, put the devices attached to it nearby. For example, if you have an IDE controller, put your IDE disk and CD-ROM declarations right after it. That way it's easier to visualize the dependencies.

The pseudo-device Keyboard

Theoretically, pseudo-device drivers are programs that act like device drivers but don't have any real hardware to back them up. I say *theoretically* because some kernel options that masquerade as pseudo-devices do not act like device drivers at all, at least from the user's point of view. The syntax for a pseudo-device line is

```
pseudo-device    device-name number-of-instances
```

device-name is the name of the pseudo-device and *number-of-instances* is an optional integer telling how many of the imaginary devices the driver should pretend are present. Many drivers do not use the instance count.

There are only a few pseudo-devices, but most of them are obligatory for correct operation of the system. Some

systems have a number of nonstandard pseudo-devices that support windowing systems, extra keyboards, or auxiliary displays. Consult the manuals of your system to learn how to deal with these, or just include all the pseudo-devices from your generic configuration file for a more festive atmosphere.

Some common pseudo-device are:

- `pty` – PTYs are pseudo-terminals. They mimic terminals, but instead of having an actual terminal on one end, they are connected to a UNIX process. PTYs are used heavily by programs such as ssh, xterm, telnet, and rlogin, and they are also used by a few standard utilities such as shell scripts to do input processing.
- `loop` – The loop driver simulates an interface to a network that contains only the local host. It allows stand-alone machines to use network software, and it also provides a standard way for a machine to address packets to itself. It is required if you specify the INET option.

A Sample FreeBSD Configuration File

Let's look at a configuration file for a simple kernel which we'll call EXAMPLE:

```
machine   "i386"
cpu    "I386_CPU"
cpu    "I486_CPU"
cpu    "I586_CPU"
cpu    "I686_CPU"
ident     EXAMPLE
maxusers  32
```

The first few lines specify that we are building a kernel for Intel PCs and that the kernel should support all the different CPU types specified. This section also identifies the configuration with the name EXAMPLE. The `maxusers` line sets the kernel tables up for approximately 32 simultaneous users and 532 simultaneous processes.

```
. . .
options       INET      # Intenet: TCP/IP
options       "CD9660"  # ISO 9660 CD-ROM filesystem
options       FFS    # (FFS) Local filesystem
options       NFS    # Network filesystem
. . .
```

This is just a snippet from the options section of the configuration file. My sample kernel is configured with support for Internet (IP) networking, the ISO-9660 filesystem (used most commonly on CD-ROMs), local filesystems, and NFS.

```
config kernel root on wd0
```

The default root device is the first IDE hard disk.

```
controller isa0
controller pnp0
controller eisa0
controller pci0
```

These lines declare the various buses supported by system: ISA, EISA, and PCI. The second line declares Plug and Pray support for ISA devices (`pnp0`) (Listing 3).

This section declares all items needed to get a console on your machine. It declares the keyboard and its controller, the mouse, the display card, and the console itself (Listing 4).

Listing 3. *Declaring the various buses supported by the system*

```
controller   atkbdc0   at isa? port IO_KBD tty
device       atkbd0    at isa? tty irq 1
device       psm0      at isa? tty irq 12
device       vga0      at isa? port ? conflicts
# splash screen/screen saver
pseudo-devicesplash
# syscons is the default console driver, resembling
                a SCO console
device       sc0    at isa? Tty
```

Listing 4. *Declaring all items needed to get a console on your machine*

```
# Floppy devices
controller   fdc0      at isa? port "IO_FD1" bio
                irq 6 drq 2
disk      fd0    at fdc0 drive 0
disk      fd1    at fdc0 drive 1
# IDE controller and disks
controller   wdc0      at isa? port "IO_WD1" bio
                irq 14
disk      wd0    at wdc0 drive 0
disk      wd1    at wdc0 drive 1
controllerwdc1       at isa? port "IO_WD2" bio irq 15
disk      wd2    at wdc1 drive 0
disk      wd3    at wdc1 drive 1
```

Here I declare the controllers and disks for my example system: a floppy controller, two floppy drives (even though only one is used, declaring them both doesn't hurt), and two IDE controllers with corresponding disks (Listing 5).

Under FreeBSD, I need to include these special options to enable IDE devices. Although IDE can be configured as a loadable kernel module (the *LKM* referred to in the comment), it must be statically configured if you're using an IDE disk as your root partition. Otherwise, the system cannot recognize your root disk at boot time (Listing 6).

Of these pseudo-devices, only `loop` is mandatory. In general, you should retain all the pseudo-devices in the GENERIC configuration. Obviously, you need the `ether` pseudo-device to use Ethernet devices. You need the `bpf` pseudo-device to run `tcpdump` and DHCP clients, so keep it.

You may want to leave out `bpf` support to prevent people from sniffing the network. However, if you omit `bpf`, you won't be able to legitimately snoop to diagnose problems.

Tuning the FreeBSD Kernel

The GENERIC kernel is not tuned for high performance, as you will notice especially if you are building a high-volume web server. Here are some suggestions for building a better FreeBSD kernel.

To some extent, you can dynamically tune the FreeBSD kernel with the `sysctl` command, which provides a user-level interface to many of the kernel's internal data structures. `sysctl` lets you dynamically change selected kernel parameters; it's a powerful (and dangerous) command.

`sysctl -a` lists the kernel variables you can look at and possibly change. Almost all the parameters in Table 2 can be changed dynamically. Documentation on what each variable does is typically meager, although the variable names usually give a hint. Be careful when tuning your kernel with `sysctl` because you can easily break things.

Listing 5. *Declaring controllers and disks*

```
options     ATAPI         # Enable ATAPI support for
                IDE bus
options     ATAPI_STATIC  # Don't do it as an LKM
device      acd0          # IDE CD-ROM
```

Listing 6. *Retain all pseudo-devices in the GENERIC configuration*

```
pseudo-device   loop      # Network loopback
pseudo-device   ether     # Ethernet support
pseudo-device   bpf 4     # Berkeley packet filter
```

The changes that `sysctl` makes to kernel variables are not remembered across reboots. A useful paradigm is to first test changes by using `sysctl`, then make them permanent by changing the kernel config files and recompiling the kernel. This procedure has the advantage of safety, since you can simply reboot to reset the system no matter how much you've screwed things up.

Table 2 lists the most commonly tuned `sysctl` variables, their default values, and their meanings.

Note that in the default configuration, a single user can consume all but one of the system's process slots. Even if only one person will actually use the system, these defaults create a potential problem because no headroom is reserved for starting system processes. You should make the gap between `maxproc` and `maxprocperuid` much larger than the default.

Below, I describe some simple parameters you might change in your kernel config to get better performance out of the GENERIC kernel. These tweaks are designed specifically for use on a web server, although they should increase performance for most network servers.

```
maxusers 256
```

The *maxusers* keyword adjusts many other variables in the kernel, such as the maximum number of processes, the maximum number of processes per user, the system-wide limit on open files, the per-process limit on open files, and the maximum number of network buffers. Be generous when configuring a server.

```
options     NMBCLUSTERS=17536
```

Here, I set the number of network buffers to a higher value.

Table 2. *Interesting FreeBSD kernel variables accessible through sysctl*

Variable	Default	Meaning
kern.maxfiles	3912	Maximum # of open files
kern.maxproc	1956	Maximum # of processes
kern.maxfilesperproc	3520	Maximum # of open files per process
kern.maxprocperuid	1760	Maximum # of processes per uid
kern.ipc.nmbclusters	8768	Maximum # network buffers
kern.ipc.maxsockets	8768	Maximum # of available sockets
kern.ipc.somaxconn	128	Maximum # of simultaneous unaccepted sockets

```
options        CHILD_MAX=1024
```

This option sets the maximum number of child processes on the system. The number should be high on a network server. In general, network server daemons create a child process for each incoming request.

```
options        OPEN_MAX=1024
```

This option sets the maximum number of file descriptors on the system. This number should generally be the same as `CHILD_MAX`, since each incoming network connection gets assigned its own file descriptor. Given the way that network servers generally work, if you have fewer file descriptors than child processes, you will be limited by the number of file descriptors; the converse is also true.

Adding a FreeBSD Device Driver

Adding a completely new device driver to a FreeBSD machine involves adding it to a couple of configuration files and editing the kernel source code to include references to the driver's routine. This procedure is not for the faint of heart!

I will use a FreeBSD system as my example, but all BSD systems (including NetBSD and OpenBSD) are essentially similar, except that the locations of files may differ. I will add a *snarf* device (a pseudo-device) for my example.

First, I have to copy my source files to the proper location:

```
# cp ~ammann/snarf.c /sys/pci/snarf.c
```

Since my device is a PCI device, I'll put the source files in `SYS/pci` with all the other PCI drivers. If your device doesn't fall into an existing category, you'll have to put it in a new directory and edit `SYS/i386/conf/files.i386`. As long as the driver belongs to an existing category, it will automatically be compiled and linked into the kernel.

Next, I add the device to the kernel configuration file. I put the following entry in my EXAMPLE configuration:

```
device         snf0   # Snarf, my fake network device.
```

This line instructs the `config` program to include the files for the driver in the kernel. Since network devices do not have major and minor numbers, we do not need to tell the kernel what the numbers are. If I was adding a block device or a character device, I would have to tell the kernel which major and minor numbers to use.

References

Chapter 9 of the FreeBSD Handbook.
http://www.freebsd.org/doc/handbook/kernelconfig.html

To tell the kernel which major number to use, edit `SYS/i386/conf/majors.i386` and add the appropriate entry. What *the appropriate entry* means varies from device to device. Refer to the driver's documentation.

The next steps include:

- Running `config` and building a new kernel
- Copying the old kernel aside and installing the new kernel
- Rebooting and testing the new kernel

These steps were explained earlier in this article. Finally, you may need to create device files and test the device itself.

Device Files

By convention, device files are kept in the `/dev` directory. Large systems, especially those with networking and pseudo-terminals, may support hundreds of devices.

Device files are created with the `mknod` command, which has the syntax

```
mknod filename type major minor
```

where *filename* is the device file to be created, *type* is `c` for character device or `b` for `a` block device, and *major* and *minor* are the major and minor device numbers. If you are creating a device file that refers to a driver that's already present in your kernel, check the man page for the driver to find the appropriate major and minor device numbers (in section 4 for FreeBSD).

Loadable Kernel Modules in FreeBSD

The FreeBSD modload, modstat, and modunload commands manipulate kernel modules. You will no doubt be shocked to discover that these commands load a module, display module status, and unload a module, respectively. Each utility performs `ioctls` on `/dev/lkm`.

By default FreeBSD's kernel modules live in `/modules`. If something can be added as a kernel module, it will found there. Any of the modules listed in `/modules` can be inserted using the aforementioned utilities.

PAUL AMMANN

Paul Ammann lives in Connecticut and recently converted to FreeBSD.

FreeBSD Mall

Your FreeBSD & PC-BSD Resource

www.FreeBSDMall.com

FreeBSD 8.2 Jewel Case CD/DVD

Set contains:

- **Disc 1**: Installation Boot (i386)
- **Disc 2**: LiveFS (i386)
- **Disc 3**: Essential Packages (i386)
- **Disc 4**: Essential Packages (i386)

FreeBSD 8.2 CD $39.95
FreeBSD 8.2 DVD $39.95
FreeBSD 7.4 CDROM $39.95
FreeBSD 7.4 DVD $39.95

FreeBSD Subscriptions

Save time and $$$ by subscribing to regular updates of FreeBSD!

FreeBSD Subscription , start with CD 8.2 ..... $29.95
FreeBSD Subscription, start with DVD 8.2 ..... $29.95
FreeBSD Subscription, CD 7.4 ..... $29.95
FreeBSD Subscription, DVD 7.4 ..... $29.95

PC-BSD 8.2 DVD (Hubble Edition)

PC-BSD 8.2 DVD $29.95
PC-BSD Subscription $19.95

BSD Magazine

BSD Magazine $11.99

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide) $39.95
The FreeBSD Handbook, Volume 2 (Admin Guide) $39.95
★ Special: The FreeBSD Handbook, Volume 2 (Both Volumes) $59.95
★ Special: The FreeBSD Handbook, Both Volumes, & FreeBSD 8.2 ..... $79.95

The FreeBSD Bundle

Inside the Bundle, you'll find:

- FreeBSD Handbook, 3rd Edition, Users Guide
- FreeBSD Handbook, 3rd Edition, Admin Guide
- FreeBSD 8.2 4-disc set
- FreeBSD Toolkit DVD

★ Special: The FreeBSD CD Bundle $99.95
★ Special: The FreeBSD DVD Bundle $99.95

The FreeBSD Toolkit DVD $39.95

FreeBSD Mousepad $10.00

FreeBSD Caps $20.00

PC-BSD Caps $20.00

For *MORE* FreeBSD & PC-BSD items, visit our website at FreeBSDMall.com!

CALL 925.240.6652 Ask about our software bundles!

NEW APPAREL!

OpenBSD 5.0:

PHP, Cacti, and Symon

Back in October, I gave instructions on how to create an OpenBSD-Nginx-MySQL-PHP (ONMP) server. Upgrading from OpenBSD 4.9 to OpenBSD 5.0 was difficult.

What you will learn…

- New PHP changes in OpenBSD 5.0.
- How to get a basic Cacti server running.
- How to monitor your OpenBSD server with Symon.

What you should know…

- The OpenBSD command line.
- The differences between Apache & Nginx.
- A basic understanding of what SNMP is.

The reason is that OpenBSD 5.0 changes some of the names and locations of PHP files. OpenBSD 5.0 supplies packages for both PHP 5.2 and 5.3. From package names to directories to binaries, everything PHP is now `...php-5.2...` or `...php-5.3...`. Also, the configuration files are moved from `/var/www/conf` to `/etc`. Here are some highlights (substitute 5.2 for 5.3 if you're using that version – I'm using 5.3):

```
/usr/local/bin/php is now /usr/local/bin/php-5.3
/usr/local/bin/php-fastcgi is now /usr/local/bin/php-fastcgi-5.3
/var/www/conf/php.ini is now /etc/php-5.3.ini
```

Now that OpenBSD supports both PHP 5.2 and 5.3, we typically want to install packages in interactive mode (otherwise, `pkg_add` will tell us that the names of PHP related packages are ambiguous).

Here is the new way to invoke `php-fastcgi-5.3` in your `/etc/rc.local` file: Listing 1.

Lastly, OpenBSD 5.0 offers the Nginx 1.0.8 package. If you're trying to follow my ONMP article from October, then you can scrap all the stuff about compiling Nginx from source. The new Nginx HTML root is at `/var/nginx/html/`, and the new Nginx config files are at `/etc/nginx/`.

For more information about building an ONMP 5.0 server, you are free to look at my notes here: *http://toby.org.org/wordpress/index.php/getting-started/openbsd-5-0/*.

Let's install Cacti, which can show graphs from just about any data source. You can graph weather metrics, stocks, or even the number of times *I Love Lucy* reruns are shown on any given day. You just need a data source.

First we need SNMP, RRDTool, and PHP-GD:

```
# pkg_add -i net-snmp php-snmp rrdtool php-gd
# cp /etc/php-5.3.sample/snmp.ini /etc/php-5.3
# cp /etc/php-5.3.sample/gd.ini /etc/php-5.3
```

Now add this to your `/etc/rc.local` so that we can get SNMP data from our server (for seeing CPU utilization, RAM, free disk space, etc.):

```
if [ -x /usr/local/sbin/snmpd ]; then
        /etc/rc.d/snmpd start
fi
```

Now you can follow the directions on *http://cacti.net* to install Cacti. There's no OpenBSD package, so use the tarball on the web site. I'm not going to rewrite the Cacti documentation here; however, you will find corrections to OpenBSD specific discrepancies below.

First, the documentation doesn't tell you where to put the files. I assume that you know enough to put them into your HTML root. For example:

```
# tar xvfz cacti-0.8.7h.tar.gz
# mv cacti-0.8.7h /var/nginx/html/cacti
```

The Cacti documents say to add this to your crontab, but it is…

WRONG:

```
*/5 * * * * cactiuser php /var/nginx/html/cacti/poller.php
                    > /dev/null 2>&1
```

RIGHT:

```
*/5 * * * * /usr/sbin/chroot -u cactiuser / /usr/local/
                    bin/php-5.3 \
/var/nginx/html/cacti/poller.php > /dev/null 2>&1
```

After installing Cacti, there are two files that need tweaking. Cacti tries to modify PHP's memory settings. OpenBSD uses Suhosin to harden PHP. Suhosin doesn't allow this. It may not be a big deal under normal circumstances, but if you've also installed a host intrusion prevention system such as OSSEC, then you're likely to get locked out of your own server. Comment out the following line in both `/var/nginx/html/cacti/cmd.php` and `/var/nginx/html/cacti/poller.php` (searching for `"512"` ought to bring you to the correct line):

```
ini_set("memory_limit", "512M");
```

OpenBSD doesn't like the way that Cacti checks whether the local host is up via SNMP. I've found that configuring Cacti to use UDP Ping instead of SNMP as the *Ping method* for localhost will get your graphs going. You'll find the setting in the web administration section of Cacti.

Here is a sample `/etc/snmpd.conf` to get you going. This is probably not a very secure SNMP configuration, but hey, we're only listening on 127.0.0.1 anyway.

Listing 1. *Starting PHP-Fastcgi*

```
chroot -g nobody -u nobody / env -i PHP_FCGI_CHILDREN=5 \
PHP_FCGI_MAX_REQUESTS=1000 \
/usr/local/bin/php-fastcgi-5.3 -q -c /etc -b
                    127.0.0.1:9000 &

if [ -x /usr/local/bin/php-fastcgi-5.3 ]; then
     chroot -g nobody -u nobody / \
    env -i PHP_FCGI_CHILDREN=5 PHP_FCGI_MAX_REQUESTS=1000 \
     /usr/local/bin/php-fastcgi-5.3 -q -c /etc -b
                    127.0.0.1:9000 &
fi
```

Listing 2. *Restarting Your Web Stack*

```
/etc/rc.d/nginx restart
/etc/rc.d/snmpd restart
kill -9 `pgrep php`
chroot -g nobody -u nobody / \
env -i PHP_FCGI_CHILDREN=5 PHP_FCGI_MAX_REQUESTS=1000 \
/usr/local/bin/php-fastcgi-5.3 -q -c /etc -b 127.0.0.1:9000 &
```

127.0.0.1 Sweet 127.0.0.1!

```
listen on 127.0.0.1
read-only community public
system services 74
```

PHP 5.3 is strict about how time functions are used. It's going to throw up a bunch of errors unless you set your `date.timezone` variable in `/etc/php-5.3.ini`. Uncomment the following line, and set your time zone:

```
;date.timezone =
```

Find out more about time zones in PHP here: *http://www.php.net/manual/en/datetime.configuration.php.* At this point we've added new PHP modules, and we've changed several config files. Let's restart some daemons: Listing 2.

Cacti only ships with graphs for your local host. It's up to you to read the Cacti documentation, and add more graphs and data sources. You'll find that your RAM and CPU graphs aren't working. The more I looked into the problem, the more complicated things started to get.

I gave up when I ran across this post from OpenBSD developer Hans Insulander, in which he calls net-snmp *…the most horrible piece of code I've ever seen*. Later in the thread, Aaron Glenn says *SNMP uses ASN.1, so no amount of polising [sic] it going to make it sane*. It seems that the bottom line is that the OpenBSD developers aren't going to put much SNMP support into the OS because they don't like the only SNMP solution that's currently available.

Enter Symon. Like Cacti, Symon makes graphs; however, Symon's purpose is to graph the performance of localhost. To make this work, we need three Symon components: Symon, Symux, and Syweb. Installing the OpenBSD packages *almost* works out-of-the-box. First, install the packages:

```
# pkg_add -i symon syweb
```

The `/etc/symon.conf` file is fine. Leave it alone. Add the following line just before the last closing curly brace in `/etc/symux.conf`:

```
datadir "/var/nginx/html/symon/rrds/localhost"
```

After installing the symon and syweb packages, we need to move some folders to accommodate nginx.

```
# mv /var/nginx/symon /var/nginx
# mv /var/nginx/html/syweb /var/nginx/html
```

Then type in some commands:

```
# mkdir /var/nginx/symon/rrds/localhost
# chown -R nobody /var/nginx/symon
# /usr/local/share/symon/c_smrrds.sh all
```

Edit `/var/nginx/html/syweb/setup.inc`. Comment out the block for *OpenBSD, Apache chrooted*. Pay close attention to where `/*` and `*/` are. Those are the comment blocks.

```
/* running OpenBSD, apache chrooted:
$symon['rrdtool_path']='/bin/rrdtool';
$symon['cache_dir']='/symon/cache';
$symon['host_tree']='/symon/rrds';
$symon['layout_dir']='/symon'; */
```

Create a new block for nginx:

```
/* running OpenBSD, nginx not chrooted: */
$symon['rrdtool_path']='/usr/local/bin/rrdtool';
$symon['cache_dir']='/var/nginx/symon/cache';
$symon['host_tree']='/var/nginx/symon/rrds';
$symon['layout_dir']='/var/nginx/symon';
```

Start Symux and Symon:

```
# /usr/local/libexec/symux
# /usr/local/libexec/symon
```

Add them to your `/etc/rc.local`

```
if [ -x /usr/local/libexec/symux ]; then
        echo -n ' symux'; /usr/local/libexec/symux
fi
if [ -x /usr/local/libexec/symon ]; then
        echo -n ' symon'; /usr/local/libexec/symon
fi
```

TOBY RICHARDS

Toby Richards has been a network administrator since 1997. Each article comes straight from the notes that he takes when doing a new project with *BSD.

Among clouds Performance and Reliability is critical

Download syslog-ng HSRL (version 4 F1) product evaluation here

Attend to a free HSRL tech webinar here

BalaBit IT Security

www.balabit.com

syslog-ng log server

The world's first HSRL logging technology

HIGH-SPEED RELIABLE LOGGING

- above 500 000 messages per second
- zero message loss
- trusted log transfer and storage

Extracting Useful

Information From Log Messages

The syslog-ng application is a powerful and flexible system logging and log message processing tool to help the work of system administrators. This article highlights some of its newer and lesser-known capabilities.

What you will learn…

- The message model behind syslog messages
- The importance of metadata
- How to process, recognize, and correlate log messages using syslog-ng

What you should know…

- The basics of system logging

Applications usually send their log messages to the system logging daemon of the operating system, which delivers the messages to the place where the log messages are stored to log files on the local machine (found typically under `/var/log/`), or to a remote server. Most UNIX and Linux operating systems use the syslogd system logging daemon. The syslog daemon adds some meta-information (called the syslog header) to the received log messages, like the date and time the message was received, or the name or address of the host where it was created.

The syslog-ng project is a popular, alternative syslog daemon well-known for its reliable message transfer and flexible message filtering and sorting capabilities. Over the recent years, it gained several useful features to extend its capabilities, including the direct logging to SQL databases, TLS-encrypted message transport, and the ability modify the content of log messages. Owing to its ability to parse and identify messages based on a pattern database, and even to correlate log messages to identify events, it can also double as a real-time, high-speed log analyzing engine.

Standard syslog message Formats

Syslog messages usually come in one of two formats: the RFC3164 (sometimes also called BSD-syslog or legacy-syslog) message format, or the newer RFC5424 message format. The main characteristics of the two formats are summarized below.

RFC3164

An RFC3164 syslog message consists of the following parts:

- PRI
- HEADER
- MSG

The total message cannot be longer than 1024 bytes. The following is a sample syslog message: `<133>Feb 25 14:09:07 webserver syslogd: restart`. The message corresponds to the following format: `<priority>timestamp hostname application: message`.

The PRI part of the syslog message (known as Priority value) represents the Facility and Severity of the message. Facility represents the part of the system sending the message, while severity marks its importance. The Priority value is calculated by first multiplying the Facility number by 8 and then adding the numerical value of the Severity.

The HEADER part contains a timestamp and the hostname (without the domain name) or the IP address of the device. The timestamp field is the local time, without timezone information.

The MSG part contains the name of the program or process that generated the message, and the text of the message itself. The MSG part is usually in the following format: program[pid]: message text.

RFC5424

The main improvement of the RFC5424 message format is that the timestamp is in ISODATE format, meaning that it includes timezone information as well. The other important addition is that it introduces the Structured Data (SDATA) part of the message, which can contain arbitrary metadata about the message or the host. For example, when reading log messages from files, the syslog-ng syslog daemon includes information about the file, like path, filename, and filesize. Unfortunately, using the RFC5424 message format is not too widespread yet.

The Power of The syslog-ng message Modell

Internally, the syslog-ng application converts every message it receives to a set of name-value pairs called properties. Every part of the incoming message becomes a property, for example, the sender hostname, the sending application, the message itself, when the message was sent, and also when it was received. The SDATA fields of RFC5424-formatted messages are also automatically converted to properties.

Other metadata, for example, tags or custom fields can be added manually to the messages, but syslog-ng can also extract data from the messages using parsers (more on that later). The advantage of this approach is that it is not limited to storing syslog messages, it can be easily extended to other message-types as well, for example, SNMP messages or audit logs.

More precisely, internally syslog-ng stores the following data about every message:

- timestamps: The time syslog-ng received the message, and also the time parsed from the incoming message (if available)
- facility/priority: Values that represent the syslog facility (kern, user, daemon, etc) and the priority of the message (commonly called severity in RFCs). Available both as numeric and text values.
- tags: Custom labels added to the message. For example, syslog-ng automatically adds the ID of the source to every message received from that source.
- properties: Any other metadata added to or extracted from the message. Properties are available as name-value pairs. For example, HOST, PROGRAM, PID, MSG, matches from filters using regular expressions ($1,$2), user-defined properties of the message, and so on.
- SDATA: A subset of the properties, the data received in the SDATA part of RFC5424-formatted messages.

You can modify most properties freely if needed, for example using rewrite rules to set the value of a property, or to search for a specific string or regular expression and replace its value. However, some of the properties are read-only, most notably the ones that are related to the date when syslog-ng received the message.

Why Metadata is Useful

Metadata is useful for several reasons. It provides additional context about the log messages, making it easier to understand the events, and can also provide information about the environment of the event.

Message properties give you a flexible and powerful way to tweak your logging configuration to best suit your needs.

You can use templates that reference properties at several places of syslog-ng, including the name of the destination files, database tables, and so on. The properties used in a template are automatically resolved to their respective value that is relevant to the message. This means that if you include the HOST property in the name of the destination file on your central log server,

syslog-ng basics

The syslog-ng application can receive messages from different types of message sources, including files, sockets, or remote hosts. The syslog-ng application reads incoming messages and forwards them to the selected destinations. Destinations are the places where the log messages are stored; they can be files, other hosts, sockets, database tables, or external applications.

Sources and destinations are independent objects Log paths define what syslog-ng does with a message, connecting the sources to the destinations. A log path consists of one or more sources and one or more destinations; messages arriving from a source are sent to every destination listed in the log path.

Log paths can include filters. Filters are rules that select only certain messages, for example, selecting only messages sent by a specific application. If a log path includes filters, syslog-ng sends only the messages satisfying the filter rules to the destinations set in the log path.

Other optional elements that can appear in log statements are parsers and rewriting rules. Parsers segment messages into different fields to help processing the messages. Rewrite rules modify the messages by adding, replacing, or removing parts of the messages.

syslog-ng will sort the incoming messages of every host to separate files based on the hostname of the sender host.

Example:
Using properties in templates

```
destination d_file {
 file("/var/log/$YEAR.$MONTH.$DAY/$HOST.log");
};
```

To create a separate logfile for every application, just add the `$PROGRAM` property to the template:

```
destination d_file {
 file("/var/log/$YEAR.$MONTH.$DAY/$HOST/$PROGRAM.log");
};
```

Example:
How to transfer metadata to the logserver
To transfer the name of a logfile to the syslog-ng servr, set the .SDATA field of the messages your client. The name of the source log file is available in the `$FILE_NAME`. Property:

```
rewrite r_setfilename { set("$FILE_NAME" value(".SDATA.file@
                    18374.4.name"));
```

On the server side, create a destination that uses the above property in its filename template:

```
destination d_test {
 file( "/var/log/apache2/${.SDATA.file@18372.4.name}"
 create_dirs(yes)
);
 };
```

You can use templates to completely reformat or restructure a log message, for example, to correct log messages that do not comply with the syslog standards, change the timestamp format of the message, or to integrate better with external log processing scripts or log analyzing applications.

Example:
Customizing message format

```
destination d_file { file("/var/log/messages"
   template("$ISODATE $HOST $MSGHDR$MSG\n"); };
```

Another important method for using message properties are the filters, to select and sort log messages, to send only the selected messages to a destination. Comparing the values of properties, or combining filters with boolean operators allows you to select exactly the messages you are looking for.

Example:
Filtering messages
The following example selects messages with priority level 4 or higher.

```
filter f_level {"$LEVEL_NUM" > "5"};
```

The following filter matches on hostnames starting with the myhost string, for example, on myhost-1, myhost-2, and so on.

```
filter f_wildcard {host("myhost*" type(glob));};
```

The true power of metadata and properties can be harnessed when you use parsers to segment the MSG part of the messages into meaningful properties. For example, you can extract the username and hostname from login and logout messages, the command executed when using *sudo*, and so on. Extracting such information allows you for example to store all these metadata in a database, and create queries and reports to better understand and review what happens on your systems. In case something happens to one of your servers, you can query every login event to see who has accessed the system.

Example
Starting from version 3.3, the syslog-ng Open Source Edition application can store the log messages and its associated properties to a MongoDB database server. From the MongoDB database, you can generate reports for example using the JasperReports application.

Parsing Log Messages

Structured messages formatted as comma-separated values can be easily segmented into separate fields using syslog-ng's `csv-parser()` functionality. These fields become properties that can be used, for example, in templates or filters if you add custom names to them. The log messages from the Apache webserver application can be parsed that way, and filtered into different files based on the username to provide audit trails for the users' actions. Note that you can use parsers on any message field, not only on the textual part of the message. You can also segment hostnames structured like myhost-1 and myhost-2 if needed.

Example:
Parsing Apache Log Messages

The following parser processes the log of Apache web servers and separates them into different fields. Apache log messages are actually whitespace-separated, and usually look something like:

```
192.168.1.1 - - [31/Dec/2007:00:17:10 +0100] „GET /
cgi-bin/example.cgi HTTP/1.1” 200 2708 „-” „curl/7.15.5
(i4 86-pc-linux-gnu) libcurl/7.15.5 OpenSSL/0.9.8c zlib/
1.2.3 libidn/0.6.5” 2 example.balabit
```

A parser that processes these messages in syslog-ng looks like: see Listing 1. Another, more powerful option is to process the messages using a pattern database. As this is a bit more complex, it is detailed in the next section. The current development version of syslog-ng (version 3.4) will be able to parse JSON-formatted fields as well.

The syslog-ng Pattern Database

The syslog-ng application can also compare the contents of the received log messages to a set of predefined message patterns. That way, syslog-ng is able to identify the exact log message, assign metadata relevant to that particular log message (for example, the type of the event like security, hardware error), and also to extract data from the recognized message.

The syslog-ng pattern database uses Radix trees, because this method is fast and scales very well. The speed of processing a message is practically independent from the total number of patterns in the database, it is only related to the length of the message and the number of *similar* patterns.

In comparison with traditional message processing solutions that typically use regular expressions, syslog-ng message patterns are easy to write, understand, and maintain. Compare the following:

Listing 1. *Parsing Apache log messages with a csv-parser*

```
parser p_apache {
csv-parser(columns("APACHE.CLIENT_IP", "APACHE.IDENT_NAME", "APACHE.USER_NAME",
                   "APACHE.TIMESTAMP", "APACHE.REQUEST_URL", "APACHE.REQUEST_STATUS",
"APACHE.CONTENT_LENGTH", "APACHE.REFERER", "APACHE.USER_AGENT",
"APACHE.PROCESS_TIME", "APACHE.SERVER_NAME")
flags(escape-double-char,strip-whitespace)
delimiters(" ")
quote-pairs('""[]')
);
};
```

Listing 2. *A pattern database for a single log message*

```
<patterndb version='4' pub_date='2010-10-17'>
   <ruleset name='ssh' id='123456678'>
      <pattern>ssh</pattern>
      <rules>
         <rule provider='me' id='182437592347598' class='system'>
         <patterns>
            <pattern>Accepted @QSTRING:SSH.AUTH_METHOD: @
for@QSTRING:SSH_USERNAME: @from\ @QSTRING:SSH_CLIENT_ADDRESS: @port @NUMBER:SSH_PORT_NUMBER:@
ssh2</pattern>
      </patterns>
         </rule>
      </rules>
   </ruleset>
</patterndb>
```

A log message from an OpenSSH server:

```
Accepted password for joe from 10.50.0.247 port 42156 ssh2
```

A regular expression that describes this log message and its variants:

```
Accepted \ (gssapi(-with-mic|-keyex)?|rsa|dsa|password|
publickey|keyboard-interactive/pam) \ for [^[:space:]]+
from [^[:space:]]+ port [0-9]+( (ssh|ssh2))?
```

An equivalent pattern for the syslog-ng pattern database:

```
Accepted @QSTRING:auth_method: @ for @QSTRING:username:
@ from @IPv4:client_addr: @ port @NUMBER:port:@
```

As you can see, the syslog-ng pattern uses data types to specify the type of value that it expects in the message, for example, quoted-string (QSTRING), number, IP address. Also, the matching values are extracted from the message and can be used as properties. In the above example, such properties include the authentication method, the username, and the IP address of the client. These properties can be used in filters and templates. If you store the log messages and these properties in a database, it is really simple to query which users have accessed your server from which clients.

A syslog-ng pattern database is an XML file that stores patterns and various metadata about the patterns. The message patterns are sample messages that are used to identify the incoming messages. Metadata can include descriptions, custom tags, a message class – which is just a special type of tag – and any other properties added to the messages matching the pattern.

Listing 3. *Defining actions in the pattern database*

```
<actions>
   <action>
      <message>
         <values>
            <value name="MESSAGE">A log message from
                  $HOST matched rule number
$.classifier.rule_id</value>
         </values>
      </message>
   </action>
</actions>
```

Example:
A pattern database containing a single pattern

The following sample database contains a rule for the SSH login messages: `Accepted password for sampleuser from 10.50.0.247 port 42156 ssh2`.

The following is a simple pattern database containing a matching rule (Listing 2).

This metadata can be used for example to tag important messages, and then filter the messages based on this tags. You can also generate special alert messages if a particular log message is received, as described in the next section.

Triggering New Messages and External Actions

With the syslog-ng pattern database you can automatically generate new messages when a particular message is recognized. The generated messages can be configured within the pattern database rules, practically a new message can be generated for every incoming log message. Of course this is only rarely needed, unless log normalization is a must for you.

Example:
Generating a new message

When inserted in a pattern database rule, the following example generates a message when a message matching the rule is received (Listing 3).

Sending alerts directly from syslog-ng is currently not supported, but it is reasonably simple to pass the selected messages to an external script that sends out alerts in e-mail or SNMP. Also, the next version of syslog-ng (3.4) will include an SMTP destination to send emails directly from syslog-ng.

Messages can be triggered also when using syslog-ng to correlate messages, and the correlation timeout of a context expires. This is explained in the next section.

Correlating Log Messages

Message correlation is one of the foundations of log analysis and reporting, because log messages tend to be hectic, and often separate important information about events into different log messages.

For example, the Postfix e-mail server logs the sender and recipient addresses into separate log messages. For OpenSSH, if there is an unsuccessful login attempt, the server sends a log message about the authentication failure with the reason for the failure in the next message.

Listing 4. *Consolidating information into a single message*

```
<pattern>Accepted @QSTRING:SSH.AUTH_METHOD: @ for@QSTRING:SSH_USERNAME: @from @QSTRING:SSH_CLIENT_ADDRESS:
 @port@NUMBER:SSH_PORT_NUMBER:@ ssh2</pattern>
<pattern>pam_unix(sshd:session): session closed for user @ESTRING:SSH_USERNAME:</pattern>
<value name="MESSAGE">An SSH session for $SSH_USERNAME from ${SSH_CLIENT_ADDRESS}@1
```

But most of the time the actual event and its circumstances are important, and not the individual log messages. Being able to collect information as events rather than separate messages about an event can make the life of every system administrator a lot easier.

Message correlation in syslog-ng operates on the log messages successfully identified by the syslog-ng's pattern database. You can extend the rules describing message patterns with instructions on how to correlate the matching messages.

Correlating log messages involves collecting the messages into message groups called contexts. A context consists of a series of log messages that are related to each other in some way, for example, the log messages of an SSH session can belong to the same context. Messages may be added to a context as they are processed. The context of a log message can be specified using simple static strings or with macros and dynamic values. For example, you can group messages received from the same host (`$HOST`), application (`$HOST$PROGRAM`), or process (`$HOST$PROGRAM$PID`).

Messages belonging to the same context are correlated, and can be processed in a number of ways. It is possible to include the information contained in an earlier message of the context in messages that are added later. For example, if a mail server application sends separate log messages about every recipient of an e-mail (like Postfix), you can merge the recipient addresses to the previous log message. Another option is to generate a completely new log message that contains all the important information that was stored previously in the context, for example, the login and logout (or timeout) times of an authenticated session (like SSH or telnet).

To ensure that a context handles only log messages of related events, a timeout value can be assigned to a context, which determines how long the context accepts related messages. If the timeout expires, the context is closed. Also, the context can be explicitly closed when the last message of a context is received If a context collects the messages of a connection, the log message about the server closing the connection can trigger the closing of the context.

When message correlation is used together with triggering actions, you can also refer to fields and values of earlier messages of the context. The following patterns would put out a correlated message that included information from both log messages (Listing 4).

To process already collected log messages, syslog-ng also allows for correlating log messages from log files. For this reason, the time elapsed between two log messages is calculated from the actual timestamps of the log messages instead of using the system time.

Where to Find syslog-ng

The syslog-ng Open Source Edition application is available in the FreeBSD Ports and Packages Collection. Earlier the port was named sysutils/syslog-ng3, which was renamed recently to syslog-ng with the removal of the syslog-ng 1.X port. To reduce the number of external dependencies, packages only have a limited feature set. To be able to use every feature described in this article, install syslog-ng version 3.3 from up-to-date ports. The source of syslog-ng is also available at the project's homepage at *http://www.balabit.com/network-security/syslog-ng/*. If you experience problems or have a comment, join the syslog-ng mailing list at *https://lists.balabit.hu/mailman/listinfo/syslog-ng/*.

ROBERT FEKETE

Robert Fekete has been the documentation maintainer of BalaBit, a Hungarian software company developing syslog-ng and other network security products. He is also an open-source enthusiast and enjoys writing articles about free software – time permitting.

Anatomy

of a FreeBSD Compromise (Part 1)

While the BSD family is more secure than most, no server or IT system is invulnerable to attack. In this article we will examine best practices to prevent disruption and what to do when the worst does happen.

What you will learn…

- How to plan a security strategy

What you should know…

- BSD administration skills

I have a love-hate relationship with the Internet. Being a bit of an IT dinosaur who comes from the days of 1200 Baud modems, leased lines and CP/M, IT security in the 1980's was a much simpler affair and so much easier to manage. In those days, if your company was fortunate enough to have a corporate network it was an expensive affair, rigorously policed and the opportunities for mischief was mainly down to the *white hat* hacker, who did not want to compromise the system for financial gain, commercial secrets or to cause embarrassment. Rather, the mindset was to find out more – in other words there was very little malevolence and when it did happen, an attack was relatively easy to detect and contain as often it was being made by an individual. Skip forward to 2011, and we have bot-nets, port scanners, malware and viruses, social engineering and technology has permeated further into our offices, homes and cars. Attacks are now on a global scale – country versus country – and anyone with an Internet connection can quickly download a guide on how to perform a compromise with surprising ease. While governments are very slow to understand and legislate effectively for these issues, as a system administrator you are the first line of defense – unfortunately prosecutions are generally rare unless the victim or the hacker is particularly high profile.

The Internet aside, computer security has always been a fascination for me and it never ceases to amaze me how organizations manage to survive without major incident. Open plan offices (particularly in IT), passwords not encrypted or data stored in clear text databases are just some of the risks, but often the foolishness of colleagues is very revealing. Sending an unencrypted database of customer details via email is one example, but I regularly have observed technical support staff (or even badly designed systems) send out emails with login details. Apart from the risk of an man-in-the-middle attack, what happens when Joe saves the email in the email archive or on his desktop? Anyone with access can quickly determine how to compromise another system. So the problem is not just one of securing the technology, but also of educating the users of best practice.

How Secure Are You?

I have always said that the only way to 100% secure a server is to encase it in concrete, and drop it at the bottom of the Atlantic ocean. This might sound extreme, but it has even been suggested that OpenBSD has been compromised (See the Register article in Table 3). There are too many attack vectors, bugs and creative ways of getting todays complex applications to perform tasks they were never really designed for. I often use telnet to test web-servers or send test emails for example, and with modern diagnostics that were not available 30 years ago, the black-hats have the advantage.

Take the example of the zero-day exploit. Black hat Fred discovers an undocumented vulnerability in software X. He tells his friends, but warns them not to tell anyone or publicize it. If the system administrators of these systems do not detect any strange behavior in their systems, and there is no process available to recognize the attack, Fred and his friends will remain undetected. It is only when the attacks become more widespread, damaging, a security researcher finds out or the attack is publicized that the developer of software X can then patch the vulnerability. It is all based on scale and risk – if the attacks are widespread someone will eventually pick up on the anomaly, and if the damage is severe, questions will quickly be asked. But what happens if Fred keeps the vulnerability to himself and only uses it once or twice? The chance of detection is small provided the payload (i.e. damage to the system) is trivial, and the developer and security community is not aware of the exploit. Until Fred's technique is released into the wild, the vulnerability will remain exploitable until it reaches a tipping point where the problem diagnosed, analyzed and fixed – provided Fred is not greedy or stupid enough to attack a honey-trap.

This is the real problem – the *unknown unknowns*. Good systems management can reduce the risk, but it will never eliminate it entirely. Like the banking system, our IT infrastructure is based very much on trust. Sure we can white-list our web-servers so only our esteemed customers can access our pages, but even this would not be a practical solution, as it is relatively trivial to spoof an IP and MAC address. Private networks have their uses, but in the Internet age everybody wants to get connected. WWW does not stand for World Wide Web, in reality it means Wild Wild West.

Table 1. *The Sys-admin 10 Commandments*

The 10 commandments of the security aware administrator
Paranoia is good – Evidence based paranoia is even better. Log everything including checksums.
Your adversary might be closer than you think – keep confidential information confidential.
Encrypt, password protect and only let others have access if they really need it. Use secure passwords and change them regularly. Don't embed passwords in application scripts etc.
Patch and upgrade often.
Only run services if you really really have to. Even then, harden if you can.
Regular backups are not optional – they are essential.
Try and hack your own systems. You'll be surprised at what you will find. Don't be complacent.
Others will not be as conscientious as you – ensure everyone else is security aware.
If you are attacked, don't panic. Be methodical and retain the evidence. If in doubt or it is a major incident, bring in security professionals before you destroy any trail. Don't be tempted to exact technological revenge – you will compromise any official investigation and possibly leave yourself open to prosecution.
You are not alone – there are more people who are with the good guys than with the bad.

Know Your Enemy

In the The Art of War Sun Tzu said *Know your enemy* and indeed the sys-admin needs to develop a war-like attitude if he has any valuable data or public facing servers. First of all, the most obvious area of vulnerability are your users. A strong Acceptable Use Policy that is enforced rigorously by senior management is a good starting point, the biggest issue is finding the right balance. Too restrictive, and IT will be considered overbearing and it will be unenforceable, too loose and the policy will not have sufficient teeth to deal with those that do not take security seriously. It is worth considering adding good practice in here about confidentiality and data security, as many countries now have legislation concerning data protection and e-commerce transactions. While there is

Table 2. *The Hackers 10 Commandments*

The black hat hacker – 10 commandments
Cover your tracks. Do not expose your originating IP address ever. Clean and flush any evidence that you have been visiting.
Get root. Once you have this, the world is your oyster and you can hide yourself even better.
Do not show your hand. Maxing out a servers' resources just announces your presence. You want your victim to be oblivious to your presence so don't be greedy.
No server is 100% secure, but some are less secure than others. Go for the easy targets first – the more common the exploit the less chance of being exposed.
Trust no-one. Bragging rights are fine until someone wants to get the drop on you.
If you hack you will upset people. Some of these people will be powerful enough to punish you – so consider the risks.
Know your code. If you don't understand C, assembler and pointers, you are just an amateur.
Get social skills. One the most powerful methods of compromise is social engineering, and this requires gaining the confidence of others.
Keep abreast of technology. New vulnerabilities appear every day as the technological field expands. There are few security experts and many bad programmers.
Physical access to machines always gives you an edge (e.g. key-loggers) – but the risks are greater

commercial software available for data loss prevention, but those who understand Unix well will appreciate the power of scripting here. As part of the AUP, any confidential information should be marked as such, and it would be trivial to configure an email server to re-route attachments to a queue where they would be monitored and released if OK. Taking things further, emails could be scanned for certain key words and queued / released as appropriate. The only problem here is those that want to get around the system, and one good example of how this can be prevented is to refuse attachments at the server, and use a secure gateway where files are uploaded and collected by the customer. The password and login details should be telephoned or faxed through to the other end and security then becomes the other guy's problem. The majority of in-house compromises are down to mistakes (Be honest – when was the last time you accidentally sent someone a sensitive email in error?) and good practice will minimize the damage in most cases. Confidential documents on public servers are a common one, but by putting the correct procedures in place (e.g. only the web-master can upload documents) many risks can be mitigated. A good firewall is essential, as well as multiple proxies to deal with HTTP and FTP traffic. Email scanning for vulnerabilities (including attachments and links) works well when centralized.

Next, we have the deliberate compromise by staff. I know of documents and applications deliberately deleted or stolen, cables damaged or re-routed and hardware damaged. A colleague of my wife deliberately sabotaged a system on being denied a promotion, so this is where management and IT have a good platform to build relationships – when there is a risk to the organization. Is Dave in accounts about to be demoted / fired? Ensure his account and system rights are locked prior to that meeting with HR.

Finally, there is just carelessness. Open plan offices and wide screen TFT monitors are a snoopers delight, as are network diagrams with IP addresses and passwords left on desks or on monitors. If everyone in your organization is trustworthy this is fine, but the larger the organization the more difficult it becomes to trust everyone. I was once almost physically abused by a manager when I refused to give him the administrator passwords for multiple servers, this despite offering him unlimited access to what he wanted under an account that was tied to him. Alarm bells were ringing, and sure enough he was eventually discovered downloading pirated software from some rather strange locations. Only allow people access to systems on a *need to know* basis, and if in doubt say no. If need be, clear it with a more senior manager, but the default should be caution.

The next category covers *the outside world*. There are different types of hacker with different agenda's, as demonstrated in Table 4. I have split hackers into different categories, as different type of attack carry different payloads, levels of compromise etc. In reality, there are only the *good guy's* and the *bad guys*, but what I am trying to illustrate here is that while all attacks are objectionable in one degree or another, there difference is level of seriousness to the attack. A script kiddie will be relatively trivial to deal with (unless of course there has been significant incident e.g. passwords compromised or data lost) whereas a full-blown attack against your organization by a foreign competitor or political adversary wanting to extract secrets will be a different matter entirely. One may require a server rebuild, the other the involvement of law enforcement and a full audit. Sadly, for the individual or small business there is little redress from the police or government when an attack takes place – I can just picture the scene if I walked into a police station to report an outbreak of the Melissa virus. If you are a large corporate, politically connected or a government agency you may be in the fortunate position to be listened to if the attack is serious (See EDF Greenpeace incident Table 1). Most organizations have the attitude that a hacking

Table 3. *References*

Reference	Site
FBI 'planted back-door' in OpenBSD	*http://www.theregister.co.uk/2010/12/15/openbsd_backdoor_claim*
EDF fined for hacking into Greenpeace	*http://www.ft.com/cms/s/0/78f3b452-0c70-11e1-8ac6-*
Post Compromise Shell Shoveling	*http://www.madirish.net/node/237*
MUH	*http://muh.sourceforge.net/*
Process faker	*http://packetstormsecurity.org/search/files/?q=process%20faker*
Metasploit framework	*http://metasploit.com*
Wireshark	*http://www.wireshark.org*
Nmap	*http://nmap.org*
Backtrack Linux	*http://www.backtrack-linux.org*

attempt is *Not their problem*. I recently received a phishing email posing as major UK bank, and being inquisitive, I checked out to see if the phishing site was still active. To my surprise it was, so as a good netizin I contacted the customer services at the bank to be informed that they did not have a security officer available to deal with incidents at the weekends. ISP's and Internet businesses are also at fault – I have lost count of the number that I have contacted regarding spam or fraudulent email claiming to originate from a blue-chip business to be met with a complacent *So what?* – so to the message is not getting through about security especially the damage this causes a companies reputation.

Adopting a Security Minded Approach – The Security Aware Administrator

I have listed the 10 commandments for both administrators and hackers (Table 1 & 2) as it is important to think the way the other side thinks. The biggest enemy of all is complacency, as I recently discovered with my very elderly FreeBSD 6.1 box was hacked. While Internet facing, I had taken great pains to ensure that the system was adequately fire-walled, penetration tested and secure when it was commissioned. As the server is really old and only used as a proxy and mail gateway, I made the decision not to manually upgrade or patch it on a regular basis as it would be quicker to rebuild with FreeBSD 8 when I eventually got round to upgrading all of the hardware etc. Part of me knew I was asking for trouble, especially as I was running a legacy version of Apache 1.3, but as there was no websites other than a holding page I thought I was safe. I also succumbed to the mantra *if it ain't broke don't fix it* as it was essential to keep the box running. I knew my E-mail was secure behind the firewall (store and forward to ISP and tied down by IP address) and there was only one non-root login account to SSH on a irregular port with a strong password. The server was regularly scanned for root-kits etc, and the IP address provided by my ISP was dynamic. While I observed irregular traffic accessing non-existent pages on the website, I did not investigate this further mainly due to time and there was no obvious pattern as to their origin. With hindsight, I should have put together some scripting that automatically fire-walled IP address on multiple requests to non-existent pages.

A few weeks ago I started experiencing problems with my broadband connection, which very shortly afterwards resulted with my broadband router becoming totally dead. As this is supplied by my ISP, I could not patch this any more than the manufacturer had, so I did make some limited modifications to improve security e.g. disabling wireless networking etc. I replaced the router with a spare and

Table 4. *Reasons for compromise*

Type of hacker	Modus Operandi
"White hat" ethical hacker	Breaks into systems just for fun / learning. No damage done. Sys-admin and software writer privately informed of exploit, may make exploit public if the attacked parties do not take threat seriously.
Script Kiddie	Pulls exploits off the web and runs against public or private servers. Does it for the kicks, not aware of the risks to themselves (e.g. exposed IP address) or attacked systems. Often takes the form of Denial of Service attacks (e.g. overloading a server with requests).
Common hackers and spammers	Potentially linked to organized crime, wants to reach as many machines as possible as they are often quickly discovered. Uses many attack vectors, from malware to phishing and website compromise. Interested in everything from harvesting email addresses to credit card numbers and denial of service attacks. Majority of attacks can be mitigated by good overall security apart from zero-day exploits.
Bandwidth theft	Use systems as devices for man-in-the-middle attacks, or to cloak illegal activity. On the rise with Wireless networking / Bluetooth. Network equivalent of phone-phreaking.
"Black hat" hacker	Wants access to specific commercial or personal secrets, passwords, to deliberately damage and corrupt systems / data and targets a specific victim for a reason. Professional level of exploit, would use any and all possible attack vectors to gain advantage (e.g. malware/social engineering/key logging/surveillance/network sniffing/phone tap etc.) Not to be confused with the common hacker who plays the numbers game – this is a targeted specific attack e.g. Stuxnet worm.

while testing performed some diagnostics with NETSTAT and discovered traffic to an IRC server in Hungary. As my wife and daughter both have Internet access, I wasn't immediately concerned, so I checked the internal LAN but this was not the origin. Isolating the server, I checked both PS and TOP for irregular processes, but nothing was there. TCPDUMP was my next stop, and traffic was going out once a minute or so but nothing was coming back, so by this time I was beginning to realize not all was well with my server. The final clue was running LSOF – there were files open in my `/tmp` directory that were not recognized, that had been written to by www-data. This was what gave the game away – I didn't have any web-pages that would write anywhere on the server, and further investigation showed a MUH infection which was cloaked by Process Faker. The hacker was running a service to an IRC server off the back of HTTPD, and was running a crontab under the www-data user account to restart the process should the server be rebooted. As the files were flagged as owner www-data, it was clear that Apache was the week link in the chain, and access was obtained over port 80, possibly using a post command and shell shoveling. All in all, quite a sophisticated hack. Fortunately, looking at the logs the hacker had not managed to successfully connect to the IRC server, so I was happy that no data loss had taken place. Removing the crontab and the files from the /tmp directory cured the problem, so I disabled Apache for the time being and the server is now functioning as expected.

In the end analysis – in breaking the 4th commandment (Table 1) it comes down to this – As Catherine Aird said: *If you can't be a good example, then you'll just have to serve as a horrible warning*. I hope my experience will useful to all.

In The Forthcoming Articles

We will look further at effective strategies for improving security as well as security tools including packet sniffers and port scanners, honey-traps and some *quick and dirty* firewall rules that will help mitigate an attack on Apache. We will also look at some vulnerability checkers, penetration testing and attempt to recreate the MUH attack again on a test server. I have provided the links in table 3 to some of the more well known tools.

Please Note

The information in these articles is designed for system administrators to help improve systems security. While testing and discovering vulnerabilities in your own personal systems for the purpose of improving security is perfectly legal in the UK, legislation is different in other parts of the world. Employers may take disciplinary and / or legal action against employees if these tools are used in the workplace without permission. Using these tools against third-party systems without permission is not only considered unethical, but is also illegal in many countries. The author and BSD Magazine do not condone unethical or illegal hacking.

ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.

Hardening

BSD With Security Levels

By default, BSD servers are more secure then other operating system installations but still require some changes in order to be production ready. Security levels are one of the tools that can be used in order to maintain the state of the system when being deployed in production.

What you will learn…

- How to configure the security level on the BSD operating systems.
- How to use chflags to lock down system configurations and log files.

What you should know…

- Basic BSD knowledge to navigate the command line .
- Basic knowledge of sysctl.

This article covers the configuration of security levels via securelevel for OpenBSD, FreeBSD, NetBSD and DragonFlyBSD.

The function of the security level is to reduce the capabilities allowed by the kernel depending on the configured level. By default, all of the BSD kernels have this functionality including the ability to make changes upon system startup with settings in the `rc.conf file`. The security level can be increased while the operating system is running, but only lowered when the system goes into single user mode or is rebooted. FreeBSD apparently has the ability with `option REGRESSION` compiled into the the kernel to use a sysctl value to lower the security level.

The settings for securelevel are primarily the same for each of the BSD operating systems. Listing 1 details the starting security levels for each BSD operating system. The versions used are FreeBSD 8.2, OpenBSD 5.0, NetBSD 5.1, DragonFlyBSD 2.10.

The most notable difference is that OpenBSD by default is set to security level 1. The other three BSD operating systems default to permanently insecure mode. OpenBSD and NetBSD have similar settings just as FreeBSD and DragonFlyBSD share the same settings. Starting with security level -1, all `devices/files/memory` can be written to and manipulated. As the man pages mention, this default setting should be changed on FreeBSD, NetBSD, and DragonFlyBSD. Setting file attributes with `chflags` can be overridden as detailed in Listing 2.

The security level can be increased by updating the `sysctl` value for `kern.securelevel` as shown in Listing 3.

For FreeBSD, NetBSD and DragonFlyBSD at securelevel 1, the `chflags` command output will be similar to the default level in OpenBSD. This allows for configuration files to be locked down to maintain the integrity of the system when using the system immutable flag. In addition to configuration files, the integrity of logs files can be maintained as they continue to grow.

Listing 1. *The following levels are for each operating system*

```
FreeBSD 8.2
kern.securelevel: -1

OpenBSD 5.0
kern.securelevel=1

NetBSD 5.1
kern.securelevel = -1

DragonFlyBSD 2.10
kern.securelevel: -1
```

Listing *2. The following are commands run on the different BSD operating systems for sappnd and schg (system append-only and system immutable) flags. Note the difference between OpenBSD and the others as OpenBSD does not allow the changing of the sappnd or schg flags at the default security level of 1*

```
FreeBSD, NetBSD and DragonFlyBSD setting schg and
                    sappnd with chflags
freebsd# ls
test.conf
freebsd# chflags schg test.conf
freebsd# ls -lo
total 2
-rw-r-----  1 root  wheel  schg 22 Nov 16 12:00
                    test.conf
freebd# rm -rf test.conf
rm: test.conf: Operation not permitted
freebsd# chflags noschg test.conf
freebsd# rm -rf test.conf
freebsd# ls
freebsd#
freebsd# echo "LogEntry: Test" >> test.log
freebsd# wc -l test.log
       1 test.log
freebsd# chflags sappnd test.log
freebsd# ls -lo
 total 2
-rw-r--r--  1 root  wheel  sappnd 15 Nov 16 12:24
                    test.log
freebsd# echo "LogEntry: Test" >> test.log
freebsd# wc -l test.log
       2 test.log
freebsd# rm -rf test.log
rm: test.log: Operation not permitted
freebsd# chflags nosappnd test.log
freebsd# rm -rf test.log
freebsd# ls
freebsd#

OpenBSD setting schg and sappnd with chflags

# ls
test.conf
# chflags schg test.conf
# ls -lo
total 4
-rw-r--r--  1 root  wheel  schg 11 Nov 16 11:23
                    test.conf
# rm -rf test.conf
rm: test.conf: Operation not permitted
# chflags noschg test.conf
chflags: test.conf: Operation not permitted
# echo "LogEntry: Test" >> test.log
# wc -l test.log
       1 test.log
# chflags sappnd test.log
# echo "LogEntry: Test" >> test.log
# wc -l test.log
       2 test.log
# rm -rf test.log
rm: test.log: Operation not permitted
# chflags nosappnd test.log
chflags: test.log: Operation not permitted
```

References

- FreeBSD man page for securelevel: *http://www.freebsd.org/cgi/man.cgi?query=security&sektion=7&apropos=0&manpath=FreeBSD+8.2-RELEASE*
- FreeBSD Handbook warning on securelevels: *http://www.freebsd.org/doc/en/books/faq/security.html#SECURELEVEL*
- OpenBSD man page for securelevel: *http://www.openbsd.org/cgi-bin/man.cgi?query=securelevel&sektion=7*
- NetBSD man page for securelevel: *http://netbsd.gw.com/cgi-bin/man-cgi?secmodel_securelevel+9+NetBSD-current*
- DragonFlyBSD man page for securelevel: *http://leaf.dragonflybsd.org/cgi/web-man?command=securelevel&section=ANY*

FreeBSD and DragonFlyBSD are also restricted from loading and unloading kernel modules. This affects the functionality of kldload and kldunload. Another important detail is how FreeBSD jails implement the securelevel. The highest value is accepted by the jail even if the host value is set lower. This allows for the host to manipulate the necessary files with the jails being restricted as necessary.

Each BSD operating system has a max security level that can be configured with a subtle difference. FreeBSD and DragonFlyBSD share the same limit of 3, which is the same as security level 2 in OpenBSD and NetBSD. This distinction is due to the network settings that OpenBSD and NetBSD include in level 2 that are not used in FreeBSD and DragonFlyBSD unless you set the level to 3. Securelevel 2 prevents the changing of the system time backwards or moving it more then 1 second. This setting when used in conjunction with the system append-only flag allows for the log files to not only be be correct but accurate to the time in which the entries were made. For security level 3 (security level 2 for OpenBSD and NetBSD), firewall rules can not be altered. The system run level would have to be changed in order to modify the operating system. Example output for OpenBSD is detailed in Listing 4.

Though these features do help to lock down the BSD operating system, they must be tested and understood before sending out a production system. The security section of the FreeBSD handbook mentions that the system files prior to booting need to be protected in order for the securelevels to be effective. If an attacker is able to run code before the securelevel is set, the protections can be evaded. Beyond this, maintenance of the files may be difficult as the system would either require single-user mode or to be taken off-line in order to be updated.

Just like any other security tool, setting a security level is not a solution in and of itself. The securelevel setting provides a way to maintain the integrity of the running operating system. There are different file system features which are more flexible such as access control lists and mandatory access controls. Utilizing all of the security tools in the BSD operating systems help to keep the servers secured and running the way in which they were originally deployed.

Listing *3. The following are the sysctl commands to increase the kernel security level. OpenBSD being already at level 1 is increased to level 2*

```
FreeBSD 8.2
freebsd# sysctl kern.securelevel=1
kern.securelevel: -1 -> 1

OpenBSD 5.0
# sysctl kern.securelevel=2
kern.securelevel: 1 -> 2

NetBSD 5.1
# sysctl -w kern.securelevel=1
kern.securelevel: -1 -> 1

DragonFlyBSD 2.10
%sysctl kern.securelevel=1
kern.securelevel: -1 -> 1
```

Listing *4. The following is the output generated on the command line for OpenBSD with securelevel set to 2 and trying to flush the firewall rules for pf*

```
# pfctl -F rules
pfctl: pfctl_clear_rules: Operation not permitted
```

MICHAEL SHIRK

Michael Shirk is a BSD zealot who has worked with OpenBSD and FreeBSD for over 6 years. He works in the security community and supports Open-Source security products that run on BSD operating systems.

FreeBSD Foundation Update

The FreeBSD Foundation is a 501(c)(3) non-profit organization dedicated to supporting and building the FreeBSD Project and community worldwide. It represents the FreeBSD Project in executing contracts, license agreements, copyrights, trademarks, and other legal arrangements which require a recognized legal entity.

It also funds and manages development projects, sponsors FreeBSD events and Developer Summits, and provides travel grants to FreeBSD developers who would otherwise be unable to attend Developer Summits.

This article summarizes the conferences and projects that the Foundation funded in 2011. It concludes with how you can assist the Foundation in its efforts.

Conferences ,Travel Sponsorship, and Conference Booths

In 2011, the Foundation provided sponsorship for the following BSD conferences:

- AsiaBSDCon, held in Tokyo, Japan from March 17-20
- BSDCan and Developer Summit, held in Ottawa, Canada from May 11-14
- KyivBSD, held in Kiev, Ukraine on September 24
- EuroBSDCon and Developer Summit, held in Maarssen, Netherlands from October 6-9
- FreeBSD Vendor Summit, held in Santa Clara, CA from November 3-4

In addition to sponsoring these conferences, the Foundation paid for developers to attend the following conferences:

- FOSDEM: 1 developer
- BSDCan: 6 developers
- EuroBSDCon: 6 developers
- GSoC Mentor Summit: 1 developer

Each sponsored developer provides a trip report that indicates the value that was gained from their travel

sponsorship. Trip reports are available online at the Foundation's blog: *http://freebsdfoundation.blogspot.com/.*

Directors from the FreeBSD Foundation also volunteer at FreeBSD booths at conferences that provide exhibition areas. Visiting a FreeBSD booth provides an excellent opportunity to discuss and suggest funded development work as well as to make a donation to the Foundation. At least one Director was present at the following conference booths:

- FOSDEM, held in Brussels, Belgium on February 5-6
- SCALE, held in Los Angeles CA on February 26
- AsiaBSDCon, held in Tokyo, Japan from March 17-20
- Indiana LinuxFest, held in Indianapolis, IN on March 26
- FlourishConf, held in Chicago, IL on April 2
- BSDCan, held in Ottawa, Canada on May 13-14
- SouthEast LinuxFest, held in Spartanburg, SC on June 11
- Ohio LinuxFest, held in Columbus, OH on September 10
- EuroBSDCon, held in Maarssen, Netherlands on October 8-9
- LISA, to be held in Boston, MA on December 7-8 (upcoming)

For 2012, the following conferences have confirmed a FreeBSD booth with Foundation representation:

- SCALE, to be held in Los Angeles, CA on January 21
- NorthEast LinuxFest, to be held in Worcester, MA on March 17
- Indiana LinuxFest, to be held in Indianapolis, IN on April 14
- BSDCan, to be held in Ottawa, ON on May 11-12

It is expected that FreeBSD booths will be arranged at the other conferences that were attended in 2011, once the conference dates and locations have been confirmed.

Funded Development Projects

In 2011, the Foundation budgeted $125,000 to fund development work. $85,000 worth of work has been completed and two additional projects are being considered for the remainder of the 2011 budget. The following development projects have met their completion milestones:

IPv6 support in FreeBSD and PC-BSD

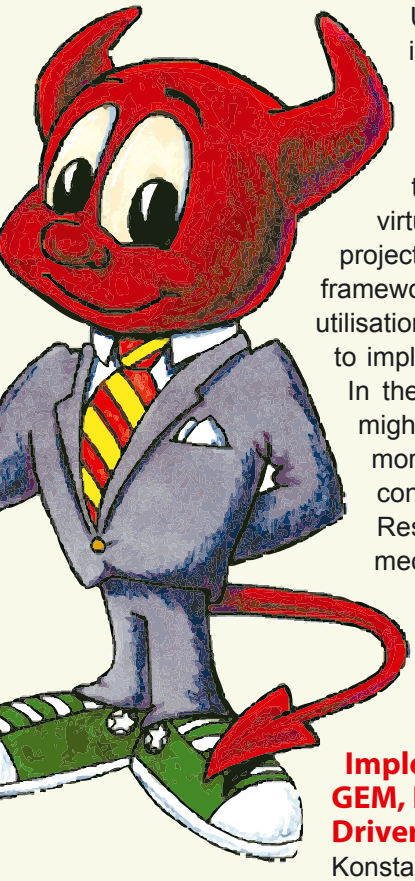
Bjoern Zeeb, recipient of the Itojun Service Award for his work on open source implementations of IPv6, was awarded a grant to improve the maturity of IPv6 support in FreeBSD and PC-BSD. This project was jointly sponsored with iXsystems, the corporate sponsor of the PC-BSD project.

FreeBSD's original KAME-based reference implementation of IPv6 first appeared in FreeBSD 4.0 and is found in a broad range of FreeBSD-derived commercial products. Before this project, IPv6 was an optionally configured feature present in the default FreeBSD kernel; however, that configuration also implied configuration of IPv4. With much „IPv6-ready" application software relying on dual-stack behavior, broken IPv6 applications go unnoticed. This project added support for an IPv6 kernel without IPv4 which makes FreeBSD and PC-BSD the ideal test and development platform for both open source and proprietary IPv6-aware application software.

This project was completed in time for both the FreeBSD and PC-BSD projects to participate in World IPv6 Day, held on June 8. IPv6-only versions of FreeBSD (*http://www.freebsd.org/ipv6/*) and PC-BSD (*http://pcbsd.org/IPv6*) are available.

Resource Containers Project

Edward Napierala was awarded a grant to implement resource containers and a simple per-jail resource limits mechanism.



Unlike Solaris zones, the implementation of FreeBSD Jails did not provide per-jail resource limits. As a result, users were often forced to replace jails with other virtualization mechanisms. This project created a single, unified framework for controlling resource utilisation, and used that framework to implement per-jail resource limits. In the future, the same framework might be used to implement more sophisticated resource controls, such as Hierarchical Resource Limits, or to implement mechanisms similar to AIX WLM. It could also be used to provide precise resource usage accounting for administrative or billing purposes.

Implementing support of GEM, KMS, and DRI for Intel Drivers

Konstantin Belousov was awarded a grant to implement support of GEM, KMS, and DRI for Intel video drivers. This project was also co-sponsored by iXsystems.

The project implemented GEM, ported KMS, and wrote new DRI drivers for Intel Graphics, including the latest Sandy Bridge generation of integrated graphic units. Once the work is fully tested, it will be committed and should allow the latest Intel open-source drivers with integrated, 3D-accelerated graphical capabilities to run on FreeBSD, expanding the range of hardware where FreeBSD is suitable for the desktop. PC-BSD testing snapshots that use the committed code are expected to be available before the end of the first quarter of 2012.

Feed-Forward Clock Synchronization Algorithms Project

Julien Ridoux and Darryl Veitch at the University of Melbourne were awarded a grant to implement support of feed-forward clock synchronization algorithms.

For many years, NTP has been the reference solution to synchronize computer clocks inexpensively. However, the ntpd daemon has begun to show limitations which are mainly due to the feed-back nature of its interaction with the kernel.

In contrast, a feed-forward approach is inherently robust and allows near-optimal performance to be reached. This project extended the FreeBSD kernel timing system to support feed-forward synchronisation daemons. This new synchronisation system allows both feed-back and feed-forward approaches to run on one system and give users the possibility to select the one more suited to their needs.

This feed-forward approach provides various new features such as faster timestamping, a new difference clock to measure time intervals with GPS-like accuracy and extremely high robustness, the ability to replay the clock offline based on stored raw timestamps, and accurate timing for virtual machines and live VM migration.

Implementing xlocale APIs

David Chisnall received a grant to implement xlocale APIs to enable the porting of libc++.

FreeBSD has always had its own C standard library implementation but uses the GPL-licensed GNU libstdc++ as the C++ standard library. libc++ is an alternative library that was developed as part of the LLVM project and which is available under the UIUC and MIT licenses. This library depends on a low-level C++ ABI library. An implementation of this ABI was written for PathScale and the FreeBSD and NetBSD Foundations jointly paid the costs for it to be open sourced.

The other dependency is the C standard library. libc++ was written by Apple and uses a set of non-portable extensions for localisation known as xlocale. This project implemented the missing xlocale APIs into FreeBSD's standard C library. Now that the project is complete, it is possible to build libc++ on FreeBSD.

DIFFUSE

The Swinburne University of Technology's Centre for Advanced Internet Architectures was awarded a grant to implement DIFFUSE for FreeBSD.

DIFFUSE (*Distributed Firewall and Flow-shaper Using Statistical Evidence*) is an extension to the FreeBSD IPFW firewall subsystem which allows IPFW to classify traffic based on statistical properties of realtime flows and to instantiate network actions across a distributed set of action nodes for particular flows. DIFFUSE uses machine learning techniques to enable robust and efficient classification of IP traffic flows based on their unique statistical properties in addition to traditional inspection of packet header or payload contents. DIFFUSE also allows traffic classification to occur in one place (e.g. in the core of a network) and trigger traffic

shaping and differentiation elsewhere (e.g. at the edges of a network).

This project integrated the DIFFUSE prototype into FreeBSD which will increase FreeBSD's utility to designers and implementers of FreeBSD-based networking infrastructure. DIFFUSE has applications in ISP, residential broadband, and large corporate network scenarios.

Five New TCP Algorithms Project

Grenville Armitage was awarded a grant to implement five new TCP congestion control algorithms.

Previous to this development work, FreeBSD's TCP stack did not have an easy-to-use mechanism for introducing new congestion control algorithms. This project delivered the following enhancements to FreeBSD's TCP stack:

- Modular congestion control framework.
- Khelp (Kernel Helper) and Hhook (Helper Hook) frameworks.
- Basic Khelp/Hhook integration with the TCP stack.
- ERTT (Enhanced Round Trip Time) Khelp module for delay-based TCP algorithms.
- Modularised implementations of NewReno, CUBIC and HTCP loss-based TCP algorithms.
- Modularised implementations of Vegas, „HD" and „CHD" delay-based TCP algorithms.
- Technical report comparing the computational overhead associated with TCP before and after integrating the new frameworks and modularised NewReno algorithm (*http://caia.swin.edu.au/reports/110228A/CAIA-TR-110228A.pdf*).

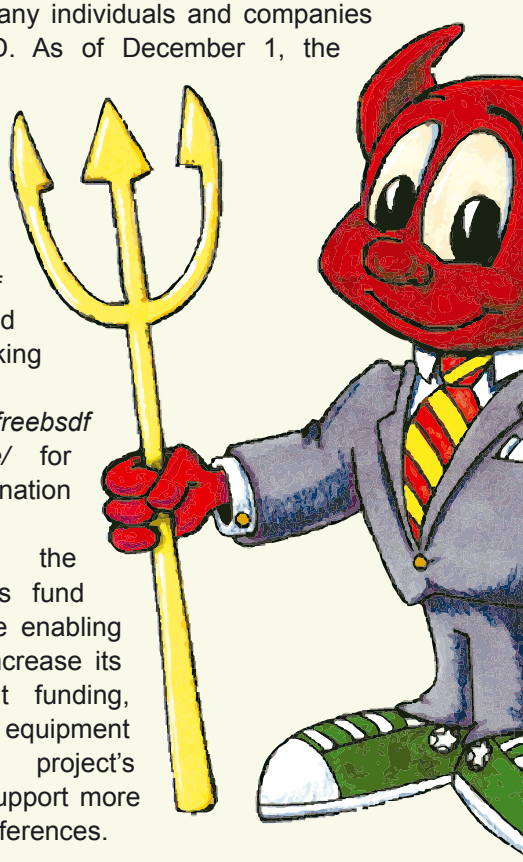
Each congestion control algorithm is implemented as a loadable kernel module. Algorithms can be selected to suit the application/network characteristics and requirements of the host's installation. The modular framework makes it much easier for developers to implement new algorithms, allowing FreeBSD's TCP to be at the forefront of advancements in this area, while still maintaining the stability of its network stack.

Other Projects

In addition to the development projects, the Foundation also negotiated a non-exclusive copyright license to the libcxxrt C++ runtime software from PathScale. This software is an implementation of the C++ ABI originally developed for Itanium and now used for the x86 family by BSD operating systems. libcxxrt will be available under the 2-clause BSD license. This implementation is a full replacement for the GNU libsupc++ library for platforms that use the Itanium C++ ABI, including i386 and x86-64, and will replace portions of the C++ stack previously only available under the GPL. It provides implementations of the dynamic features of C++, including dynamic casting, exception handling, and thread-safe static initializers, and will continue the gradual replacement of GNU toolchain and runtime components, furthering the aim of a purely BSD-licensed system.

The Foundation Needs Your Help!

The FreeBSD Foundation is entirely supported by donations. The Foundation is thankful for the support it receives from the many individuals and companies who value FreeBSD. As of December 1, the Foundation has raised $201,000 towards its 2011 goal of $400,000.



You can help us to reach our fund raising goal of $400,000 by the end of December by making a donation.

See *http://www.freebsdfoundation.org/donate/* for details about the donation process.

By helping the Foundation meet its fund raising goal, you are enabling the Foundation to increase its project development funding, purchase needed equipment for the FreeBSD project's infrastructure, and support more FreeBSD related conferences.

DRU LAVIGNE

Dru Lavigne is author of BSD Hacks, The Best of FreeBSD Basics, and The Definitive Guide to PC-BSD. As Director of Community Development for the PC-BSD Project, she leads the documentation team, assists new users, helps to find and fix bugs, and reaches out to the community to discover their needs. She is the former Managing Editor of the Open Source Business Resource, a free monthly publication covering open source and the commercialization of open source assets. She is founder and current Chair of the BSD Certification Group Inc., a non-profit organization with a mission to create the standard for certifying BSD system administrators, and serves on the Board of the FreeBSD Foundation.

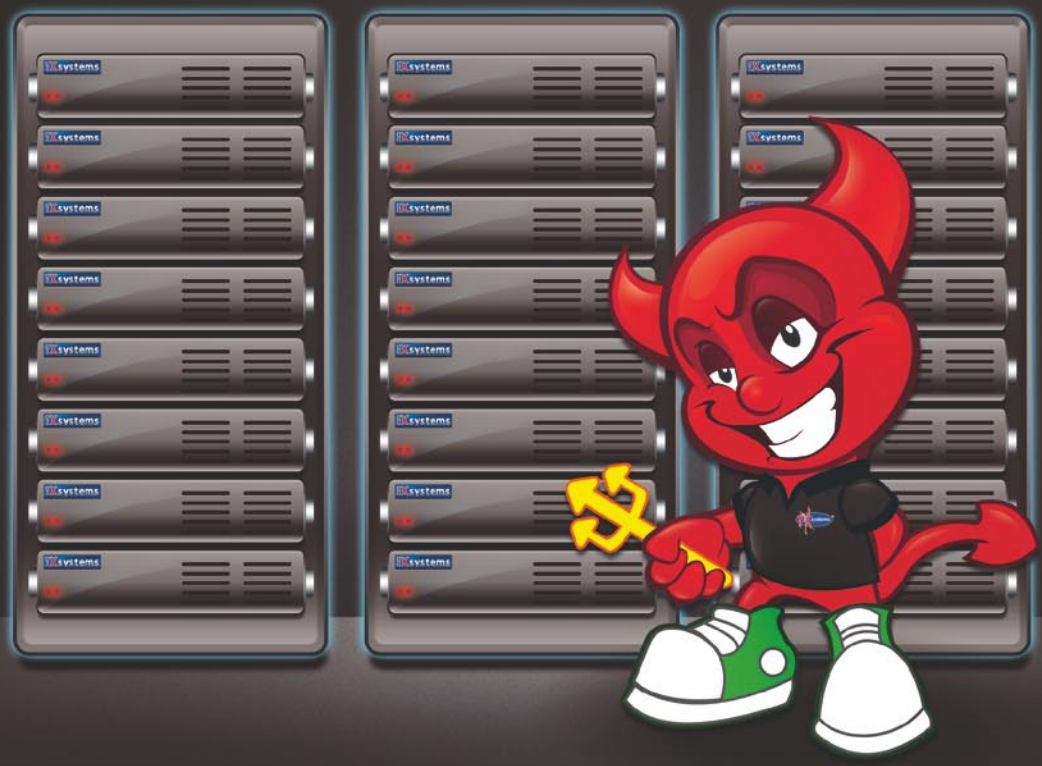
In the next issue:

- **More about security for admis**
- **Highly loaded WEB servers**
- **OSPFv6**
- **and Other !**

Next issue is coming in January!


Looking for help, tip or advice?
Want to share your knowledge with others?
BSD MAGAZINE
Give us your opinion about the magazine's content
and help us create the most useful source for you!

What has your server vendor done for BSD lately? Probably, not much.



Work with a vendor that supports the operating system you love!

iX is the corporate sponsor of the PC-BSD® Project, a major corporate donor to the FreeBSD Foundation, and leads the FreeNAS™ development team -- all while employing some of the most brilliant minds in the FreeBSD® community. For BSD hardware and software expertise, look no further.

1-855-GREP-4-IX
http://www.iXsystems.com/community

iXsystems®