MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

THE INEVITABILITY OF IPV6

INSIDE

VOL.4 NO.10
ISSUE 10/2011(27)
1898-9144

TrueNAS™ Pro Storage Appliance: You are the Cloud

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

Expansion Shelves Available

TrueNAS™ 2U Pro System

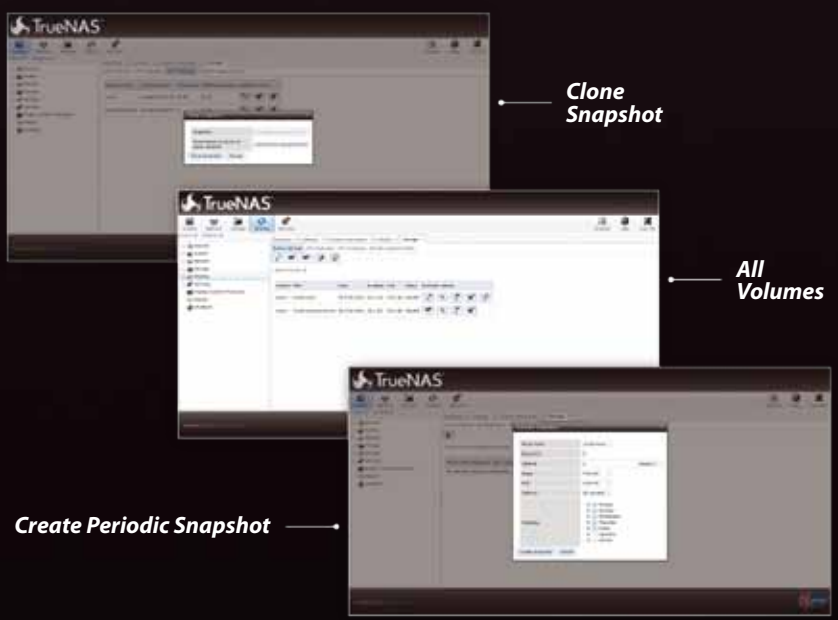
TrueNAS™ 4U Pro System

FreeNAS 8 POWERED

Storage. Speed. Stability.

In order to achieve maximum performance, the TrueNAS™ Pro 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™ Pro 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™ Pro Systems offer excellent capacity at an affordable price.

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



TrueNAS™ 2U PRO 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
- Up to 4.48TB of Fusion-io Flash Memory
- 2 x 1GbE Network Interface (Onboard) + Up to 4 Additional 1GbE Ports or Single/Dual Port 10GbE Network Cards

TrueNAS™ 4U PRO 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
- Up to 14.08TB of Fusion-io Flash Memory
- 2 x 1GbE Network Interface (Onboard) + Up to 4 Additional 1GbE Ports or Single/Dual Port 10GbE Network Cards

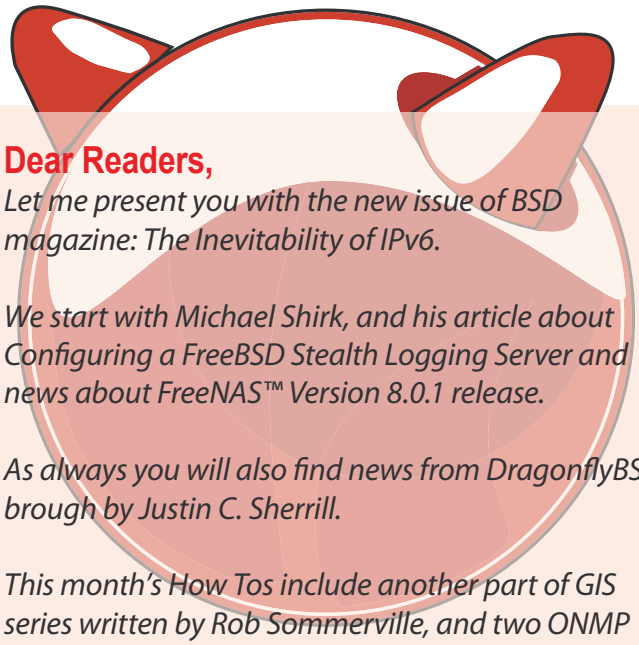
JBOD expansion is available on the 2U and 4U Pro Systems

** 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,

Let me present you with the new issue of BSD magazine: The Inevitability of IPv6.

We start with Michael Shirk, and his article about Configuring a FreeBSD Stealth Logging Server and news about FreeNAS™ Version 8.0.1 release.

As always you will also find news from DragonflyBSD brough by Justin C. Sherrill.

This month's How Tos include another part of GIS series written by Rob Sommerville, and two ONMP articles from Toby Richards. They are followed by Jasper Lievisse Adriaanese LibGTop article – a brief introduction to this handy library.

You will also find a piece of advice in protecting from DDoS attacks given by Stavros N. Shaeles – in Security section of the magazine.

In the end we present the cover story (or stories) of the isse – Inevitability of IPv6 written by Paul Ammann – two articles which will convince you that switch to IPv6 is Inevitable.

We all hope you will enjoy the reading and find it informative – make sure to make it before November issue hits!

Yours,
Zbigniew Puchciński
Editor in Chief
zbigniew.puchcinski@software.com.pl

MAGAZINE BSD

Editor in Chief:
Zbigniew Puchciński
zbigniew.puchcinski@software.com.pl

Contributing:
Michael Shirk, Justin C. Sherril, Rob Somerville, Toby Richards, Jasper Lievisse Adriaanese, Stavros N. Shaeles, Paul Ammann

Proofreaders:
Sander Reiche, Tristan Karstens

Special Thanks:
Denise Ebery

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

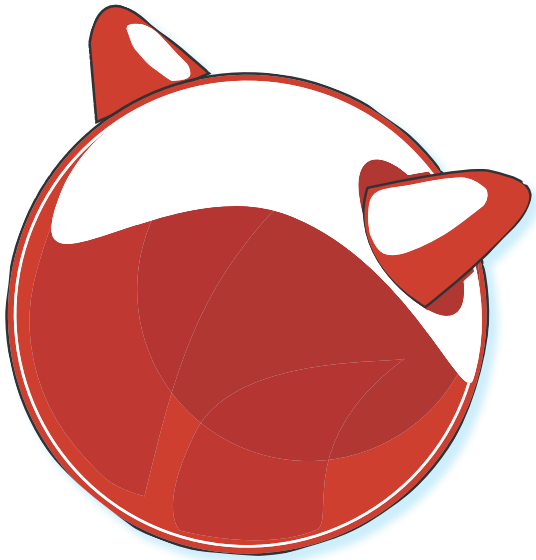
Executive Ad Consultant:
Ewa Dudzic
ewa.dudzic@software.com.pl

Advertising Sales:
Zbigniew Puchciński
zbigniew.puchcinski@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*

Contents



What's New

Developers Corner

How Tos

Tips & Tricks

Security

IPv6

FreeNAS

iXsystems Announces Release of FreeNAS™ Version 8.0.1

Release features back end changes and bugfixes, as well as new front end user features

The FreeNAS™ Team has announced the release of FreeNAS™ Version 8.0.1. FreeNAS™ 8.0.1 represents a major leap in functionality and stability for FreeNAS™ 8. Features added to FreeNAS™ in the 8.0.1 branch include S.M.A.R.T. and UPS services, USB 3.0 support, and OSX Lion AFP and Time Machine compatibility. In addition, cronjob support and rsync have been added to the GUI, and replication has been improved for increased data integrity.

In addition to the many back end changes and bugfixes, FreeNAS™ 8.0.1 also includes new front end user features. A new stoplight icon in the top right of the GUI functions as an alert system, keeping administrators in tune with the overall health of their installation. This icon is visible from every page of the GUI, and will change color in keeping with the condition of the system as indicated by the alert messages. Clicking the icon brings up a dialogue outlining which messages have keyed the alert.

The stoplight system will be most noticeable to new users and administrators booting a fresh install. As of 8.0.1, FreeNAS™ no longer has a default password, which will cause the alert light to flash red until one is added. This has the added security benefit of blocking SSH or root shell access until a root password is set by the administrator. The GUI also now includes a checkbox to set the root user shell password to be the same as the webGUI administrator's password, if desired.

8.0.1 includes another less immediately obvious, but still notable update – the ZFS deletion system now actually functions as a volume export utility. "Deleted" ZFS volumes can be added through the volume importer until the member disks are eventually reused in another volume. For the security-conscious, the GUI has an option to wipe the disks on deletion rather than leaving them usable, as well as an option to prevent the volume deletion from cascading over and affecting shares attached to the deleted volume.

Another important back end change in 8.0.1 is support for arbitrary mount points for UFS volumes. The size of the FreeNAS™ boot device no longer sets a cap on the size of the /var slice, if properly exported to another storage volume. While this only affects a small number of users in specific applications, this is an important milestone for users with large amounts of temporary data to cache, such as an Active Directory's 'users and groups' data.

"8.0.1 represents a significant advancement towards the goals outlined by the current FreeNAS™ roadmap," says Josh Paetzel, Director of IT at iXsystems. "With all the significant issues addressed, FreeNAS™ development will be able to better focus on total feature parity with

About FreeNAS™ 8

FreeNAS

FreeNAS™ is a free and open source Network Attached Storage operating system based on FreeBSD. The goal of the project is to design a lightweight, BSD-based software package that acts as a full featured NAS server, complete with a Django-based web user interface, full ZFS implementation, and the ability to interface perfectly with existing networks – regardless of operating system or protocol.

About iXsystems

iXsystems is the all-around FreeBSD® company that builds FreeBSD-certified servers and storage solutions, oversees FreeNAS™ development, and is the corporate sponsor of the PC-BSD® Project. iXsystems is an employee-owned and operated, open source-centric, customer focused organization, dedicated to providing the highest quality built-to-order enterprise rackmount server solutions, pre-configured server appliances, and scalable storage solutions to our customers around the globe.

Version .7, rather than just solid completion for existing new features."

Eventually, with the final release of FreeNAS™ 8.0.1, development will shift to the 8.1 branch, which will add a third-party plug-in system. The plug-ins will use a variation on the PBI system pioneered by PC-BSD®. Through plugins, FreeNAS™ 8 will be able to support most or all of the features that were part of FreeNAS™ .7 (such as BitTorrent and UPNP) while keeping the base install image slim for those who only want the core functionality of FreeNAS™. Version 8.1 will also feature a supported upgrade path from FreeNAS™ .7.x.

JOSH PAETZEL

Josh Paetzel – A 37 year old advocate, user and developer of BSD UNIX based systems. he resides in Minneapolis, Minnesota, USA where he hacks on FreeBSD and PC-BSD, both as a volunteer and as part of his full time work as the Director of IT at iXsystems.

Configuring

a FreeBSD Stealth Logging Server

The collection of log files provides security administrators with the ability to have an audit trail for the behavior of an information system. In the event that a system is compromised, remote logging provides a forensic trail to determine what occurred on the system.

What you will learn…

- A configuration for out-of-band remote logging

What you should know…

- Basic FreeBSD knowledge to navigate the command line
- Basic knowledge of how tcpdump and syslog work

The remote log files maintain the integrity of the original system logs as the compromised host can no longer be trusted. Going beyond a normal log server is the configuration of a stealth log server which doesn't interact with the network it is monitoring much like an intrusion detection system. Because the system is not accessible to the network, it is nearly impossible beyond physical access to compromise the logging system.

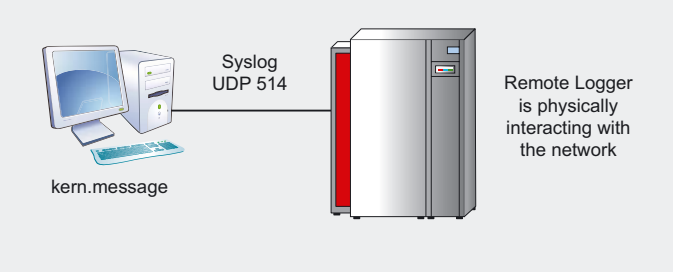


Figure 1. *Normal Syslog Remote Logging Setup*

Syslog has been the standard for system logging since its inception along side sendmail back in the 1980's. syslogd is normally the service used to handle the system logging in most *nix based operating systems. Updated services include syslog-ng and rsyslog which provide finer grained controls over the log messages. One of the important features of any syslog daemon is the ability to forward log files to a remote host. Normally, a remote

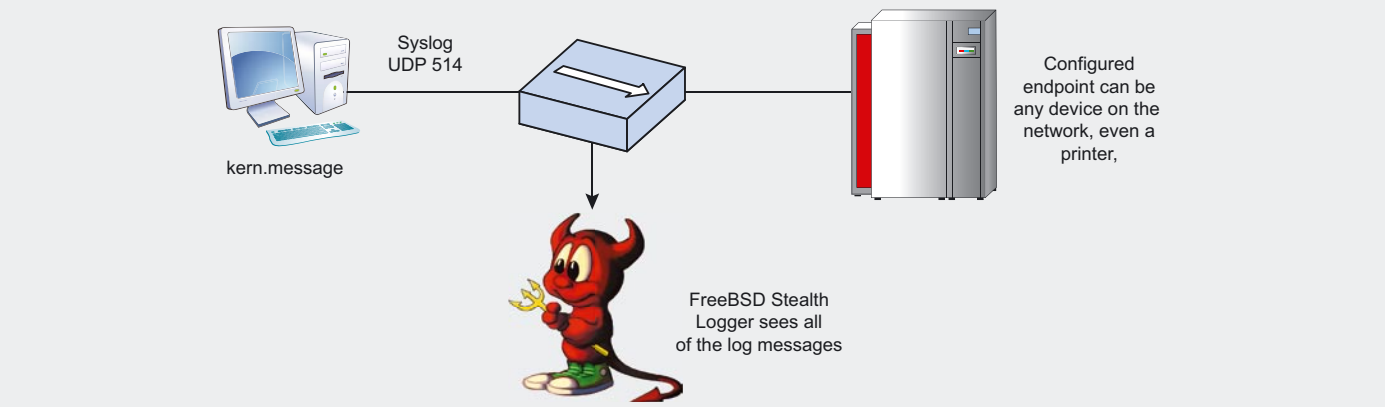


Figure 2. *Stealth Logger setup on Hub or SPAN port*

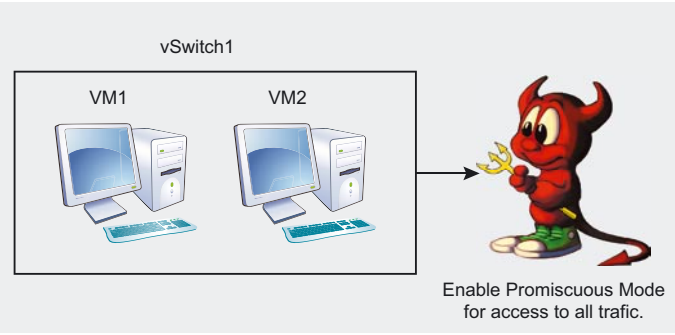


Figure 3. *VMware setup with Two VM's and the logging*

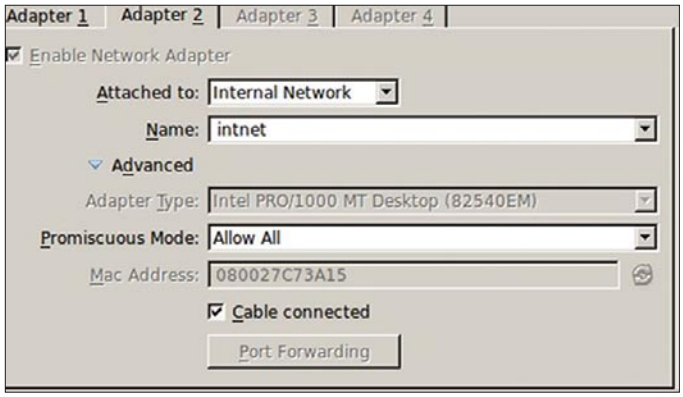


Figure 4. *Promiscuous Mode Settings in VirtualBox*

server accepts connections on UDP port 514 and writes out log files as show below in Figure 1.

If the system was compromised in this case, the remote logging server would have a record of the system logs before the attacker gained control of the system. The question is, what happens if the logging system itself is compromised? This is the same issue faced by intrusion detection and preventions systems in regards to establishing a separation between the monitoring and management network. It is a mistake to have interfaces configured on the same network that is being monitored due to the risk of possible exploitation of a vulnerability giving access to backdoor the system. One solution is

Listing 1. *The following steps makes sure the em1 interface is in promiscuous mode without an IP address upon*

```
ifconfig em1 promisc up
echo 'ifconfig_em1="promisc up"' >> /etc/rc.conf
```

Listing 2. *The following is the stealth_logger script that is called as a cronjob. The interface is passed into the script. The example interface here is em1*

```
#!/bin/sh
# Kill all TCPDUMP processes
/usr/bin/killall -9 tcpdump 2>/dev/null
# Start off with date setup and make a directory for each hour
YEAR='date "+%Y"';
MONTH='date "+%m"';
DAY='date "+%d"';
HOUR='date "+%H"'
MIN='date "+%M"'
WORKDIR="/var/log/stealth_logger/$YEAR/$MONTH/$DAY/$HOUR"
mkdir -p $WORKDIR
# Read interface to listen on
INTF=$1
# Run packet capture for syslog packets, saving at max
# 50 20MB pcap files per hour
(tcpdump -C 20 -W 50 -w $WORKDIR/SYSLOG_ -XXnns 0 -i $INTF udp and port 514)
echo
echo "Log Started: $MONTH/$DAY/$YEAR $HOUR:$MIN"
echo

# Exit to make cron happy  exit 0
```

Listing 3. *Run the following to add a cronjob to rotate the packet capturing every hour. The example interface used here is em1*

```
echo "# Rotate the packet capture every hour" >> /etc/crontab
echo "0  *  *  *  *  root   exec /usr/local/bin/stealth_logger em1" >> /etc/crontab
```

Listing 4. *Example parsing output of Syslog data:*

```
NOTICE: Sep  4 11:15:23 ubuntu-server su[4764]: - /dev/tty1 testuser:root
NOTICE: Sep  4 11:15:23 ubuntu-server su[4764]: FAILED su for root by testuser
NOTICE: Sep  4 11:15:24 ubuntu-server su[4765]: pam_unix(su:auth): authentication failure; logname=root uid=1000
                euid=0 tty=/dev/tty1 ruser=testuser rhost=  user=root
NOTICE: Sep  4 11:15:26 ubuntu-server su[4765]: - /dev/tty1 testuser:root
NOTICE: Sep  4 11:15:26 ubuntu-server su[4765]: FAILED su for root by testuser
```

to setup a stealth logging server with an interface in promiscuous mode to sniff the syslog packets out-of-band. This is shown in Figure 2.

Because of the UDP protocol being connectionless, the destination of the syslog messages can be any type of device, even a printer. The promiscuous interface on the stealth log server will receive all of the traffic. An extra security step for the paranoid is to disable the transmit pair on the Cat5 cable, preventing any chance of the server sending packets out.

In the case of virtual machines, a FreeBSD VM can be given an interface with Promiscuous Mode in VMware or Virtualbox to allow all of the traffic on the virtual switch to be monitored. Figure 3 gives the example for VMware.

The first thing that needs to be completed is the install of FreeBSD. All of the steps listed were performed on a Virtual Machine with a FreeBSD minimal install with the ports tree (See FREEBSD-INSTALL for installation instructions). Using VirtualBox, navigate to the Settings->Network->Advanced as shown in Figure 4.

Once this has been completed, startup the VM and login as root. All of the commands are to be run with an administrative account (using sudo if preferred). Run the commands in Listing 1 to enable promiscuous mode for the interface to be used. In this example, the Stealth Logger is connected to an Internal network with several other devices on interface em1.

Listing 2 is a simple script to log syslog packets on UDP 514 into a directory structure based on the year/month/day/hour. Running this in cron hourly will keep a record for each hour of log data.

Listing 3 is the process to add the hourly log rollover for tcpdump which will create is a simple script to log syslog packets on UDP 514 into a directory structure based on the `year/month/day/hour`. Running this in cron hourly will keep a record for each hour of log data.

The stealth log server will continue to collect any syslog traffic that is seen and log it into `/var/log/stealth_logger` (with the default script settings). In a later article, additional details will be provided for setting up Snare on Microsoft Windows and `syslog-ng/rsyslog` on other BSD and Linux operating systems to send to the log server. In addition to the configuration, parsing tools will be demonstrated to utilize the log data. Example output from the syslog data is displayed in Listing 4. In this example, the testuser has failed to login as root.

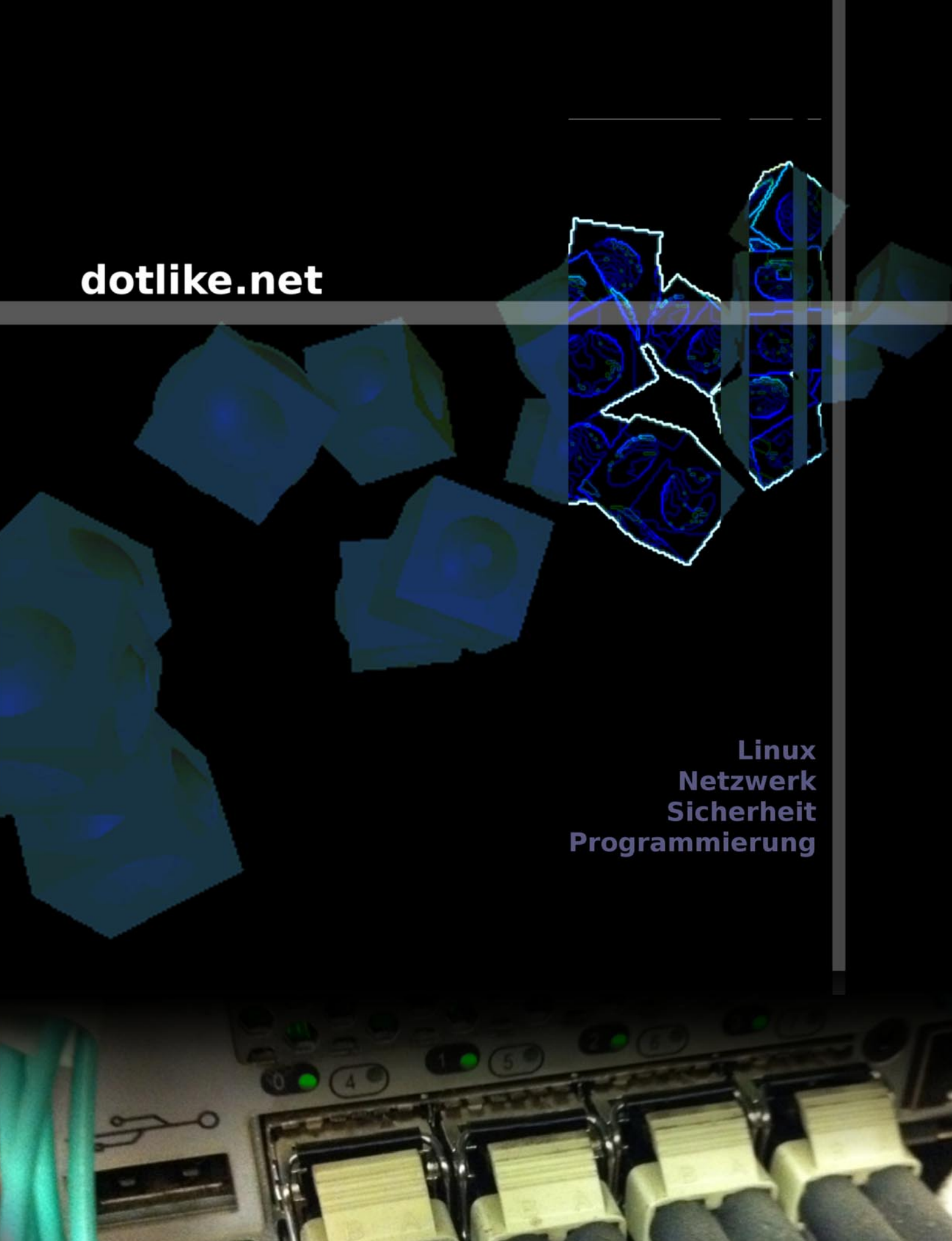
References

- FREEBSD-INSTALL: *http://www.freebsd.org/doc/handbook/install-start.html*
- VirtualBox: *http://www.virtualbox.org*

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. He wishes to thank Thomas Conway and J.J. Cummings for testing the instructions in this article.

dotlike.net



Linux
Netzwerk
Sicherheit
Programmierung

Recovering data with hammer

It's been a while since we had a straightforward news report for DragonFly; the time since then has been filled with reports on Hammer and bulk pkgsrc builds.

In fact, we've managed to cover the whole space between releases, since the last news report like this in BSD Magazine was just after the 2.10 release of DragonFly. DragonFly follows an even/odd stable/development cycle. 2.11 is the development version at this time. DragonFly 2.12 is due to start the release process very soon, and the development version will be 2.13.

You can think of this as a *what's in the next release of DragonFly* report. I'm totally going to use these notes when writing the 2.12 release notes, in fact.

Encryption

The encryption framework in DragonFly has seen a major upgrade. Alex Hornung has added libdm, a BSD-licensed equivalent of Linux's libdevmapper, and a utility called tcplay. This new utility is compatible with TrueCrypt, so you can create encrypted volumes, hide them, and so on. See truecrypt.org for more details on what is supported. In any case, encrypting your date automatically and reading/writing it as encrypted data on the fly is now possible.

Google Summer of Code results

DragonFly participated in Google Summer of Code for the 4th year in a row, supplying mentors for college student working on DragonFly-linked projects.

We had 6 projects total, with 5 of them passing. (One student went AWOL at the end.) Some of the code has made it to DragonFly, and should show up in the 2.12 release.

Here's all the finished projects.

- Bring kernel event notification in DragonFly BSD to its logical conclusion *Samuel Greear*

Figure 1. *Fuzzer*

- Implementing a mirror target for device mapper *Adam Hoka*
- Improve dsched interfaces and implement BFQ disk scheduling policy *Brills Peng*
- Port PUFFS from NetBSD/FreeBSD *Nick Prokharau*
- Porting Virtio Drivers from NetBSD to DragonFly BSD to speed up DragonFly BSD as a KVM guest *Stephanie Ouillon*

pkgsrc

The next quarterly release of pkgsrc is due very soon. It will be 2011Q3, for all platforms that pkgsrc supports, including DragonFly. As is customary, binary packages for DragonFly will be built to go with the 2.12 release, though they may not be available at release time. It usually takes several weeks to build the 10,000+ packages in pkgsrc.

The packages from pkgsrc-2011Q1 are still the default for DragonFly, both for binary installs and from source builds. Why haven't we switched to 2011Q2? There's a slight chicken-and-egg problem. Pkgsrc will not install binary packages built with a newer version of `pkg_install`, so downloading and installing packages from a newer quarterly release will fail, with an error. (Get ready for a digression.)

DragonFly has a tool that comes with the install, called `pkg_radd`. It sets `$PKG_PATH` to the appropriate path on a remote server for that system's release version and processor architecture, and downloads pkgsrc binaries using pkgsrc's `pkg_add`. This is very nice for installation, and even upgrading installed packages, but changing the default packages for DragonFly to a new quarterly release means that any further binaries installed will error out on that `pkg_install` version issue.

The fix is to force-upgrade `pkg_install`, since binary packages usually exist for it, or if building from source, use 'bmake package' and upgrade using the binary built from that. As an alternative, the binary package management tool, pkgin, will be able to handle this circumstance in version 0.5, though that version will not make it into DragonFly 2.12.

In other news, DESTDIR support in pkgsrc is almost complete for all packages. Support of DESTDIR means that packages can be installed as non-root, and the remaining 40 or so stragglers are mostly software no longer maintained by the original creators.

Java users will notice the 1.6 JDK now runs on DragonFly, as does OpenJDK7, thanks to Francois Tigeot. (OpenJDK is i386 only)

Hardware support

There's been updates for various network card and other hardware in the time between DragonFly 2.10 and now, either original or brought in from other BSDs. Broadcom and Marvell now have more supported chipsets, thanks to the efforts of Michael Neumann and Sepherosa Ziehau. If that's still leaving you with an unsupported network card, Sascha Wildner has updated ndis, which may support an otherwise unavailable network card by use of information from the Microsoft Windows version of a driver. Sascha Wildner has also updated the drivers for the LSI MegaRAID SAS 92xx series of RAID cards, along with the HighPoint RocketRAID. The SafeNet crypto hardware accelerator chip is supported now too. DragonFly's interrupt routing has been thoroughly upgraded by Sepherosa Ziehau, so newer models that do not work for you with DragonFly 2.10, or did not play well with different ACPI modes, may perform better. Also, as a sign of the times, support for certain ISA devices was removed entirely. Some of these devices were in the default kernel config, so remove them manually if you use a custom kernel configuration and are upgrading from 2.10 to 2.12. Goodbye (most of) ISA; nobody will miss it at this point.

Benchmarks of 2.10 vs. 2.11

Francois Tigeot ran some disk benchmarks, comparing HAMMER on DragonFly 2.10, Hammer on DragonFly 2.11, and ZFS on OpenIndiana. The ZFS numbers show a difference in activity. The performance difference between the two different version of DragonFly is noticeable, and should be a good experience for anyone upgrading.

JUSTIN C. SHERRILL

Justin Sherrill has been publishing the DragonFly BSD Digest since 2004, and is responsible for several other parts of DragonFly that aren't made out of code. He lives in the northeast United States and works over a thousand feet underground.

Using Openmaps data with Geoserver

In this article in our GIS series, we will examine how to import Openmaps data

Open Street Map (openstreetmap.org) founded in July 2004 by Steve Coast, is a treasure trove of worldwide street maps available under the Creative Commons licence.

What you will learn…

- How to create street maps of any region of the world

What you should know…

- Basic FreeBSD administration skills, Previous FreeBSD GIS tutorials in this series

Unfortunately, some of the maps do not give complete coverage so consideration should be given to the suitability of using this data in mission critical or production environments. That said, living in a fairly remote part of the UK I was pleasantly surprised by the accuracy of the street map, I was expecting many more errors than I found, mostly missing street names from areas well off the beaten track.

The sheer quantity of map data available is enormous – the full planet PBF file is 14Gb which expands to > 110Gb when extracted, so unless you have lots of storage, bandwidth and time, a subset is a more practical approach. Weekly updates are available in diff format. For this article, I have used map data for Kentucky, which was a reasonable 220Mb uncompressed. Even that data set pushed my Virtualbox Geoserver to the limit as I only have a twin processor PC with 4GB of RAM. There is an OSM plugin available for QGIS covered in the previous article, so the map data can be manipulated albeit in a rudimentary fashion as the plugin is still in the early stages of development.

As a lot of this code will not fit easily on a page, I am using the convention ? to denote a carriage return.

Listing 1. *Installing bzip2*

```
pkg_add -r  bzip2
```

Listing 2. *Extracting the files*

```
cd /geodata/OSM
bunzip2 kentucky.highway.osm.bz2
bunzip2 kentucky.administrative.osm.bz2
bunzip2 kentucky.coastline.osm.bz2
```

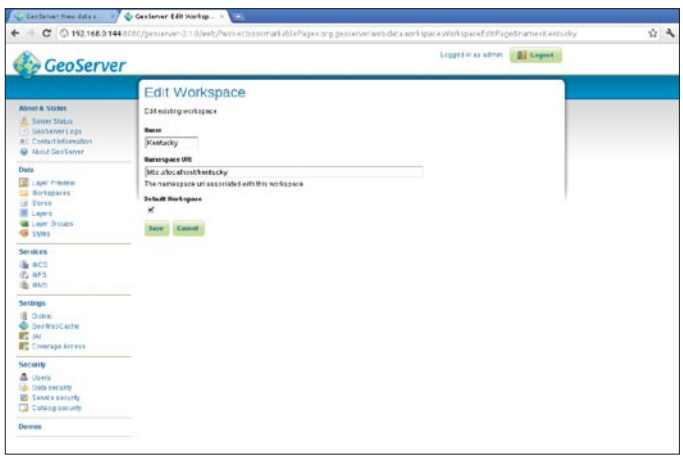


Figure 1. *Creating the Workspace*

Pre-requisites

You will need a working Geoserver installation with Postgres / PostGIS extensions and optionally QGIS running in a workstation for editing the map data.

Choosing and downloading your maps

Visit *http://downloads.cloudmade.com* and download the OSM map of your choice. Cloudmade also has TomTom and Adobe Illustrator maps available, but we will be using

Listing 3. *Sample OSM XML file*

```
head kentucky.osm
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="Osmosis 0.36">
<bound box="36.39672,-89.67530,39.24822,-81.85895" ◄    origin="http://www.openstreetmap.org/api/0.6"/>
<node id="300559" version="1" timestamp="2005-12-08T19:18:38Z" ◄
 changeset="1000" lat="38.2830757" lon="-85.9401045"/>
<node id="14827169" version="2" timestamp="2010-11-14T18:44:23Z" uid="9176" ◄    user="Maarten Deen"
                changeset="6369617" lat="39.031468" lon="- ◄   84.575264"/>
<node id="14832854" version="2" timestamp="2010-11-14T18:44:23Z" uid="9176" ◄    user="Maarten Deen"
                changeset="6369617" lat="38.989377" lon="- ◄   84.577611"/>
<node id="16249577" version="2" timestamp="2010-07-24T00:16:43Z" ◄
      uid="120468" user="Gone" changeset="5300041" lat="39.110063" lon="-◄ 84.502348">
```

Listing 4. *Updating the ports tree and installing OSM2PGSQL*

```
portsnap fetch
portsnap update
cd /usr/ports/converters/osm2pgsql/ && make install clean
```

Listing 5. *Installing LIBTOOL*

```
pkg_delete -f libtool-2.2.10
cd /usr/ports/devel/libtool/ && make install clean
ln -s /usr/local/share/osm2pgsql/default.style ◄
               /usr/local/share/default.style
```

Listing 6. *Creating the database*

```
su pgsql
createdb OSM
createlang plpgsql OSM
psql -d OSM -f /usr/local/share/postgis/contrib/postgis-1.5/postgis.sql
psql -d OSM -f /usr/local/share/postgis/contrib/postgis-1.5/spatial_ref_sys.sql
exit
```

Listing 7. *Importing the data*

```
/usr/local/bin/osm2pgsql -d OSM -U pgsql kentucky.highway.osm
/usr/local/bin/osm2pgsql -d OSM -U pgsql kentucky.administrative.osm
/usr/local/bin/osm2pgsql -d OSM -U pgsql kentucky.coastline.osm
```

the OSM format for import into Geoserver. Once you have downloaded transfer the .bz2 archives across to the Geoserver box using SSH or MC etc. In this example I have placed them in the `/geodata/OSM` directory.

Extracting and converting the files

Install bzip2 using the package manager (Listing 1). Extract the archives (Listing 2). Examining the files we will find that they are in standard XML format (Listing 3).

We now need to install osm2pgsql to import the files into Postgresql (Listing 4).

If you receive an error concerning the libtool version, you will need to upgrade libtool to version 2.4 (Listing 5).

Create the database in Postgres and make it spatially aware (Listing 6).

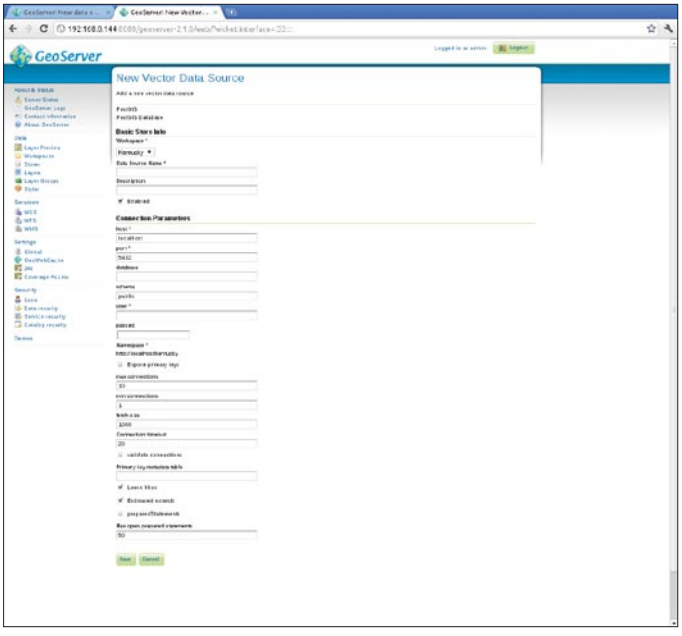


Figure 2. *Creating the Data Source*

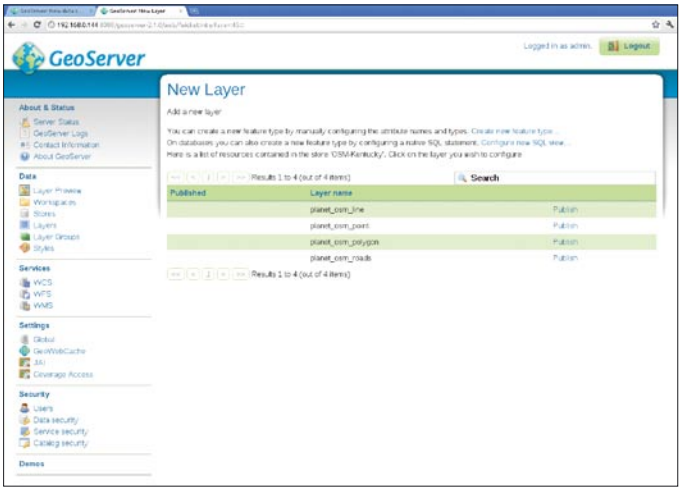


Figure 3. *Prompt to publish after creating store*

The next step is the actual import itself. The resulting XML files will be resident in the OSM database (Listing 7).

Table 1. *Postgis Store parameters*

Field	Value
Workspace	Kentucky
Data Source Name	OSM-Kentucky
Description	OSM Kentucky Data
Host	localhost
Port	5432
Database	OSM
Schema	public
user	pgsql
Passwd	Your PGSQL password (I used pgsql in the demo)

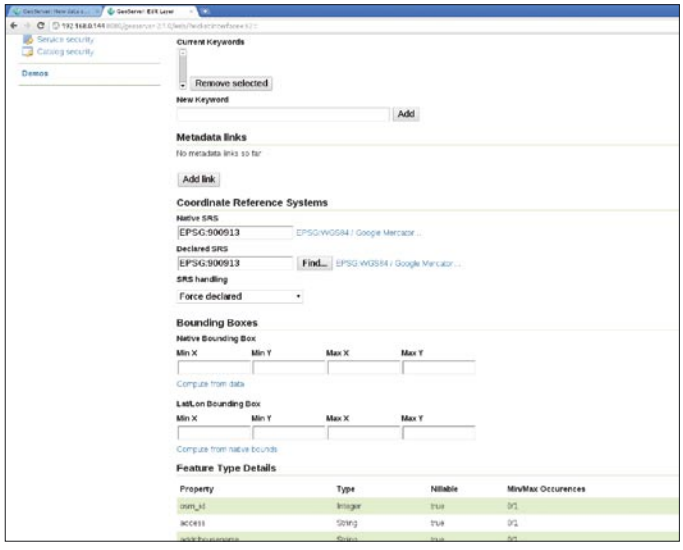


Figure 4. *Computing the Bounding Box*

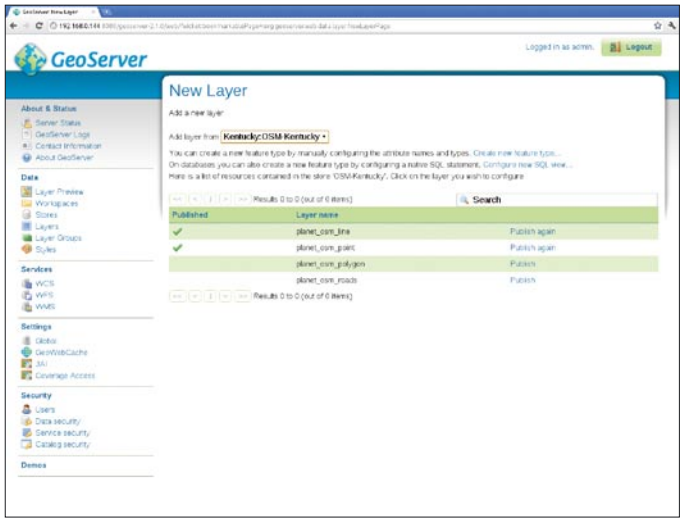


Figure 5. *Repeat and publish each layer*

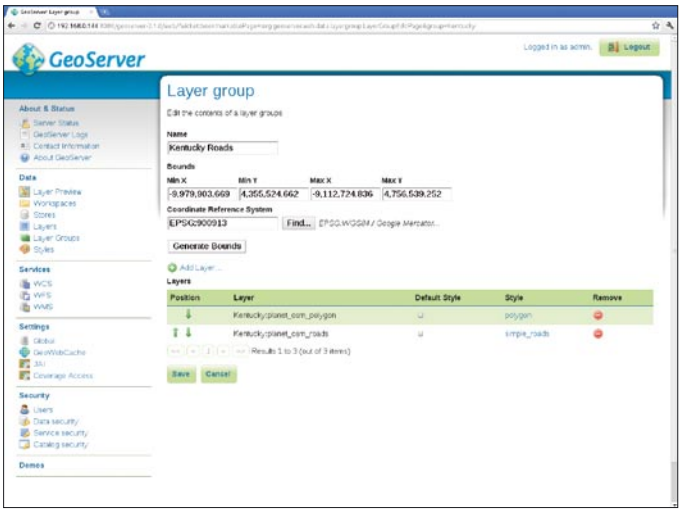


Figure 6. *Creating a layer group with polygon and roads*

The open street maps data will now have been imported into Postgresql. This will take about 20 minutes depending on the speed of your server.

Configuring Geoserver

Create a workspace called Kentucky with a dummy url pointing to *http://localhost/kentucky* (Figure 1).

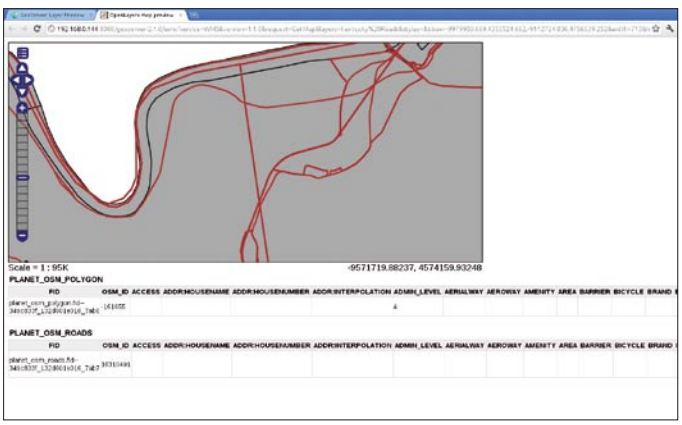


Figure 7. *Polygon and Roads layer for Kentucky*

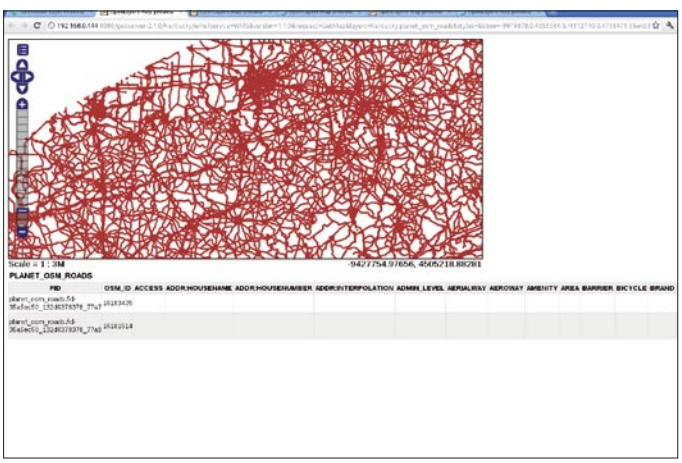


Figure 8. *Roads layer for Kentucky*

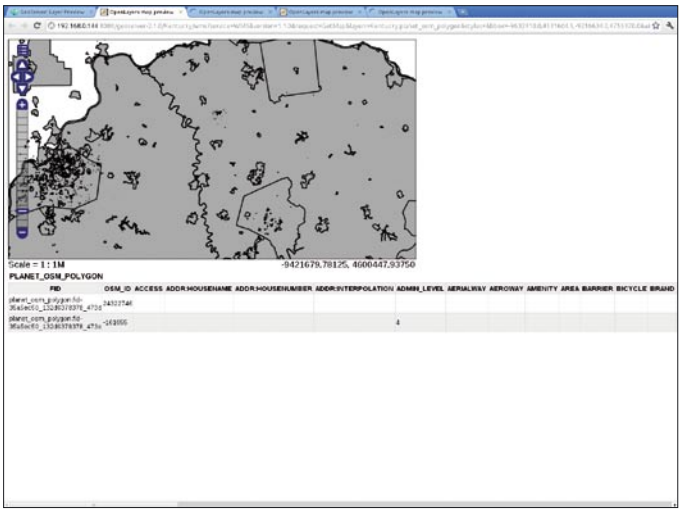


Figure 9. *Polygon layer for Kentucky*

Create the PostGIS Vector datastore for the Kentucky Workspace (Figure 2, Table 1).

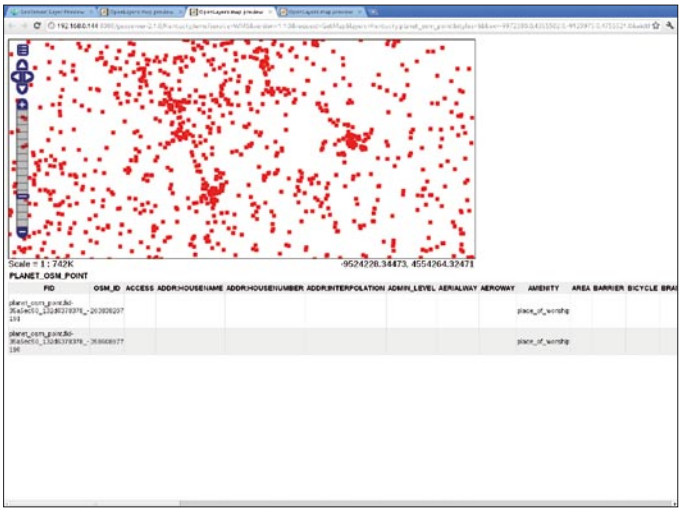


Figure 10. *Points layer for Kentucky*

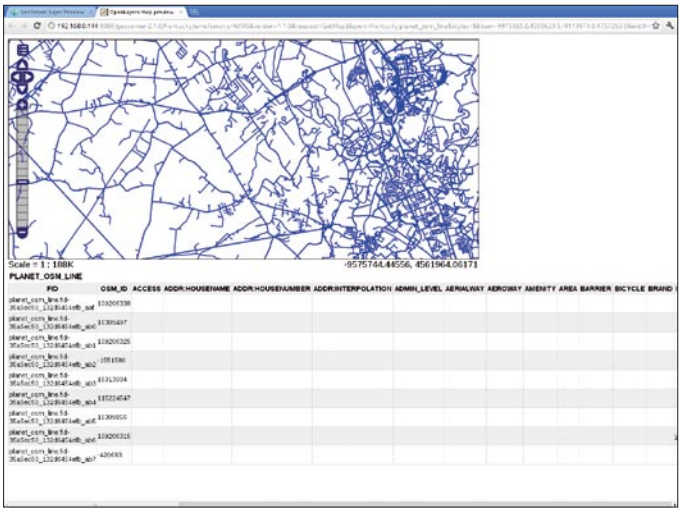


Figure 11. *Lines layer for Kentucky (Zoomed in)*

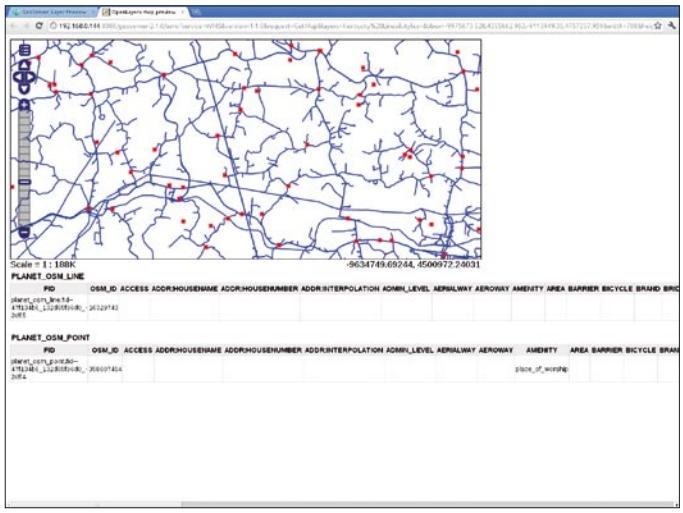


Figure 12. *Lines and Points layer (X2 CPU)*

Geoserver will now prompt you to publish the layers. Publish each layer in turn, computing both bounding boxes Lat/Lon and Native. N.B: If you find that the layer preview seems inaccurate, recalculate bounding box. I found this cured an inaccuracy in the layers. I also downloaded and imported the kentucky.osm, but from what I can see this is the complete set of maps, roads and lines etc. and doesn't need to be loaded. You will have to revisit the layers and add the remaining 3 layers from Kentucky: OSM-Kentucky (Figure 3 – 5).

Layer groups

If you have lots of processing power at hand, you can create a layer group (Figure 6). You may have to reorder the layers accordingly, so the correct layer in on top. On my VM, I was only able to group together polygon and roads before Geoserver gave up after 60 seconds trying to serve the map (Figure 7). While you can adjust the time-out value in `../data/wms.xml`, maybe I should have picked a smaller US state!

Regardless of layer groups, the layers are now ready for styling, which was covered in a previous article (Figure 8 – 11).

I did finally manage to get my PC to process the lines and points group layer, but I had to add an extra CPU to the VM (Figure 12).

ROB SOMERVILLE

Rob Somerville has been passionately involved with technology both as an amateur and professional since childhood. A passionate convert to *BSD, he stubbornly refuses to shave of his beard under any circumstances. Fortunately, his wife understands him (she was working as a System/36 operator when they first met). The technological passions of their daughter and numerous pets are still to be revealed.

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!

ONMP on OpenBSD 4.9

OpenBSD is my BSD of choice. In fact, it is my OS of choice wherever possible. I always challenge those who disagree with me to name another OS with a similar track record for security.

What you will learn…

- How to build an OpenBSD/Nginx/MySQL/PHP (ONMP) server from a freshly installed OpenBSD system.

What you should know…

- How to use the command line.
- How to set environment variables.
- The difference between OpenBSD vs. other BSD's.

We've all heard of LAMP (*Linux Apache MySQL PHP*). My web server of choice happens to be Nginx, not Apache. My BSD server in the cloud isn't very beefy. It's a VPS with 512MB RAM. Nginx, being much easier on resources than Apache seems to be the best choice for me. Creating an *OpenBSD Nginx MySQL PHP* (ONMP) server was my first goal upon starting to teach myself OpenBSD.

Before we begin a by-the-numbers tutorial on creating an ONMP server, I'd like to give a plug for my hosting provider: *bsdvm.com*. This is the only BSD hosting provider that I could find who gives you access to the VMware console to your server. This makes it easy to re-install your OS from scratch, and specifically customized for your own needs.

Let's get started

Step 1

Let's install MySQL, wget, PHP (Fast CGI), and several core PHP modules from the packages system. Users of other BSD systems will be appalled that I'm not using the ports. Unlike certain other BSD's, OpenBSD recommends packages over ports. Be sure to have set your `PKG_PATH` environment variable:

```
# pkg_add wget mysql-server php5-core php5-fastcgi php5-
                    mysql php5-mysqli \
php5-pdo_mysql php5-mcrypt php5-mhash
```

I'm trying to run this as lean as possible. I chose not to (for now) install the popular (but very large) php5-mbstring which gives PHP unicode support.

At the moment, I don't plan on needing to serve up any language or symbol that isn't included in ASCII.

Step 2

Fix MySQL & PHP discrepancies.

Step 2a

Create the default databases because `pkg_add` didn't do that for you.

```
# mkdir /var/mysql && chown -R _mysql:_mysql /var/mysql
# mysql_install_db
```

Step 2b

Enable the PHP modules. The official documentation says to make symbolic links.

I prefer to copy the files so that I can always reference the original sample files.

```
# cp /var/www/conf/php5.sample/*.ini /var/www/conf/php5/
```

Step 2c

Uncomment `#cgi.fix_pathinfo=0` in `/var/www/htdocs/conf/php.ini`.

Step 3
Install Nginx. Unfortunately, OpenBSD 4.9's Package/Ports system comes with a pre 1.0 version of Nginx. I don't like that, so I'm going to compile Nginx 1.0.6 from source:

Step 3a
First we need pcre

```
# pkg_add pcre
```

Step 3b
Install Nginx with OpenSSL in case we want to use certificates later.

```
# cd ~
# wget http://nginx.org/download/nginx-1.0.6.tar.gz
# tar xvfz nginx-1.0.6.tar.gz
# rm -f nginx-1.0.6.tar.gz
# cd nginx-1.0.6
# ./configure --with-openssl=/usr/include/openssl
# make && make install
```

Step 4
Reconcile OpenBSD's html root with Nginx's. Nginx puts the html root at `/usr/local/nginx/html`. OpenBSD (and the PHP package) expect `/var/www/htdocs/`. There are many ways that you might choose to fix this, but the easiest is to simply create a symlink:

```
# rm -Rf /var/www/htdocs
# ln -s /usr/local/nginx/html /var/www/htdocs
```

Step 5
Configure Nginx for PHP.

Step 5a
Uncomment the following lines in `/usr/local/nginx/conf/nginx.conf` except for `root html;`

```
#location ~ \.php$ {
```

Do not uncomment this line: `# root html;`

DO uncomment these lines

```
# fastcgi_pass 127.0.0.1:9000;\
# fastcgi_index index.php;\
# fastcgi_param SCRIPT_FILENAME /scripts$fastcgi_script_
                    name;
# include fastcgi_params;\
#}
```

Step 5b

Change `fastcgi_param SCRIPT_FILENAME /scripts$fastcgi_script_name;` to `fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;` in `/usr/local/nginx/conf/nginx.conf`.

Step 5c

Insert `fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;` just above `fastcgi_param SCRIPT_NAME $fastcgi_script_name;` in `/usr/local/nginx/conf/fastcgi_params`.

Step 5d

The Nginx official pitfalls page (*http://wiki.nginx.org/Pitfalls*) Section 1.2 (as of this writing) tells us where to put the `root html;` directive. Comment out every instance of `root html;`, and then insert that directive just below these lines:

```
server {
    listen 80;
    server_name  localhost;
```

Step 5e

The pitfalls page also recommends that we move the index directive to avoid needing multiple index directives later. Comment out the line in `/usr/local/nginx/conf/nginx.conf index index.html index.htm;` Add this modified version of the line (which includes `index.php`) just below the `http {` line: `index index.php index.html index.htm;`.

Step 6

Configure Nginx to start at boot time. I spent some time trying to figure out how to jail Nginx with chroot. I couldn't make it work because it always complained of not having access to the library files in various `/usr/` subfolders that it needed. I suppose that's ok because while the Nginx master process runs as root, then Nginx worker process runs as *nobody*. Add the following to `/etc/rc.local` (unlike with GNU/Linux, `/etc/rc.local` is the official way to start custom daemons in OpenBSD):

```
if [ -x /usr/local/nginx/sbin/nginx ]; then
    echo -n ' nginx'; /usr/local/nginx/sbin/nginx
fi
```

Step 7

Configure PHP-FastCGI to start at boot time. We can't jail PHP to a particular directory, but we can use chroot to make PHP run as *nobody*. Add this to `/etc/rc.local`:

```
chroot -g nobody -u nobody / env -i PHP_FCGI_CHILDREN=5 \
PHP_FCGI_MAX_REQUESTS=1000 \
/usr/local/bin/php-fastcgi -q -c /etc/php5 -b 127.0.0.1:9000 &
```

Step 8

Configure MySQL to start at boot time. I really don't know why, but the default scripts from packages in `/etc/rc.d`. don't work. In fact, through four re-installations of OpenBSD 4.9, I have yet to see any script from any package in `/etc/rc.d` function. We need to start everything in `/etc/rc.local` instead (even though we're launching MySQL as root, `mysqld_safe` will automatically run as `_mysql`:

```
# rm -f /etc/rc.d/mysqld
```

Add the following to `etc/rc.local`:

```
if [ -x /usr/local/bin/mysqld_safe ]; then
    echo -n ' nginx'; /usr/local/bin/mysqld_safe &
fi
```

Step 9

Let's let `/etc/rc.local` do it's magic now. Reboot. Then re-logon, and gain a root prompt (sudo -s).

Step 10

Let's give the root user of MySQL a password:

```
# mysqladmin -u root password <password>;
```

Step 11

Check Nginx & PHP. We're going to create a `phpinfo()` file. **WARNING!** This is insecure. Having a phpinfo file is a security risk. Do not host this file in a production environment!:

```
# echo „<?php phpinfo(); ?>" > /var/www/htdocs/phpinfo.php
```

Now… from your laptop or whatever, go to `http://<your server>/phpinfo.php`. If you've done everything right so far, then you see a nice web page that tells you all about your server's PHP configuration.

Congratulations. You have a working and secure ONMP server!

TOBY RICHARDS

Toby Richards has been a network administrator since 1997. He considers himself to be a jack of all operating systems, but a true master of none. He feels this to be a mastery in its own right since he understands principles that are common to all operating systems. His articles are the product of teaching himself to become better with OpenBSD and PC-BSD. He simply writes about what he has learned most recently. For a hosting provider, he highly recommends bsdvm.com. They give you access to your VMware console so that you can re-install your OS at will, and with the settings of your own choosing.

a-team systems

FreeBSD Server Specialists

Over 15 Years of Professional FreeBSD® Experience!

http://www.ateamsystems.com/

Full Server Management

- Support for Web & DB Servers
- Troubleshooting
- OS & Security Updates
- Encrypted Off-Site Backups
- Scripting & Automation

Server Monitoring

- Custom Tailored Plans
- 24x7 On Call Available
- Performance Trends
- Report Email Monitoring
- Kernel Log & HW Monitoring

We are a FreeBSD focused shop:
FreeBSD is not just another bullet point for us!

OSSEC on OpenBSD (ONMP) 4.9

It is worth saying up front that these instructions assume that you're running Nginx compiled from source vice Apache or Nginx from Ports or Packages.

What you will learn…

- How to harden your server with a Host Intrusion Prevention System

What you should know…

- Command Line BSD
- An ability to understand basic attacks... like why http://yourserver.com/../../etc/master.passwd is a malicious request.
- A desire for a daemon to prevent malicious activity

Anyone comfortable on the *nix command line ought to easily know how to modify this guide to suit his or her particular operating system & choice of web server software.

OSSEC is a *host intrusion prevention system* (HIPS). It is open source, and sponsored by Trend Micro. It can notify you when important files change. It can temporarily (10 minutes by default) block IP addresses that do questionable stuff like:

- Try to browse to URL's with `/../` in them, such as `http://yourserver/../../etc/master.passwd` ten times in two minutes or less.
- Enter bad username/password combos via SSH ten times in two minutes or less.
- Invoke ten or more 40x and/or 50x errors in two minutes or less.
- Lots of other bad guy activity…

OSSEC is not necessarily BSD specific, but since OpenBSD's primary focus is security, then what can be more OpenBSD than even more security?

Of course, all that *ten times in two minutes or less* is customizable as well. The installation is pretty straightforward. Download the tarball. Extract it. Run the included install.sh script. Now we have some tweaking to do.

First, we need to edit `/etc/pf.conf` so that OSSEC can block bad IP addresses. Add the following just under *set skip on lo* in your `/etc/pf.conf` file:

```
table <ossec_fwtable> persist #ossec_fwtable
block in quick from <ossec_fwtable> to any
block out quick from any to <ossec_fwtable>
```

Wordpress Users

OpenBSD's PHP package comes with something called Suhosin to harden PHP. One of the things that Suhosin prevents is any PHP script from changing the maximum memory setting. A file in WordPress does this. We need to prevent WordPress from doing this, or else OSSEC will block OUR IP address when we log into WordPress administration. Edit `/var/www/htdocs/wordpress/wp-admin/admin.php`.

In my version of WordPress the line number is 109. Yours may vary. The line that we need to comment out is this:

```
@ini_set( 'memory_limit', apply_filters( 'admin_memory_
                    limit', WP_MAX_MEMORY_LIMIT ) );
```

Now, let's modify the config files to look at Nginx and MySQL logs. By default, this file isn't writable, even by root. So we have to change that.

```
# chmod 640 /var/ossec/etc/ossec.conf
```

Now we can edit `/var/ossec/etc/ossec.conf`. Add the following lines just above the last line of the file:

```
<localfile>
   <log_format>syslog</log_format>\
   <location>/var/mysql/toby.org.org.err</location>
</localfile>
<localfile>
   <log_format>syslog</log_format>
   <location>/usr/local/nginx/logs/access.log</location>
</localfile>
<localfile>
   <log_format>syslog</log_format>
   <location>/usr/local/nginx/logs/error.log</location>\
</localfile>
```

Be sure to put the permissions back the way they were:

```
# chmod 440 /var/ossec/etc/ossec.conf
```

We can restart OSSEC with:

```
# /var/ossec/bin/ossec-control restart
```

Lastly, reload your pf.conf file:

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

Now drop from the SSH session on your server so that you're back on your laptop. The following command issued from a host that is not your server should lock you out for ten minutes. This assumes that:

- You hadn't chosen to whitelist yourself when running *install.sh*.
- You enabled Active Response.
- You have nmap installed.
- You should also get an e-mail if you enabled notifications.

```
$ nmap -T4 -A -v -PE -PS22,25,80 -PA21,23,80,3389 <your server>
```

Or, if we really have to:

```
C:\> nmap -T4 -A -v -PE -PS22,25,80 -PA21,23,80,3389 <your server>
```

TOBY RICHARDS

Toby Richards has been a network administrator since 1997. He considers himself to be a jack of all operating systems, but a true master of none. He feels this to be a mastery in its own right since he understands principles that are common to all operating systems. His articles are the product of teaching himself to become better with OpenBSD and PC-BSD. He simply writes about what he has learned most recently. For a hosting provider, he highly recommends bsdvm.com. They give you access to your VMware console so that you can re-install your OS at will, and with the settings of your own choosing.

Taking a Peek Under the Hood

Without Compromising Security

LibGTop allows developers to peek under the hood of the kernel and export lots of system data in a convenient and easy to use library.

What you will learn…

- Some LibGTop internals
- How to write simple applications (and scripts) with the library
- What went into getting LibGTop in shape on OpenBSD

What you should know…

- Basic programming knowledge
- The world is not Linux/i386

LibGTop (LibGTop manual: *http://developer.gnome.org/libgtop/stable/*) is a library used to obtain various system statistics such as CPU and memory usage. This article is a a brief introduction to the workings and usage of libgtop, as well a description of OpenBSD's libgtop port and some of the challenges involved.

What is LibGTop?

LibGTop is one of the older libraries supporting the GNOME platform. It was initially imported into the GNOME source repository as early as May 1998. To put this into perspective, libgnome was imported in November 1997. Back then LibGTop already supported several platforms: GNU/Linux, DEC OSF/1 and SunOS4. So for a change it was designed with non-Linux systems in mind. This greatly improved portability and as such it currently has backends for ten different operating systems. The FreeBSD backend was one of the first new backends to be added in August 1998, and it was the base for the generic BSD backend that was added in 2007. By this time the FreeBSD backend was infested with `ifdef` blocks for many of the other BSD's, including OpenBSD.

Thus this generic BSD backend has been used by NetBSD, BSDi and OpenBSD, and only recently a separate OpenBSD backend was created, as described later in this article.

OpenBSD has had a porti (LibGTop port: *http://openports.se/devel/libgtop2*) of LibGTop since OpenBSD 3.0 and as such packages are available for all supported architectures. This poses various challenges, but it also ensures correctness and an even greater degree of portability.

A great advantage of LibGTop is that application developers need not know on which platform the code is going to be used. This allows them to not worry about SunOS or Linux or BSD specifics and focus on what matters instead. LibGTop abstracts the platform specifics away and only exposes the developer to a well defined and stable API.

What uses it?

As part of the GNOME platform there are various applications using LibGTop. The most well known would be gnome-system-monitor and gnome-nettool. The applications use LibGTop extensively to retrieve CPU, memory, disk and filesystem usage. As well as the network interfaces, MAC addresses, network load and IP addresses. Apart from the obvious users, there are many more applications using it in less obvious ways. For example baobab from the gnome-utils package used LibGTop to retrieve disk and filesystem statistics.

Also non-GNOME projects such as gDesklets (gDesklets homepage: *http://gdesklets.de/*) use LibGTop.

And of course there are many scripts out there that use the old Python bindings provided by gnome-python-desktop. Recently it's also become possible to use the GObject Introspection data, I'll elaborate on that later in this article.

Thanks to the modular design in both the backend and frontend, applications can use LibGTop without knowing about the underlying operating system or architecture.

How does it work?

LibGTop's goal is to take information exported by the kernel to userland, on a host of different platforms and present them to the caller in a uniform and standard way. Regardless of the environment and of whether the backend for this operating system supports the feature the caller requested.

I must say that the developers of LibGTop solved this problem in a rather elegant and clean way. This allowed the library to be successfully ported to (and used on) ten different operating systems, and at least an equal number of different hardware architectures.

Various backends use different ways of retrieving the information from the kernel. For example the Linux backend uses the `/proc` filesystem intensively, even though accessing this filesystem is inefficient and slow.

The BSD backends mostly use `sysctl(3)` and `kvm(3)` to retrieve the needed information from the kernel. There are some places where specialized mechanisms are used. For example `swapctl(2)` gets used swap information, and `struct vnnode`, `struct vmspace` and `struct vm_map_entry` are used to retrieve detailed information about a process in `procmap.c` in the OpenBSD backend.

As most of you are probably aware, `sysctl(3)` is a commonly used interface to retrieve (and set) system information on BSD systems. For almost every call to LibGTop on Linux that backend has to read the correct file in `/proc`, parse it, get the needed lines from it, then do some more string parsing before having the needed value. Needless to say this is slow and error prone, and I'll leave it as an exercise to the reader to compare the Linux and BSD backends on the level of using `sysctl(3)` versus `/proc`.

`sysconf(3)` is another platform independent way to retrieve system variables, though it is only sparsely used by the AIX and Solaris backends. The OpenBSD backend only uses it to get the page size, as POSIX.1 says one should not use `getpagesize()` for this anymore.

As mentioned before, different platforms need different ways of accessing the information available in the kernel. In the general case this requires the program to be setgid kmem in order to read information such as CPU and memory information from `/dev/kmem`. Since making all the applications using LibGTop or LibGTop itself setgid `kmem` is a ridiculously insecure idea, a different approach was used. On platforms that require this, a special LibGTop *server* is being used. This program contains the system dependent code that needs special privileges and in case of BSD, it's installed setgid `kmem`.

The collected data gets stored in C structures, like `glibtop_swap` for example. The library's header files declares this structure along with its members. Such as `total`, `used`, `free`, `pagein` and `pageout` in case of `glibtop_swap`. All of the structures that contain system data, also have a special `flags` member. This is used as a bitmask which

Listing 1. *Using bitmasks the backends make their features known*

```
static const unsigned long _glibtop_sysdeps_mem =
(1L << GLIBTOP_MEM_TOTAL) + (1L << GLIBTOP_MEM_USED) +
(1L << GLIBTOP_MEM_FREE) +
(1L << GLIBTOP_MEM_SHARED) +
(1L << GLIBTOP_MEM_BUFFER) +
#if defined(__FreeBSD__) || defined(__FreeBSD_kernel__)
(1L << GLIBTOP_MEM_CACHED) +
#endif
(1L << GLIBTOP_MEM_USER) + (1L << GLIBTOP_MEM_LOCKED);
```

Listing 2. *Retrieving total amount of memory with LibGTop*

```
1  #include <unistd.h>
2  #include <glib.h>
3  #include <glibtop.h>
4  #include <glibtop/mem.h>
5
6  int main(int argc, char **argv)
7  {
8      glibtop_mem mem;
9      glibtop_init();
10     glibtop_get_mem(&mem);
11     printf("total mem (in kilobytes) = %llu\n",
12         mem.total/1024);
13     glibtop_close();
14     return 0;
15 }
```

is the way LibGTop tells callers about which fields of the structure contain correct data. In other words, using bitwise operations the backends can conditionally implement parts of the LibGTop API, called *features*. For example the generic BSD backend code contains the following piece of code: see Listing 1.

Thus `mem.cached` will only be made visible to the caller on FreeBSD, as it doesn't contain information on NetBSD (OpenBSD implemented `mem.cached` later in it's own backend). This mechanism is simple, yet quite effective.

How to use it?

Everyone knows how to use a library; learn the API, call the API in the code and link with the library and thus using LibGTop is no different. As explained earlier, its architecture is different from many libraries since it's using a server which actually retrieves data exported by the kernel and passes it to our process.

Here follows a trivial example in C to demonstrate retrieving the total memory currently available in the machine (and visible to the kernel): see Listing 2.

This program can be compiled with the following command (adding `xau` to the pkg-config command may or may not be necessary, depending on your platform):

```
cc -O2 -pipe  `pkg-config --cflags --libs libgtop-2.0 xau` \
   mem.c -o mem
```

And when run gives the following output:

```
total mem (in kilobytes) = 4067716
```

As with every other C program, first the headers need to be included, which is done on lines 1 to 4. On line 8 we declare the variable `mem` which will to contain the structure which will have the memory information for us. On line 9 we set up our connection to the privileged server, as well as obtain the features supported by this platform. Next we finally retrieve and store the memory statistics into the previously declared `mem` structure. This particular structure can have at most nine members depending on the current platform backend. Right now we're only interested in the total, which is then printed in kilobytes before closing our connection with the server.

This example works regardless of the operating system and architecture it's run on as all the backends of LibGTop implement `glibtop_mem.total`. Of course if you were to print one of the other members, say `glibtop_mem.buffer`, the results may differ between platforms due to the way memory is handled in their kernels.

Listing 3. *Mini ifconfig-like example*

```
1  #include <sys/types.h>
2  #include <sys/socket.h>
3  #include <netinet/in.h>
4  #include <arpa/inet.h>
5  #include <glibtop.h>
6  #include <glibtop/netload.h>
7
8  int main (int argc, char *argv[]) {
9     glibtop_netload netload;
10    struct in_addr addr, subnet;
11    char *address_string, *subnet_string;
12    char address6_string[INET6_ADDRSTRLEN,
13        prefix6_string[INET6_ADDRSTRLEN];
14
15    glibtop_init();
16    glibtop_get_netload(&netload, argv[1]);
17
18    addr.s_addr = netload.address;
19    subnet.s_addr = netload.subnet;
20
22    subnet_string  = g_strdup(inet_ntoa(subnet));
23
24    inet_ntop(AF_INET6, netload.address6, address6_
                string,
25        INET6_ADDRSTRLEN);
26    inet_ntop(AF_INET6, netload.prefix6,  prefix6_
                string,
27        INET6_ADDRSTRLEN);
28
29    printf("%s: flags=0x%llx mtu %d\n"
30       "\tinet6 %s%%%s scopeid %#03x\n"
31       "\tinet %s subnet %s\n"
32       "\tbytes in: %llu out: %llu\n",
33       argv[1], netload.if_flags, netload.mtu,
34       address6_string, argv[1], (int)
                netload.scope6,
35       address_string, subnet_string,
36       netload.packets_in, netload.bytes_out);
37    g_free(address_string);
38    g_free(subnet_string);
39    glibtop_close();
40    exit(0);
41 }
```

A more elaborate example is the following which will print the IPv4/IPv6 address and some more information from the specified interface: see Listing 3.

Again compile it with:

```
cc -O2 -pipe  `pkg-config --cflags --libs libgtop-2.0 xau` \
   net.c -o net
```

Running the code on my system as ./net re0 gives:

```
re0: flags=0x10846 mtu 1500
        inet6 fe80:1::e2cb:4eff:fe53:bfbb%re0 scopeid 0x1
        inet 192.168.178.89 subnet 255.255.255.0
        bytes in: 66744 out: 8017659
```

I won't hold your hand and walk you through this example, instead I would like to invite you to explore the API yourself, perhaps using the previous code as an example.

GObject Introspection

As of LibGTop version 2.28.3 GObject Introspection (GI; GObject Introspection homepage: *http://live.gnome.org/GObjectIntrospection*) support was added.

This allows programmers to use LibGTop from any language, using only the C library and the introspection data. This makes it possible to write scripts in JavaScript with Seed to gather some quick statistics, as well as writing full blown monitoring applications with Python or Java.

GObject Introspection is like the universal bindings to a library, provided there is a bridge between the introspection GIR and typelib data, and the targeted programming/scripting language. For Python this is the standard package provided by `gobject-introspection` along with `pygobject` library. Together these provide the packages to create and parse the GIR format as well as the bindings for GLib, GObject etc. These interfaces are available for many other languages, e.g. for JavaScript there is seed and for Ruby there is `ruby-gir-ffi`.

Listing 4. *The code in listing 1 ported to Python and GObject Introspection*

```
1  import gi
2  from gi.repository import GTop
3
4  mem = GTop.glibtop_mem()
5  GTop.glibtop_get_mem(mem)
6  print('total mem (in kilobytes) = %s' %
                    str(mem.total/1024))
7  Gtop.glibtop_close()
```

To give an example of Python using GObject Introspection, analogous to the first C example the following script can be used: see Listing 4.

This example needs no further explanation as its behavior is identical to the C program demonstrated earlier.

I think that one of the great advantages of GObject Introspection is that one doesn't need to learn another API to achieve something with a library one is already familiar with.

Port to OpenBSD

Port's history

The original LibGTop port was imported back in 2001 and first shipped with OpenBSD 3.0 as part of the GNOME 1.x port for OpenBSD. At this time the port was actually using the FreeBSD backend which had many OpenBSD (and NetBSD and BSDi, etc) `ifdef` blocks and as such the source was very hard to read and understand. It made the Emacs source code look pretty!

In 2003 a port of LibGTop 2.x was imported as part of the GNOME 2 platform which was still using the FreeBSD backend. OpenBSD kept using this backend until 2008 when LibGTop was released with a generic BSD backend. It wasn't until May 2011 that OpenBSD finally got its own backend implementation, but more on this shortly.

Before 2008 the port was basically only there to satisfy the dependency chain of other GNOME ports. Although one could use it to retrieve basic information, LibGTop turned out to be very unstable. Applications such as gnome-system-monitor would not work reliably for more than a minute before crashing due to LibGTop blowing up. The system information applets for the GNOME panel wouldn't work correctly, gnome-nettool was unusable. Ergo, things needed to change and LibGTop needed to get fixed.

The original LibGTop port for GNOME 1.x had in the meantime been removed (in 2007). Nobody bothered to fix the new version, so why keep the old one around if it's only going to be rotting away? No offense to the people who worked on the original port, but it was only marginally working.

So back in 2008 an update to 2.20.x was committed by Antoine Jacoutot with a clear commit message:

```
Note that it does not work better than the previous
in-tree version but it will give us a better base to fix it.
```

And fixing it we did, at least for a short while... one fix and a year and a half later I committed an update to 2.28.x:

```
it's not any less broken than the previous version, but
at least it gives us a recent base to hack on.
```

Sometime in the beginning of 2010 Antoine started working on a port of gnome-nettool and needless to say, he had to start fixing LibGTop (again). He committed about a dozen fixes and thanks to his work LibGTop became much more stable and robust. At least good enough to import gnome-nettool and a week later we imported gnome-system-monitor too. Though it was still rather unstable and wasn't displaying all the correct data, but it was a start.

Standalone OpenBSD backend

In May 2011 I decided to pickup work on LibGTop again and to finish it this time. At this point the generic BSD backend had become one horrendous piece of code that was tied together with lovely `ifdef` blocks like:

```
#if (defined(__NetBSD__) && (__NetBSD_Version__ >=
104000000)) || (defined(OpenBSD) && (OpenBSD >= 199912))
```

And that's only a harmless, non-nested block! I decided to take measures and fork the BSD backend into a separate OpenBSD implementation free of ifdef blocks and a proper base to use to fix the remaining issues. Having a standalone backend also made it much easier to submit, and eventually commit, patches upstream as it wouldn't interfere with any of the other backends. Over the course of the next few weeks many bugs were squashed and issues fixed. Varying from implementing small IPv6 tweaks to fixing crashers and correctly retrieving CPU/memory/swap/disk/network data.

Challenges

Even though the current port works great (or at least close to it..), it was far from an easy ride. Some of the biggest challenges we ran into when doing this port were (in random order):

- Type juggling: As LibGTop needs to run on various architectures with many different type widths this posed a small challenge. Of course this is no different from any other program, yet it did bite us. Some machines (like amd64) had millions of megabytes of RAM, while 32-bit machines had negative amounts of RAM, which was rather odd to see. Though we quickly diagnosed and fixed it.
- Changing API: The LibGTop API has been very stable. In fact, it hasn't changed at all since 2008 when a new function was added. The challenge here was to keep up with changes in OpenBSD. While most things are just using the simple `sysctl(3)` interface, there are pieces of code, like that in `procmap.c` that actually needed a UVM-hacker in order to fix the code when OpenBSD switched to vmmap (ariane@'s commit: *http://marc.info/?l=openbsd-cvs&m=130625098223964&w=3*). Sadly the kernel patch was backed out shortly thereafter due to loss of memory address randomization, but it will probably be committed again in time for OpenBSD 5.1.
- Unreadable source: As I just described in the previous section, at one point the generic BSD backend sources became completely unreadable and very hard to maintain and extend. Most of the code there was wrapped in various levels of `ifdef` blocks so maintenance became too hard and it was thus decided to split away from the generic BSD backend. I think this was one of the best decisions we made for this port.

Current status

I think we can say, with certain pride, that the LibGTop port has matured well. There are still some issues we need to address but generally it works very well on OpenBSD. One of the issues that exist as of writing this article is that we still depend on calling the external `lsof(8)` to get a list of open files; this needs to be migrated to `kvm(3)`. Also, we'll need to do some extensive cross-architecture testing to ensure there are no more type-casting bugs in the code and we that get correct results on all the architectures OpenBSD supports.

Conclusion

In this article I have tried to give an overview of GNOME's LibGTop project in which I've been actively involved on both sides; being an OpenBSD developer working on the port, as well as having committed to the LibGTop repository. As such I've given a brief overview of how LibGTop works and a description of the OpenBSD port.

In my opinion LibGTop is a good example of a portable project that works well in the modern desktop environment.

In the past few years there have been various lowlevel projects that claim to be portable and lightweight. Although in reality they tend to have either one big dependency (the Linux kernel) or they require massively intrusive changes to the targeted operating system kernel. Prime examples are systemd, HAL and gudev, respectively.

LibGTop solved this by having operating system independent backends which implement LibGTop's features using the operating systems' own interfaces.

Over the past two years the OpenBSD port of LibGTop has seen some major improvements. From a library that was basically only there to complete the dependency chain and wasn't doing much good; to a fully functional library that is well supported upstream too. Of course there is always room for improvement, but we're getting there and OpenBSD's upcoming 5.0 release will finally have a stable LibGTop!

I would like to thank the *gnome@FreeBSD.org* team, and Joe Marcus Clarke (*marcus@FreeBSD.org*) in particular, for their continued efforts to improve GNOME (and thus LibGTop too) on FreeBSD. Various bits of code and patches have been merged from the FreeBSD LibGTop port into the OpenBSD port.

Finally I would like to thank my fellow GNOME-maintainer in OpenBSD, Antoine Jacoutot (*ajacoutot@OpenBSD.org*) with whom I've shared several years of tough challenges, but most of all laughter and joy as a direct result from working on GNOME and OpenBSD.

JASPER LIEVISSE ADRIAANSE

Jasper Lievisse Adriaanse has been an OpenBSD developer since 2006 and a GNOME committer since June 2011. Since getting his account he has been committing minor and major contributions to basically all areas of OpenBSD, as well as portability fixes to various GNOME projects. When he's not working on OpenBSD (either professionally or as a hobby) he has a keen interest in embedded system design/programming as well as traveling and hiking.

Protecting Apache

From Dos And Ddos Attacks

DOS(Denial of Service) or DDOS(Distributed Denial of Service), it is an attack where multiple compromised systems (which are usually infected with a Trojan) are used to target a single system in attempt to make the system resources(cpu,memory,network) unavailable to its intended users and causing system to crash.

What you will learn…

- What is dos and ddos attack
- Installing and configure mod_evasive for apache2.2.x in order to protect your webserver from dos-ddos attacks

What you should know…

- Using vi or nano or pico or any text editor

In this tutorial i am introducing you an apache module that will help you protect your webserver from dos or ddos attacks.

The module i am going to use in this tutorial is called `mod_evasive`. Is a module as i said above for Apache, and its purpose is to provide evasive action in the event of an HTTP DoS or DDoS attack or brute force attack. It is also designed to be a detection tool, and can be easily configured to talk to ipchains, firewalls, routers, and etc.

Detection is performed by creating an internal dynamic hash table of IP Addresses and URIs, and denying any single IP address from any of the following:

Figure 1. *Finishing apache mod_evasive installation*

- Requesting the same page more than a few times per second
- Making more than 50 concurrent requests on the same child per second
- Making any requests while temporarily blacklisted (on a blocking list)

This method has worked well in both single-server script attacks as well as distributed attacks, but just like other evasive tools, is only as useful to the point of bandwidth and processor consumption (e.g. the amount of bandwidth and processor required to receive/process/respond to invalid requests), which is why it's a good idea to integrate this with your firewalls and routers.

Installing

```
#cd  /usr/ports/www/mod_evasive
#make install clean
#vi /usr/local/etc/apache22/httpd.conf
```

Figure 2. *Apache httpd.conf mod_evasive enable module*

Find line

```
#LoadModule evasive20_module   libexec/apache22/mod_
                    evasive20.so(figure 2)
```

And change it to

```
LoadModule evasive20_module   libexec/apache22/
mod_evasive20.so
```

Save the file and exit vi (using command `:wq`). Create `mod_evasive` config file

```
# touch /usr/local/etc/apache22/Includes/mod_evasive20.conf
#cat > /usr/local/etc/apache22/Includes/mod_evasive20.conf << EOF
<IfModule mod_evasive20.c>
DOSHashTableSize 3097
DOSPageCount 20
DOSSiteCount 100
DOSPageInterval 2
DOSSiteInterval 2
DOSBlockingPeriod 100
DOSWhitelist 127.0.0.1
DOSWhitelist 192.168.0.*
DOSLogDir „/var/log/httpd-modevasive"
DOSEmailNotify yourmail@domain.com
</IfModule>
EOF
```

Create mod_evasive log dir

```
#mkdir /var/log/httpd-modevasive
```

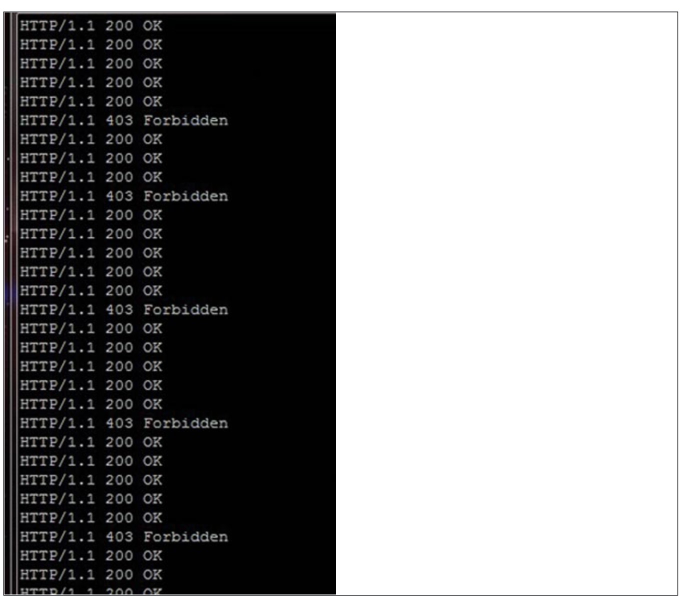


Figure 3. *Testing mod_evasive*

Make it writetable so mod_evasive can write inside this folder

```
#chmod -R 777 /var/log/httpd-modevasive
```

restart apache to activate the module

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

Note

You can modify the config of `mod_evasive` according to your needs.

Now to test it if is working create this small script

```
#touch /root/evasive_test.pl
#chmod 755 /root/evasive_test.pl
#vi /root/evasive_test.pl
```

Copy and paste the above text to the file

```
#!/usr/bin/perl
# test.pl: small script to test mod_dosevasive's effectiveness
use IO::Socket;
use strict;

for(0..100) {
  my($response);
  my($SOCKET) = new IO::Socket::INET( Proto   =>
                    „tcp",PeerAddr=> „127.0.0.1:80");
  if (! defined $SOCKET) { die $!; }
  print $SOCKET „GET / HTTP/1.0\n\n";
  $response = <$SOCKET>;
  print $response;
  close($SOCKET);
}
```

Save it and close the file. Now run the script

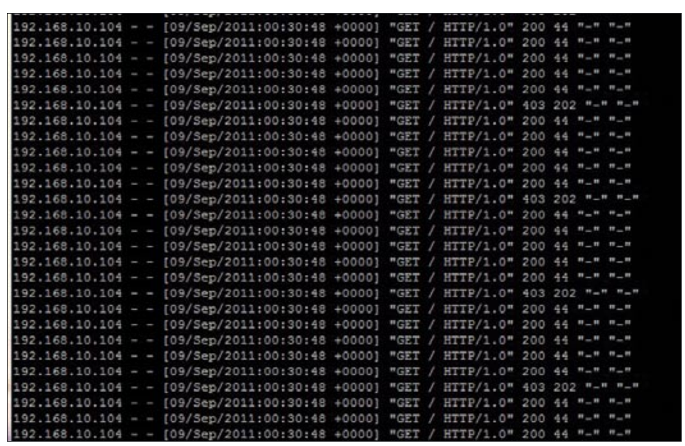


Figure 4. *Apache log showing dos attack*

```
#perl /root/evasive_test.pl
```

And if you will see figure 3 that means is running perfectly and blocking dos or ddos attacks. You will also get mail if you running mail server on the pc with the attacker ip. Now run

```
#tail -f  /var/log/httpd-access.log
```

Now in apache `httpd-access.log` you will see Figure 4 and in folder `/var/log/httpd-modevasive`

```
#ls -al /var/log/httpd-modevasive
```

you can see the blocked ips.

Now lets tune a little bit our system for ddos attacks. Edit `/etc/sysctl.conf` using `vi` or any editor and add the values

```
net.inet.tcp.msl=7500
net.inet.tcp.blackhole=2
net.inet.udp.blackhole=1
net.inet.icmp.icmplim=50
kern.ipc.somaxconn=32768
```

`net.inet.tcp.msl` defines the Maximum Segment Life. This is the maximum amount of time to wait for an ACK in reply to a SYN-ACK or FIN-ACK, in milliseconds. If the computer does not receive an ACK in this time, it considers the segment lost and frees the network connection.

This has two implications. When you are trying to close a connection, if the final ACK is lost or delayed, the socket will close more quickly. However, if a client is trying to open a connection to you and their ACK is delayed more than 7,500 ms, the connection will not form. RFC 753 defines the MSL as 120 seconds (120,000 ms). However, this was written in 1979; timing issues have changed slightly since then. Today, FreeBSD's default is 30,000 ms. This is sufficient for most conditions, but for stronger DoS protection you can lower this to 7,500 or less.

`net.inet.tcp.blackhole` defines what happens when the system receives a TCP packet on a closed port. When set to 1, SYN packets arriving on a closed port will be dropped without a RST packet being sent back. When set to 2, all packets arriving on a closed port are dropped without an RST being sent back. This saves CPU time, because packets don't need as much processing, and outbound bandwidth, by not sending out packets.

`net.inet.udp.blackhole` resembles `net.inet.tcp.blackhole` in its function. As the UDP protocol does not have states like TCP, there is only one choice when it comes to dropping UDP packets. When `net.inet.udp.blackhole` is 1, the system will drop all UDP packets that arrive on a closed port.

The name `net.inet.icmp.icmplim` is somewhat misleading. This controls the maximum number of ICMP *Unreachables* and also TCP RST packets to return every second. It helps curb the effects of attacks that generate a lot of reply packets.

`kern.ipc.somaxconn` limits the maximum number of concurrently open sockets. The default here is just 128. If an attacker can flood you with a sufficiently high number of SYN packets in a short enough period of time, he can use up all of your possible network connections, successfully denying your users access to the service.

You may find these settings to be either too aggressive or not aggressive enough. Tune them until you receive satisfactory results.

Now your server if a little more secure against dos and ddos attacks.

STAVROS N. SHAELES

Stavros N. Shaeles is a member of the IEEE and the IEEE Computer Society. He received his diploma in Electrical and Computer Engineering in Democritus University of Thrace in 2007. He is working with unix system for 8 years. Currently he is a phd student in research area of data mining with applications to computer security in *nix Systems, under the supervise of Associate Professor Alexandros S. Karakos, and also he is administrator of LPDP Lab in Democritus University of Thrace in Greece (DUTH). This article is dedicated to Stavroula Memouri (my girlfriend) for her patient and understanding to my work.


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!

The Inevitability of IPv6, Part 1

A switch from IPv4 to IPv6 is on your horizon. Are you ready for it?

What you will learn…

- IPv6 terminology and features

What you should know…

- Basic TCP/IP knowledge

IPv6 is the set of protocols that will replace today's IPv4. IPv6 offers many benefits necessary to support the Internet's continuing expansion – most notably an expanded address space that overcomes pressures in regions such as Africa, Asia, China, and the Middle East. Temporary solutions such as *Network Address Translation* (NAT) – although effective in the short term – won't provide long-term help. Recognizing that IPv6 is the future, many governments are mandating that their systems and networks support IPv6, including the US government. If your company does business with entities that use (or plan to use) IPv6, you'll feel the pressure to support IPv6, if only to support communications between your company and your partners. Simply put, IPv6 might become a competitive advantage.

In this first part of a three-part series, I describe IPv6 addressing in detail, focusing on how its addressing scheme works. I also describe some of the new features of IPv6, as well as some of the reasons you should care about it – even if you don't plan on implementing it in the near future. In two future articles, I'll describe how to configure interfaces with addresses and enable DNS resolution. I'll also describe in detail how to configure your systems and networks to use IPv6 and IPv4 together while you transition to an all-IPv6 network. Finally, I'll look into strategies for using IPv6 over the IPv4 Internet if your ISP doesn't support IPv6. But first, we need to lay down a foundation.

BSD Support for IPv6

Almost every modern OS supports IPv6 out of the box, and the BSD family of operating systems is no different. IPv6 came to BSD through the KAME project, which was a joint effort of six organizations in Japan with the aim to provide a free IPv6 and IPSec (for both IPv4 and IPv6) protocol stack. If you are a history buff like myself, you will want to Chapter 1 in *IPv6 Core Protocols Implementation* by Qing Li, Tatuya Jinmei, and Keiichi Shima.

Because of the significant internal differences between IPv4 and IPv6, some of the lower level functionality available to programmers in the IPv6 stack do not work identically with IPv4 mapped addresses. Some common IPv6 stacks do not support the IPv4-mapped address feature, either because the IPv6 and IPv4 stacks are separate implementations (e.g., Microsoft Windows 2000, XP, and Server 2003), or because of security concerns (OpenBSD). On these operating systems, it is necessary to open a separate socket for each IP protocol that is to be supported. On some systems, e.g., the Linux kernel, NetBSD, and FreeBSD, this feature is controlled by the socket option `IPV6_V6ONLY` as specified in RFC 3493.

IPv6 Addressing

IPv6 gives you a whole new means of uniquely addressing a node (or end system). In IPv6, there are 128 bits available to uniquely identify a node. IPv4 offers 32 bits, for a total of more than 4 billion possible combinations, but far fewer are practically available because of the way address space has been organized. With 128 bits, we'll have sufficient addresses for the next millennium – even given the way addresses are allocated.

Before I discuss the allocation and use of IPv6 addresses, it's helpful to understand the format that's used to represent them. Whereas IPv4 uses a dotted-decimal system (e.g., 192.168.16.10), IPv6 uses a different format. An IPv6 address is split into eight 16-bit blocks: Each block is represented by four hexadecimal digits, and each block is separated by a colon (:) – for example, `2001:0000:0000:e388:0092: fb7f:a827:fad6`. Within each block, leading zeroes can be omitted so that the address can be read as `2001:0:0:e388:92: fb7f:a827:fad6`. Also, blocks of zeroes can be omitted, so that the address can be further simplified as `2001::e388:92:fb7f:a827:fad6`. Note the use of the double colon to represent the blocks of zeroes. If you have more than one block of consecutive zeroes in an address, only one block can be omitted. (Otherwise, it would be impossible to reconstruct the original address.)

Currently, three types of IPv6 addresses can be allocated to a node: *unicast*, *multicast*, and *anycast*. A unicast address uniquely identifies a single interface (or network connection) on a node (or a virtual interface on clustered systems). A multicast address is similar to an IPv4 multicast address and can be shared by several interfaces on several nodes. A packet with a multicast destination address is delivered to all interfaces on all nodes that share the address. However, a packet with an anycast destination address is delivered to only one interface: the nearest interface to the sending interface. Regardless of type, the address identifies an interface on a node – not the node itself. A node will likely have multiple IPv6 addresses, even if it has only one interface.

Unicast Addresses

Each interface can have more than one unicast address. A unicast address can be an Aggregatable Global Unicast Address (aka global address), or a LocalUse Unicast Address.

Global address

A global address is unique to the interface it's assigned to and can be used to reach that interface from any other interface. Global IPv6 addresses are hierarchical and contain routing information. Figure 1 shows the format of a global address. A unicast address's first three bits – called the *Format Prefix* (FP) – are always 001. FPs can be of varying length (e.g., the multicast FP is eight bits in length). The next thirteen bits comprise the *TopLevel Aggregation Identifier* (TLA ID). This ID is allocated to top-level ISPs, of which there can be 8,192.

Next in the address is a reserved field – eight bits in length and designed for future expansion of the TLA ID. The next field in the address, the Next-Level Aggregation Identifier (NLA ID), is 24 bits in length and is used by the top-level ISP to organize networks or to support second-tier ISPs, each of which would have one or more NLA IDs assigned to them.

These combined 48 bits uniquely identify a site belonging to the top-level or second-tier ISP's customer. Sites are determined by geography. For example, an international company might have many sites. Each site's IPv6 connection will have a 48-bit address unique to the site. Each site can use the next sixteen bits in the address – called the *Site-Level Aggregation Identifier*

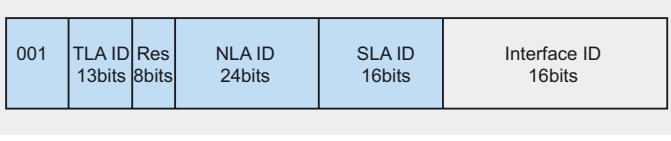


Figure 1. *Global Unicast Addressing*

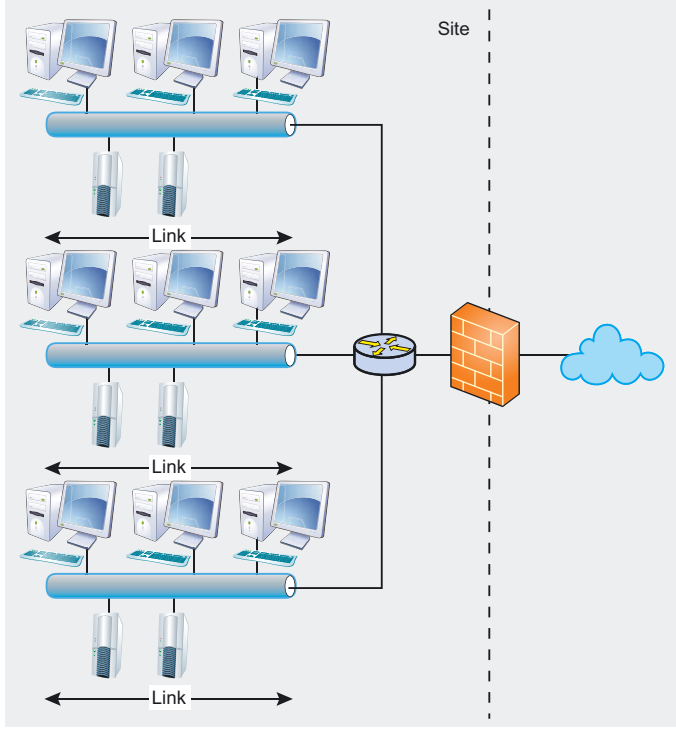


Figure 2. *IPv6 Address Scope*

(SLA ID) – to divide the site into subnets. Each site can have 65,535 subnets. Alternatively, if a company has multiple sites but only one IPv6 connection via an ISP, it can use the SLA ID to route between the sites and to the connection. The last field in the global address is the Interface ID, which is 64 bits in length. This field is similar to IPv4's host identifier, which uniquely identifies the host on the network.

Local-Use Unicast Address

There are two types of Local-Use Unicast Addresses. The first is called a *link-local address*, which is used to communicate between interfaces belonging to nodes on a single link. The second is called a *site-local address*, which is used to communicate between interfaces belonging to nodes in a site. Both are viable alternatives to a global address, depending on the scope. Figure 2 shows the scope of a link and a site.

Link-local addressing is similar to IPv4's Automatic Private IP Addressing (APIPA)[1]. Link-local addresses begin with an FP of FE80: – the last 64 bits of a link-local address are the Interface ID, and the bits in between the FP and the Interface ID are zeroed out. As with APIPA, link-local addresses are automatically configured without the need for a DHCP server or manual configuration. In fact, every IPv6 capable interface automatically has a link-local address configured for it. If you have any nodes on your network that support interfaces with IPv6, they'll have a link-local address and might be sending packets onto your network as part of Neighbor Discovery. Two nodes on the same link with interfaces that support IPv6 will automatically be able to communicate with each other, without any further configuration or management. However, communication using link-local addresses is restricted to a link – IPv6-aware routers should never forward packets with link-local source or destination addresses.

Site-local addresses are similar to the IPv4 private addresses, which have the network identifiers 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16. Site-local addresses always begin with an `FP` of `FEC0:`. As with link-local addresses, the last 64 bits of the address comprise an Interface ID. The lower 16 bits of the top 64 bits – called the Subnet ID field – uniquely identify subnets in the site, the same as the SLA ID field in a global address. The bits between the FP and the Subnet ID field are zeroed out.

IPv6 uses two special constant addresses. The first is called the *unspecified address* and is always set to `0:0:0:0:0:0:0:0`, or just `::` for short. This address – similar to the IPv4 address `0.0.0.0` – functions as a source address when no other address is available (e.g., when requesting an IP address from an IPv6-capable DHCP server). The second address is the *loopback address* and is always `0:0:0:0:0:0:0:1`, or simply `::1`. This address – equivalent to the IPv4 loopback address 127.0.0.1 – can be used for local testing of applications and configuration. Every interface will respond to the loopback address.

The Interface ID

The Interface ID in a unicast address is always 64 bits in length. It was designed this way to support 48-bit MAC addresses of current 802.x LAN technologies such as Ethernet, and wireless technologies such as Bluetooth and Wi-Fi, as well as the 64-bit addresses that FireWire uses. Future 802.x series LAN and wireless technologies will also use 64-bit addressing. The requirement to support 48-bit and 64-bit MAC addresses comes from the requirement that the Interface ID in a unicast address can be derived from a MAC address using an *Extended Unique Identifier* (EUI) 64 address. The Interface ID can also be assigned manually or by an IPv6-capable DHCP server.

In the most common scenario, the Interface ID is derived from the 48-bit MAC address of an Ethernet card. A 48-bit MAC address is split into two 24-bit halves. The IEEE assigns the first 24 bits to manufacturers. The manufacturer uses the second 24 bits to uniquely identify the card. Although it's possible to override the MAC address of an Ethernet card, let's assume that it hasn't been overridden. To convert a 48bit MAC address to a 64-bit Interface ID, the system first copies 24 bits of the MAC address to the first 24 bits of the Interface ID. Bits 17 and 16 of the first 24 bits representing the manufacturer (reading from right to left, starting at 0) are always set to 00. During the copy, the system sets them to 10. After the 24 bits are copied over, 16 bytes are added, and they're always `0xFFFE`. The system then copies 24 bits in the second half of the MAC address to produce the 64-bit Interface ID.

In dial-up scenarios, the Interface ID can be generated using a process designed to guarantee the anonymity of the user. If not for this provision, a system could be tracked as it used the Internet, regardless of the ISP used, because the Interface ID would be unique to the computer regardless of the ISP.

Multicast Addresses

IPv6 multicasting is similar to IPv4 multicasting. A node that wants to listen for multicast traffic will set the IPv6 address of an interface to the multicast address that the

traffic is being sent to. Multicast addresses have an FP of `0xFF`. The next four bits of the multicast address comprise the Flags field.

The lowest bit in the Flags field is called the Transient flag. If set to 0, the multicast address is a well-known address set by IANA; if set to 1, it's a non-permanent or transient multicast address. The next four bits of the multicast address comprise the Scope field. The purpose of this field is to identify the scope of the multicast traffic, and to identify the traffic as node-local, link-local, site-local, organization-local, or global. Routers use this field to determine whether to forward traffic. The last field in the multicast address is the Group ID, which is 112 bits in length. The Group ID identifies the multicast group. As with unicast addresses, there are predefined multicast addresses. Table 1 lists the three most common ones.

When using multicasting in IPv6, you should use only the bottom 32 bits of the Group ID field and zero out the top 80 bits. Doing so eases conversion support of the multicast address to an Ethernet multicast address. An Ethernet multicast address takes the form `33:33:xx:xx:xx:xx`. Using the recommended multicast addressing format, the bottom 32 bits of the Group ID create the Ethernet multicast address.

IPv6 also uses multicast addresses to support link address resolution. Every interface adds a multicast address for each of its unicast addresses. The multicast address takes the form `FF02::1:FFxx:xxxx`. The system copies the last 24 bits of the unicast address to the multicast address to replace the `xx:xxxx`. The system then maps the IPv6 multicast address to the MAC multicast address, as described above. This scheme reduces the number of nodes that have to process address-resolution requests. In IPv4, when one node wants to obtain another node's interface MAC address, the system sends a broadcast message to the broadcast MAC address. Therefore, every interface on the link is forced to process the request – even if it's not intended for it. In IPv6, a node that wants to find another node's

Table 1. *Common Predefined Multicast Addresses*

Multicast Address	Use
FF01::1	Node-local scope for all nodes
FF02::1	Link-local scope for all nodes
FF05::1	Site-local scope for all nodes

interface MAC address will send a broadcast message to the multicast address `FF02::1:FF:xx:xxxx`, where `xx:xxxx` is the bottom 24 bits of the interface ID. This, in turn, is translated into a MAC multicast address `33:33:FF:xx:xx:xx`. Only those interfaces on the link with matching lower 24 bits in their Interface ID need to respond to the address-resolution request.

IPv6 Features

There's more to IPv6 than simply an expanded address space. IPv6 includes a new header format, improved support for extensions and options, flow-labeling capabilities, and authentication and privacy capabilities.

New header format

IPv6's new header format minimizes the overhead often spent processing fields or information in packet headers. In IPv4, routers and end systems are required to examine packets in detail, looking for information necessary to determine whether the packet should be processed further. With IPv6, you'll now find those fields (when required) after the main packet header in Extension Headers. The new header format makes header processing much more efficient at routers, which can ignore information in any Extension Headers – with the exception of a Hop-by-Hop Extension Header, which must immediately follow the IPv6 header. The Hop-by-Hop Extension Header might contain information necessary for a router, such as a warning that a packet is a Jumbo packet (greater than 65,535 bytes), or that a router must perform additional processing on the packet.

Improved support for extensions and options

The change in the IPv6 packet header format and the use of Extension Headers facilitate this new feature. Options in Extension Headers have fewer limitations on size than in IPv4, and IPv6 is extensible by adding more defined Extension Headers over time.

In IPv6, if a destination node receives an IPv6 packet containing an Extension Header that it doesn't recognize, it informs the source node via *Internet Control Message Protocol* version 6 (ICMPv6) that it can't process the packet. This feature lets nodes implement IPv6 extensions independently of each other and still communicate.

Flow-labeling capabilities

IPv6 uses flow labeling for *Quality of Service* (QoS). Flow labeling lets a source node define a priority (e.g., real time), which might be used in *Voice over IP* (VoIP) or video-over-IP solutions to guarantee delivery of a packet within a certain time window. In IPv4, QoS often requires a router or node to look beyond a packet's header for information. In IPv6, all necessary information is in the header.

Authentication and privacy

IPv6's authentication and privacy capabilities are, essentially, IPSec. IPSec is now a requirement in IPv6 implementations, whereas in IPv4 it's an optional component. IPSec supports Authenticated Headers, which authenticate nodes to each other and ensure the integrity of data exchanged between them, and *Encapsulating Security Payload* (ESP), which has similar functionality but also includes the ability to encrypt data for confidentiality.

Unlike IPv4, in which different implementations of the protocol by different vendors could – and would – result in an inability of nodes to communicate with each other, in IPv6 interoperability is almost guaranteed, thanks to the underlying standards.

Stay Tuned

We've only just started. Now that you've got some solid foundational knowledge about IPv6, you're primed to dive into the actual configuration and use of the protocol. Get ready to make it work on FreeBSD and PC-BSD, and prepare yourself for configuring interfaces with addresses and enabling DNS resolution. In Part 2, I'll talk about how to enable IPv6 and IPv4 interoperability on your way to an all-IPv6 network.

Footnotes

Both IPv4 and IPv6 have standard methods for address autoconfiguration. For link-local addressing IPv4 uses the special block 169.254.0.0/16 as described in RFC 3927 while IPv6 hosts use the prefix `fe80::/10`. Some books and documentation refer to this as Zero Configuration networking while Microsoft refers to this as *Automatic Private IP Addressing* (APIPA). The APIPA name has stuck ever since.

PAUL T. AMMANN

Paul lives in New Fairfield, CT with his wife Eve and two cats. He recently converted from Linux to OpenBSD although he still misses his TI 99/4A and Timex Sinclair.

The Inevitability of IPv6, Part 2

Configure IPv6 in your network – even if your routing infrastructure doesn't yet support it.

What you will learn…
- IPv6 terminology and features

What you should know…
- Basic TCP/IP knowledge

As I maintained in *The Inevitability of IPv6, Part 1*, even if you have no immediate plans to migrate to IPv6 in your enterprise, you need to be ready for it, and you need to understand how FreeBSD uses it. If you communicate regularly with business partners over the Internet, you might be forced to tackle IPv6 because many companies are already beginning to make the transition. Increasingly, governments – including the U.S. government – are mandating its use.

In Part 1, I described how the BSD family of operating systems are supporting IPv6, and I provided an overview of how IPv6 addressing works. Be sure you're well-versed in that article's foundational information before taking the plunge into this article. Now, without further ado, let's investigate how to enable and configure IPv6 in FreeBSD and how to use IPv6 to communicate – even if your routing infrastructure doesn't yet support it.

Enabling IPv6 in FreeBSD

As I explained in Part 1, the BSD family of operating systems come with IPv6 installed and running. For this article, I'll be using FreeBSD 8.2 that has been updated and patched using portsnap. Let's get to it!

The FreeBSD kernel is already IPv6 enabled. You can manually enable IPv6 by adding the following line to the `/etc/rc.conf` configuration file:

```
ipv6_enable="YES"
```

You can manually start the appropriate rc script (or reboot the system) for the changes to take effect:

```
# /etc/rc.d/network_ipv6 start
```

This will enable IPv6 on all interfaces that are IPv6 capable. This behavior is changed by modifying the following variable in the `/etc/rc.conf` file:

```
ipv6_network_interfaces="em0"
```

This will enable IPv6 support on specified interfaces. The default value for this variable is auto.

Once you enable IPv6, interfaces will discover the IPv6 enabled routers on the network and build their own IPv6 addresses based on the network prefix they receive from the router.

Configuring Interfaces

In a typical scenario, IPv6 network stack will automatically look for an IPv6 enabled router on the same network for each interface and try to automatically configure the IPv6 address on the interface.

The following is an example of an automatically configured interface: Listing 1.

Beside the IPv4 address, there are two IPv6 addresses on the interface. One address begins with `fe80::` and identified with the `scopeid 0x1` tag, which is called a *link-local address.* ****

The unicast address prefix is obtained from the IPv6 router on the network. The whole address is created using the 64 bits Extended Unique Identifier (EUI-64) algorithm, which consists of the hosts MAC address with some minor modifications.

The link-local address (that is from the reserved address pool) always with `fe80::` and is used for local network usage. This can be compared with RFC 1819 private addresses that are suitable for local use. The network stack will automatically assign a link-local address to each IPv6 enabled interface, regardless whether an IPv6 router is discovered on the network. This means that in a scenario of a home network or a lab network, you don't need to run an IPv6 router or have a valid IPv6 prefix in order to establish an IPv6 network. All the hosts will be automatically provisioned with a link-local address, so they can exchange IPv6 traffic.

The *network discovery protocol* (NDP) helps the host find the router on the network and then create a unicast address for the interface. NDP is known as the equivalent to the ARP protocol in IPv6. The ndp(8) utility is used to control the behavior of this protocol: Listing 2.

The above example shows the discovered IPv6 hosts. The em0 interface is connected to an IPv6 enabled network and receives a valid prefix via a router (the first entry of the list).

The second entry is the unicast address of the `em0`. The third and fourth entries are link-local address for the router and our host.

As you have seen so far, there are some special (reserved) IPv6 addresses. The following table shows a list of reserved addresses: Table 1.

In case you want to configure the static IPv6 address on an interface, it can be done as in a typical IPv4 scenario: Listing 3.

This will manually configure an IP address on the specified interface. Note the prefixlen keyword that is equivalent to subnet mask in IPv4.

Routing IPv6

Similar to IPv4, your host doesn't automatically forward IPv6 traffic between interfaces, by default. In order to enable packet forwarding between the two IPv6 enabled interfaces, you should modify the `net.inet6.ip6.forwarding` sysctl variable:

```
# sysctl net.inet6.ip6.forwarding=1
net.inet6.ip6.forwarding: 0 -> 1
```

Listing 1. *An example of an automatically configured interface*

```
grumpy# ifconfig
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
    options=9b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM>
    ether 00:1c:42:10:c2:b2
    inet6 fe80::21c:42ff:fe10:c2b2%em0 prefixlen 64 scopeid 0x1
    inet6 ::21c:42ff:fe10:c2b2 prefixlen 64 autoconf
    inet 10.211.55.76 netmask 0xffffff00 broadcast 10.211.55.255
    nd6 options=3<PERFORMNUD,ACCEPT_RTADV>
    media: Ethernet autoselect (1000baseT <full-duplex>)
    status: active
```

Listing 2. *The ndp(8) utility*

```
$ ndp -a
Neighbor                         Linklayer Address  Netif Expire    S Flags
fe80::21c:42ff:fe00:18%em0       0:1c:42:0:0:18       em0 23h12m33s S R
::21c:42ff:fe10:c2b2             0:1c:42:10:c2:b2     em0 permanent R
fe80::21c:42ff:fe10:c2b2%em0     0:1c:42:10:c2:b2     em0 permanent R
```

Table 1. *List of reserved IPv6 addresses*

Address	Name	Description
::	Unspecified	Equivalent to 0.0.0.0 in IPv4
::1	Loopback address	Equivalent to 127.0.0.1 in IPv4
fe80::	Link-local	
fec0::	Site-local	
ff00::	Multicast	

This can also be achieved by adding the following variable to the `/etc/rc.conf` file:

```
ipv6_gateway_enable="YES"
```

After enabling IPv6 forwarding in the `/etc/rc.conf` file, you should reboot your system or run relevant rc script:

```
# /etc/rc.d/network_ipv6 restart
```

The `rtadvd(8)` daemon is another component that you may want to enable on an IPv6 router. As mentioned earlier, the hosts automatically configure the IPv6 addresses on their interface, based on the advertisements they receive from the IPv6 enabled routers on the same subnet. These advertisements are called *Router Advertisement* (RA) *packets*. The `rtadvd(8)` daemon sends router advertisements on the specified network interfaces, helping hosts to automatically configure IPv6 address on their interfaces. This is done based on the IPv6 prefix it advertises, as well as identifying itself as the gateway for the network.

To enable `rtadvd(8)`, add the following lines to `/etc/rc.conf` (ensuring that your host is also configured to forward IPv6 traffic):

```
rtadvd_enable="YES"
rtadvd_interfaces="em0"
```

Note

Make sure that you only enable transmission of RA packets on interfaces that you need to do. This can be done using the `rtadvd_interfaces` variable.

Now you should create a configuration file for the `rtadvd(8)` daemon. This file controls the behavior of the `rtadvd(8)` daemon. The `rtadvd` daemon reads `/etc/rc.conf` upon start up, to find out how it should send RA packets. A sample `rtadvd.conf` file looks like the following:

```
ef0:\
     :addr="2001:db8:ffff:1000::":prefixlen#64:tc=default:
```

This tells `rtdadvd` to advertise itself as a router for subnet `2001:db8:ffff:1000::/64`.

Please see the `rtadvd.conf(5)` man pages for more information about various options that you can use in this configuration file.

Note

It would be a good idea to use the `tcpdump` utility to see how the RA packets are being sent.

Please note that in this case your machien is configured as a *router* and not a *host*, which has a special meaning in

Listing 3. *The IPv4 scenario to configure the static IPv6 address on an interface*

```
$ ifconfig em0 inet6
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
    options=9b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM>
    inet6 fe80::21c:42ff:fe10:c2b2%em0 prefixlen 64 scopeid 0x1
    inet6 ::21c:42ff:fe10:c2b2 prefixlen 64 autoconf
```

Listing 4. *A sample tunnel setup*

```
# ifconfig gif0 create
# ifconfig gif0 tunnel x.x.x.x y.y.y.y
# ifconfig gif0 inet6
2001:470:1F03:26c::2
2001:470:1F03:26c::1 prefixlen 128

# route -n add -inet6 default 2001:470:1F03:26c::1
```

IPv6. In IPv6 terminology, a *host* is a machine that sends *Router Solicitation* messages or listen for RA packets to figure out its IPv6 address configuration as well as its gateway. On the other hand, a *router* is a machine that sends RA packets and is able to forward packets to the correct destination.

RIPv6

FreeBSD has built-in daemons that support RIPv1 and RIPv2 for IPv4 and RIPng or RIP6 (RFC 2080) for IPv6. The routing daemon that supports RIP6 is `route6d(8)`.

The `route6d(8)` daemon is almost equivalent to its IPv4 counterpart and can be enabled by setting the following variable in the `/etc/rc.conf` file:

```
ipv6_router_enable="YES"
```

Multicast Routing

The ability to route multicast traffic in FreeBSD is available using third-party software that can be used from the ports collection. The `net/mcast-tools` port allows *Protocol Independent Multicast Sparse-Mode* (PIM-SM Version 2), *PIM-Source-Specific Multicast* (SSM using PIM-SM), and *Protocol Independent Multicast Dense-Mode* (PIM-DM Version 2) routing. Once installed, the functionality is enabled by adding this line to `/etc/rc.conf`:

```
mroute6d_enable="YES"
```

This will automatically enable the `pim6dd(8)` (dense mode) daemon. If you are planning to use `pim6sd(8)` (sparse mode), you should also add the following line to `/etc/rc.conf`:

```
mroute6d_program="/usr/local/sbin/pim6sd"
```

Tunneling

There are certain cases where you want to set up a tunnel to transport IPv6 traffic over your existing IPv4 network. This can be a site-to-site VPN between two IPv6 enabled networks, or getting IPv6 connectivity to an IPv6 service provider. There are different methods by which you can set up such tunnels. The most popular methods are `gif(4)`, `faith(4)`, and `stf(4)`.

GIF Tunneling

There are chances that you don't have native IPv6 connectivity to the Internet. In that case, you can still set up a non-native (tunneled) IPv6 connection to the Internet.

There are several services that offer tunneling to IPv6 networks, such as *www.sixxs.net*. The only thing you should do is to sign up for such a service and set up a tunnel according to their instructions.

This is mostly done by encapsulating IPv6 traffic over a `gif(4)` tunnel that is established over IPv4 to the other end. In most cases, setting up such connectivity is pretty straightforward.

A sample tunnel setup would look like this: Listing 4.

In the above example, a `gif` interface is created and established between `x.x.x.x` (your IPv4 address) and `y.y.y.y` (your tunnel broker's IPv4 address). Then you should assign IPv6 addresses to the tunnel. In this case, `2001:470:1F03:26c::2` is assigned to your side of the tunnel and `2001:470:1F03:26c::1` to the other side of the tunnel. The latter is used as your IPv6 gateway as well.

The tricky part is setting up a default gateway for all IPv6 traffic to the other side of the tunnel, which is done using the `route` command (note the `-inet6` flag).

Once you have finished setting up the tunnel, you may want to test your connectivity by pinging the other side of the tunnel.

Summary

FreeBSD has had IPv6 support in the base operating system since its early versions. This support has become more mature in recent releases. Since we covered basic configuration for IPv6 in this article, you may want to do more complex things that are not covered here. There are a few useful and up-to-date resources that you can find on the Internet – one of them being the FreeBSD handbook section on IPv6 and *IPv6 Internals* in the developer's handbook.

PAUL T. AMMANN

Paul lives in New Fairfield, CT with his wife Eve and two cats. He recently converted from Linux to OpenBSD although he still misses his TI 99/4A and Timex Sinclair.

In the next issue:

- **Equip your CA with a HSM for < 50 Euros**
- **Terminals Served Up BSD Style**
- **Overview from EuroBSDcon 2011**
- **and Other !**

Next issue is coming in November!

EuroBSDcon 2011

The **Anniversary**

BSD Daemon © Marshall Kirk McKusick. Used with permission. http://www.mckusick.com/copyright.html

6 until 9 October, 2011

Meeting Plaza, **Maarssen**

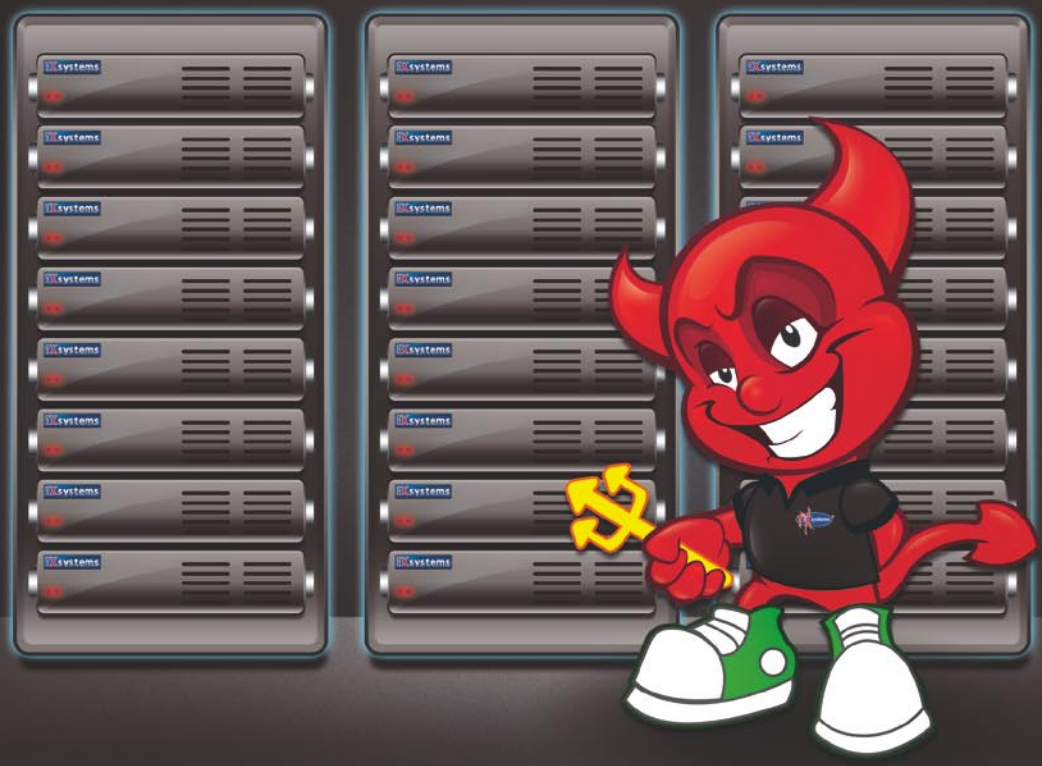
Address: **Planetenbaan** 100
3606 AK Maarssen
The Netherlands

GPS: **N**52.12840, **E**5.0360

10th European BSD Conference

http://2011.eurobsdcon.org/

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®