MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

HOSTING BSD

HOSTING ENVIRONMENT NETWORK AND FIREWALL REDUNDANCY WITH THE BSDS

FREEBSD FIREWALL WITH TRANSPARENT PROXY SERVER, DHCP SERVER AND NAME SERVER

RUNNING VIRTUALBOX OSE WITH VNC UNDER FREEBSD 8.0

MODERN FREEBSD INSTALL

X11 WITHOUT DBUS/HALD AND WITH 'THREE KINGS"

BSD FILE SHARING – PART 2. SAMBA

COMPARISON OF FREEBSD AND OPENBSD: NOT ONE CAKE BUT THE TWO ONES

EXCLUSIVELY

▶ INTRODUCING BEASTIE TO STRANGERS

VOL.3 NO.4
ISSUE 4/2010(10)
1898-9144

Energy Efficient, Powerful Performance

With dual Intel® Xeon® 5600/5500 series Six-Core, Quad-Core, or Dual-Core processors and up to 192GB of DDR3 memory, the iX-Athena is designed for small businesses seeking a high performance computing solution.

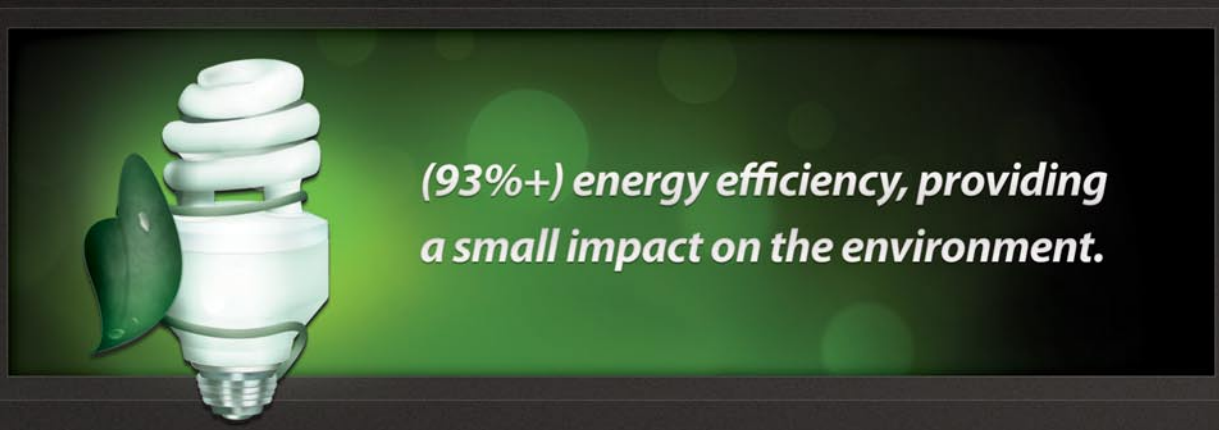
iX-Athena

Notable features include:

- Dual 64-Bit Socket 1366 Six-Core, Quad-Core, or Dual-Core, Intel® Xeon® Processor 5600/5500 Series
- Eight 3.5" Hot-swap SAS/SATA HDDs in a 4U/Tower Configuration (Optional 4U Rackmount Rail Kit Available)
- Dual Intel® 5520 Chipsets with Quick-Path Interconnect (QPI) up to 6.4 GT/s
- Up to 192GB DDR3 1333/1066/800MHz ECC Registered DIMM/24GB Unbuffered DIMM (18 DIMM Slots)
- Two (x16) PCI-E 2.0 slots, Four (x8) PCI-E 2.0 slots (1 in x 16 slot), and One (x4) PCI-E slot (in x8 slot)
- Intel® 82576 Dual-Port Gigabit Ethernet Controller
- Matrox G200eW Graphics Support
- Integrated IPMI 2.0 with Dedicated LAN
- Realtek ALC888 7.1 HD audio
- Two 5,000 RPM Hot-swap Cooling Fans
- Two 5,000 RPM Hot-swap Rear Exhaust Fans
- 1400W Redundant High Efficiency Power Supply (Gold Level 93%+ power efficiency)

Front View

Back View



iXsystems Presents the iX-Athena Workstation

The iX-Athena showcases amazing computing performance and energy efficiency, while keeping noise levels to a minimum.

The iX-Athena delivers the most powerful performance available on the market today. Dual Intel® Xeon® 5600 Series Processors (Six/Quad-Core) or Dual Intel® Xeon® 5500 Series Processors (Quad/Dual-Core) boost performance for specific workloads by increasing processor frequency. Next-generation Intel® Virtualization Technology enhances performance in virtualized environments by up to 2.1x with new hardware-assist capabilities. Up to 192GB of DDR3 memory with eighteen DIMM sockets supports higher performance for data-intensive applications and makes it easy for the iX-Athena to handle any workload.

In terms of energy efficiency, the iX-Athena also leads the pack. The automated low-power states of the Intel® Xeon® 5600/5500 series processors intelligently save power during low-use periods and increase performance when the system requires it. The iX-Athena also features an FCC Class B certified power supply with gold level (93%+) energy efficiency to provide 1400W of power and minimize impact on the environment.

The Super-Quiet operation of the iX-Athena allows users to spend less time distracted by a loud machine, and more time focusing on its powerful computing capabilities. At normal operation levels, the iX-Athena workstation's 5,000 RPM cooling and exhaust fans perform at a hushed 38 decibels to make this an ideal machine for any office or lab environment.

With eight 3.5" hot-swappable SAS/SATA hard drive bays, the iX-Athena also offers ample storage for all conceivable technical computing and graphics applications. The iX-Athena even includes four dedicated power connectors for high-end graphics cards, all contained in a stylishly sleek, high-end quality, dark gray chassis.

To order today call:
1-800-820-BSDi

For more information about the iX-Athena visit:
http://www.iXsystems.com/Athena

intel Xeon inside
Powerful. Intelligent.

Dear Readers!

Happy Easter !
I hope you had great time during this holidays!

Today we have two very important messages for you!

First one! We are opening new section in our magazine: Questions from Readers!
Feel free to send your questions concerning BSD to our team so that they could appear in the next issues of our magazine.
I hope you will like this idea and participate in it!

In second message I want to ask you to do us a favor and answer a short questionnaire concerning our magazine. This will certainly help us to improve our magazine and make it more interesting than ever before!

You can find the questionnaire in your e-mail boxes attached to your newsletter.
If you are not subscribed to our newsletter, please do this, or contact our team directly editors@bsdmag.org.

Thank you and enjoy your reading!

Olga Kartseva
Editor in Chief

MAGAZINE BSD

Editor in Chief:
Olga Kartseva
olga.kartseva@software.com.pl

Contributing:
Jan Stedehouder, Rob Somerville, Marko Milenovic, Petr Topiarz, Paul McMath, Eric Vintimilla, Matthias Pfeifer, Theodore Tereshchenko, Mikel King, Machtelt Garrels, Jesse Smith

Special thanks to:
Marko Milenovic, Worth Bishop and Mike Bybee

Art Director:
Agnieszka Marchocka

DTP:
Ireneusz Pogroszewski

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

National Sales Manager:
Ewa Łozowicka
ewa.lozowicka@software.com.pl

Marketing Director:
Ewa Łozowicka
ewa.lozowicka@software.com.pl

Executive Ad Consultant:
Karolina Lesińska
karolina.lesinska@bsdmag.org

Advertising Sales:
Olga Kartseva
olga.kartseva@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*

The editors use automatic DTP system AUPUS

Mathematical formulas created by Design Science MathType™.

get started

how-to's

let's talk

interview

Modern FreeBSD Install

Slawomir Wojtczak (vermaden)

All these years sysinstall(8) was helping us to install FreeBSD with most needed options.

Today it is not anymore up to the task with new filesystems and technologies like gjournal(8) and more important ZFS, swap and full disk encryption with `geli(8)` or RAID1/RAID0 redundancy/speed increase with `gmirror(8)` and `gstripe(8)`. Currently `sysinstall(8)` only supports installation on UFS filesystem with optional *SoftUpdates*. This article will show You how to create more modern FreeBSD installation without using `sysinstall(8)`.

This article assumes, that You would want to create fresh installation of FreeBSD, using one or three harddisks, ZFS filesystem cat be used on systems with, for example 768 MB RAM (which will require a lot of tunning in `/boot/loader.conf`), but 2 to 4 GB of RAM will be best for this king of setup. Also i386 architecture is not welcome here, since ZFS works a lot more reliably on amd64, but You may of course use i386 FreeBSD variant on system with 512 MB as well, with some heavy tweaking, but You may be facing occasional kernel panics. I would even say that i386 with small amount of RAM can be treated as testing sandbox for this kind of setup (like under VirtualBox with virtual harddisks). This setup will need these requirements:

- 64bit CPU
- 2-4 GB RAM
- 1/3 disks
- DVD/USB boot support

This install method will put / on UFS filesystem w/o *SoftUpdates* (can be later mounted read only), 2-3 GB of swap space, `/tmp` filesystem mounted on swap with mdmfs(8) and all other filesystems like `/usr` and /var mounted on ZFS pool. Mounting `/tmp` on swap makes sense cause swap is random small chunks of data often kept there for short period of time, same for `/tmp` filesystem. Many other well known UNIX systems also use that by default, like Solaris or AIX for example. It will not require rebuilding anything, just simple setup on plain MBR partitions (as opposite to

Listing 1. The layout of system with 1 harddisk

```
MBR SLICE 1 |    / | 512 MB | UFS
            | SWAP |   2 GB |
            | /tmp | 512 MB | mdmfs(8)
------------+------+------------------
MBR SLICE 2 | /usr |   REST | ZFS
            | /var |   REST | ZFS
```

Listing 2. The redundancy planning for system with 3 disks

```
[ DISK0 ]           [ DISK1 ]           [ DISK2 ]
[   /   ] < RAID1 > [   /   ] < RAID1 > [   /   ]
[ SWAP0 ]           [ SWAP1 ]           [ SWAP2 ]
[   Z   ] < RAID5 > [   F   ] < RAID5 > [   S   ]
```

Listing 3. The layout for single disk for system with 3 disks

```
MBR SLICE 1 |    / | 512 MB | UFS
------------+------+--------+---------
MBR SLICE 2 | SWAP |   1 GB |
            | /tmp | 512 MB | mdmfs(8)
------------+------+--------+---------
MBR SLICE 3 | /usr |   REST | ZFS
            | /var |   REST | ZFS
```

Listing 4. The whole procedude, described as simple as possible

1.0. I assume that disk for installation would be ad0

```
(while ad0/ad1/ad2 for system with 3 disks)
```

1.1. Boot *-dvd-* from DVD or *-memstick-* from pendrive

```
On first two screens select options as described below.
Country Selection --> United States

Fixit --> CDROM/DVD (for *-dvd-* image)
          USB       (for *-memstick-* image)
```

1.2. Create your temporary working environment

```
fixit# /mnt2/bin/csh
# setenv PATH /mnt2/rescue:/mnt2/usr/bin:/mnt2/sbin
# set filec
# set autolist
# set nobeep
```

1.3. Load needed modules

```
# kldload /mnt2/boot/kernel/geom_mbr.ko
# kldload /mnt2/boot/kernel/opensolaris.ko
# kldload /mnt2/boot/kernel/zfs.ko
```

1.4. Create/mount needed filesystems

This section is split across two versions, for system with 3 disks on the left side and for the system with dingle drive on the other.

```
DISKS: 3                                    | DISKS: 1
# cat > part << __EOF__                     | # cat > part << __EOF__
p 1 165 63  512M                            | p 1 165 63 2560M
p 2 165  * 1024M                            | p 2 159  *     *
p 3 159  *     *                            | p 3   0  0     0
p 4   0  0     0                            | p 4   0  0     0
a 1                                         | a 1
__EOF__                                     | __EOF__
                                            |
# fdisk -f part ad0                         | # fdisk -f part ad0
# fdisk -f part ad1                         |
# fdisk -f part ad2                         | # cat > label << __EOF__
                                            | # /dev/ad0s1:
# kldload /mnt2/boot/kernel/geom_mirror.ko  | 8 partitions:
# gmirror label  rootfs ad0s1               |   a: 512m  0 4.2BSD
# gmirror insert rootfs ad1s1               |   b: *     * swap
# gmirror insert rootfs ad2s1               | __EOF__
                                            |
# bsdlabel -B -w /dev/mirror/rootfs         | # bsdlabel -B -w ad0s1
                                            | # bsdlabel       ad0s1 | tail -1 >> label
# glabel label swap0 ad0s2                  | # bsdlabel -R    ad0s1 label
# glabel label swap1 ad1s2                  | # glabel label rootfs ad0s1a
```

```
# glabel label swap2 ad2s2                    | # glabel label swap   ad0s1b
                                              |
# newfs /dev/mirror/rootfsa                   | # newfs /dev/label/rootfs
# zpool create basefs raidz ad0s3 ad1s3 ad2s3 | # zpool create basefs ad0s2
# zfs create basefs/usr                       | # zfs create basefs/usr
# zfs create basefs/var                       | # zfs create basefs/var
# mkdir /NEWROOT                              | # mkdir /NEWROOT
# mount /dev/mirror/rootfsa /NEWROOT          | # mount /dev/label/rootfs /NEWROOT
# zfs set mountpoint=/NEWROOT/usr basefs/usr  | # zfs set mountpoint=/NEWROOT/usr basefs/usr
# zfs set mountpoint=/NEWROOT/var basefs/var  | # zfs set mountpoint=/NEWROOT/var basefs/var
```

1.5. Actually install needed FreeBSD sets

```
# setenv DESTDIR /NEWROOT
# cd /dist/8.0-RELEASE

# cd base
# ./install.sh (answer 'y' here)
# cd ..

# cd manpages
# ./install.sh
# cd ..

# cd kernels
# ./install.sh generic
# cd ..

# cd /NEWROOT/boot
# rm -r kernel
# mv GENERIC kernel
```

1.6. Provide basic configuration needed to boot new system

```
DISKS: 3                                              | DISKS: 1
# cat > /NEWROOT/etc/fstab << __EOF__                 | # cat > /NEWROOT/etc/fstab << __EOF__
#dev                #mount #fs  #opts #dump #pass     | #dev              #mount #fs  #opts #dump #pass
/dev/mirror/rootfsa /      ufs   rw    1     1        | /dev/label/rootfs /      ufs   rw    1     1
/dev/label/swap0    none   swap  sw    0     0        | /dev/label/swap   none   swap  sw    0     0
/dev/label/swap1    none   swap  sw    0     0        | __EOF__
/dev/label/swap2    none   swap  sw    0     0        |
__EOF__                                               |
                                                      |
# cat > /NEWROOT/boot/loader.conf << __EOF__          | # cat > /NEWROOT/boot/loader.conf << __EOF__
zfs_load="YES"                                        | zfs_load="YES"
ahci_load="YES"                                       | ahci_load="YES"
geom_mirror_load="YES"                                | __EOF__
__EOF__                                               |
... and part that is same for both ways in that section.
# cat > /NEWROOT/etc/rc.conf << __EOF__
zfs_enable="YES"
__EOF__
```

1.7. Unmount filesystems and reboot

```
# cd /
# zfs umount -a
# umount /NEWROOT
# zfs set mountpoint=/usr basefs/usr
# zfs set mountpoint=/var basefs/var
# zfs set mountpoint=none basefs
# zpool export basefs
# reboot
```

As the last command says, we will be restarting out system now and booting into newly installed one (but not yet configured), so after reboot remove installation media that You used for install process (USB/DVD).

2.0. At boot loader select boot into single user mode

```
4. Boot FreeBSD in single user mode
Enter full pathname of shell or RETURN for /bin/sh: /bin/csh
% /rescue/zpool import -D
% exit
```

2.1. Login as root without password

```
login: root
password: (just hit ENTER)
```

2.2. Set root password

```
# passwd
```

2.3. Set hostname

```
# echo hostname=\"HOSTNAME\" >> /etc/rc.conf
```

2.4. Set timezone and date/time

```
# tzsetup
# date 201001142240
```

2.5. Mount /tmp on swap

```
# cat >> /etc/rc.conf << __EOF__
tmpmfs="YES"
tmpsize="512m"
tmpmfs_flags="-m 0 -o async,noatime -S -p 1777"
__EOF__
```

2.6. Move termcap into /etc (instead of useless link on crash)

```
# rm /etc/termcap
# mv /usr/share/misc/termcap /etc
# ln -s /etc/termcap /usr/share/misc/termcap
```

2.7. Add latest security patches

```
# freebsd-update fetch
# freebsd-update install
```

2.8. [OPTIONAL] Make all changes to configuration in /etc, then set / to be mounted read-only

```
DISKS: 3                                          | DISKS: 1
 #dev                #mount #fs  #opts #dump #pass |  #dev               #mount #fs  #opts #dump #pass
+/dev/mirror/rootfsa /      ufs   ro    1     1     | +/dev/label/rootfs /      ufs   ro    1     1
-/dev/mirror/rootfsa /      ufs   rw    1     1     | -/dev/label/rootfs /      ufs   rw    1     1
 /dev/label/swap0    none   swap  sw    0     0     |  /dev/label/swap   none   swap  sw    0     0
 /dev/label/swap1    none   swap  sw    0     0     |
 /dev/label/swap2    none   swap  sw    0     0     |
```

2.9. [ONLY FOR i386] Tune the ZFS filesystem

```
# cat > /boot/loader.conf << __EOF__
vfs.zfs.prefetch_disable=0      # enable prefetch
vfs.zfs.arc_max=134217728       # 128 MB
vfs.zfs.vdev.cache.size=8388608 #   8 MB
vm.kmem_size=536870912          # 512 MB
vm.kmem_size_max=536870912      # 512 MB
__EOF__
```

2.10. Reboot and enjoy modern install of FreeBSD system

```
# shutdown -r now
```

3.1. After reboot finish installing security updates

```
# freebsd-update install
```

```
Now You have complete basic FreeBSD installation using all newest available features/technologies like ZFS filesystem,
AHCI mode that enables Native Command Queuing, small and compact / filesystem without need to fsck(8) anymore (if you
mount it read only). If you chosen to use read only /, then this little listing will make adding changes to it easier.
# mount -w /
# (...) [make changes on /]
# mount -r /
```

GPT partitions which FreeBSD also supports). It will also enable new AHCI mode for harddisks which increases performance by about 33%.

FreeBSD's *base system* consists of files spread across / and `/usr`, but with just / You have access to most important core of the base system which will be more then enought for recovery with all needed tools under `/rescue` (only in case when something wrong will happen with ZFS pool). You will need amd64/i386 `*-dvd-*` disk or `*-memstick-*` image for this installation, unfortunelly `*-disk1-*` will not do since it does not contain *livefs* system.

Here is layout of system with 1 harddisk: see Listing 1.

Redundancy planning for system with 3 disks: see Listing 2.

… and here layout for single disk for system with 3 disks: see Listing 3.

Here is the whole procedude, described as simple as possible (see Listing 4).

You can now add your users, services and packages as usual on any FreeBSD system, have fun ;)

WHB
WEBHOSTINGBUZZ.COM

800.252.1887

sales@webhostingbuzz.com

HOSTING
Without The Dark Side

Visit www.webhostingbuzz.com
to see all plans or order

Shared Hosting

FreeBSD powered shared web hosting with a free domain name and cPanel control panel. Perfect for smaller websites and prices from $4.95 per month

750GB disk space
15000GB bandwidth
1 Free Domain Name
24 x 7 phone, chat and helpdesk support

from $**7.95** per month

Virtual Private Servers

Full Control: Root access, power on/off/reboot at any time.
Choice of operating systems including FreeBSD
Dedicated Resources: Guaranteed share of CPU and RAM
Scalability: Buy only the resources you need now, and upgrade them as demand increases with virtually limitless capacity

CPU 1Ghz Guaranteed
512MB RAM Guaranteed
50GB Disk Space
FreeBSD

$**34.95** per month

Dedicated Servers

Our completely customizable range of Managed Services allows you to focus on running your business, while knowing a team of IT professionals are hard at work managing your IT environment.

- Dedicated servers to fit any budget
- Fully managed clusters
- High availability

Dual Xeon 5520
8GB RAM
2 x 1.5TB HDD
FreeBSD

$**549** per month

X11 without dbus/hald and with three kings

Slawomir Wojtczak (vermaden)

FreeBSD Handbook suggests (check section 5.4.2 Configuring X11), that running sysutils/hal (hald) and devel/dbus daemons is mandatory to have working x11/xorg ... nothing further from the truth.

X11 do not require them to run as usual, its just that FreeBSD supports two ways of handling mouse and ketboard for X11, the `hald/dbus` way and without them using good old `moused(8)` daemon. This guide will show You how to have X11 on Your FreeBSD using the second of mentioned methods. I would also add information how to disable [*CAPS LOCK*] key and bring back the working *three kings* behaviour, which means that You would be able to kill X11 with [*CTRL*] – [*ALT*] – [*BACKSPACE*] combination.

Install FreeBSD along with x11/xorg or add it by package

```
root# pkg_add -r xorg
```

Enable and start moused(8) daemon

```
root# echo moused_enable=\"YES\" >> /etc/rc.conf
root# /etc/rc.d/moused start
```

Generate new X11 config

```
root# X -configure
```

Move config to its proper place.

```
root# mv /root/xorg.conf.new /usr/local/etc/X11/xorg.conf
```

Add needed options in sections ServerFlags and InputDevice

See Listing 1.

Disabling the CAPS LOCK key

To disable it, you need to also add `ctrl:nocaps` to `XkbOptions` line, so in the end it will look like that one below.

```
Section "InputDevice"
  Identifier "keyboard0"
  Driver     "kbd"
  Option     "XkbOptions" "terminate:ctrl_alt_bksp,ctrl:
nocaps"
EndSection
```

Basic client configuration

```
user% cat > ~/.xinitrc << __EOF__
xterm &
twm
__EOF__
```

Of course `twm` is only for testing purposes, you can replace it with some more modern window manager like `openbox/fluxbox/pekwm`. If You do not prefer black console text login, then use slim light graphical login manager, it is as simple as that.

Start X11 with some custom options

```
user% xinit -- -dpi 75 -nolisten tcp
```

Example full xorg.conf config

See Listing 2.

Light and simple graphical login manager [OPTIONAL]

After You add slim with `pkg_add -r slim` it will also require a line like `ttyv8 /usr/local/bin/slim` xterm on secure in `/etc/ttys` file and `slim_enable="YES"` line in `/etc/rc.conf` file. Then You will just have to start it with `/usr/local/etc/rc.d/slim` start.

Listing 1. Add needed options in sections ServerFlags and InputDevice

```
root# vi /usr/local/etc/X11/xorg.conf

Section „ServerFlags”
  (...)
  Option „DontZap”          „off”
  Option „AllowEmptyInput” „off”
  Option „AutoAddDevices”  „off”
EndSection

Section „InputDevice”
  (...)
  Option „XkbOptions” „terminate:ctrl_alt_bksp”
EndSection

Following options are needed to have working X11 without
hald/dbus daemons.
Section „ServerFlags”
  (...)
  Option „AllowEmptyInput” „off”
  Option „AutoAddDevices”  „off”
EndSection

... and following for ‘three kings’ terminate keyboard
shrtcut.
Section „ServerFlags”
  (...)
  Option „DontZap” „off”
EndSection

Section „InputDevice”
  (...)
  Option „XkbOptions” „terminate:ctrl_alt_bksp”
EndSection
```

Listing 2. Example full xorg.conf config

```
Section „ServerFlags”
  Option „DontZap”          „off”
  Option „AllowEmptyInput” „off”
  Option „AutoAddDevices”  „off”
EndSection

Section „InputDevice”
  Identifier „keyboard0”
  Driver     „kbd”
  Option     „XkbOptions” „terminate:ctrl_alt_bksp,ctrl:
nocaps”
EndSection

Section „ServerLayout”
  Identifier  „xorg0”
  Screen 0    „screen0”   0 0
  InputDevice „mouse0”    „CorePointer”
  InputDevice „keyboard0” „CoreKeyboard”
EndSection

Section „Module”
  Load „dbe”
  Load „dri”
  Load „extmod”
  Load „glx”
EndSection

Section „InputDevice”
  Identifier „mouse0”
  Driver     „mouse”
  Option     „Protocol”     „auto”
  Option     „Device”       „/dev/sysmouse”
  Option     „ZAxisMapping” „4 5 6 7”
EndSection

Section „Monitor”
  Identifier „monitor0”
  Option     „DPMS”
EndSection

Section „Device”
  Identifier „gfx0”
  Driver     „intel”
  Option     „DPMS”
EndSection

Section „Screen”
  Identifier „screen0”
  Device     „gfx0”
  Monitor    „monitor0”
  SubSection „Display”
    Modes „1440x900”
  EndSubSection
EndSection

Section „Files”
  ModulePath „/usr/local/lib/xorg/modules”
  FontPath   „/usr/local/lib/X11/fonts/misc/”
  FontPath   „/usr/local/lib/X11/fonts/TTF/”
  FontPath   „/usr/local/lib/X11/fonts/OTF”
  FontPath   „/usr/local/lib/X11/fonts/Type1/”
  FontPath   „/usr/local/lib/X11/fonts/100dpi/”
  FontPath   „/usr/local/lib/X11/fonts/75dpi/”
EndSection
```

Converting a FreeBSD Port Using PBI Builder

Dru Lavigne

This is an excerpt from the "Becoming a Developer" chapter of the recently released book, The Definitive Guide to PC-BSD.

The Definitive Guide is meant to be so, taking the reader from complete PC-BSD novice to advanced, PC-BSD power user. This means that some of the concepts used in later chapters are covered in detail in earlier chapters. The book is available with a companion DVD of PC-BSD 8.0 from the FreeBSD Mall.

Chapters 9 and 10 introduced you to FreeBSD ports and packages and gave some insight into the work port maintainers go through so that the package and port "just work." PBI Builder simplifies the process of converting an existing FreeBSD port into a PBI, which means anyone can create a PBI without needing much (if any) previous development experience. If you have a bit of time to spare, like to learn new things, and are interested in seeing as much software as possible available to the PC-BSD community, try your hand at creating a PBI with PBI Builder. The more PBIs that are available, the better it is for everyone because it ensures that even brand new PC-BSD users can safely install and keep up to date about the software that they need.

Information about and the download for PBI Builder can be found at *http://www.pcbsd.org/ content/view/45/30/*. PBI Builder is a command-line utility that requires you to edit a few configuration variables that are used when the PBI is built. PBI Builder automates the entire build process: the creation of the build sandbox, fetching the source for the port and all its dependencies, building everything that is needed, and converting the results into the PBI.

PBI Builder uses a large archive that contains the system source and world environment used by PC-BSD. It provides all the libraries needed to ensure that the resulting PBI works on the version of PC-BSD that matches the version of PBI Builder.

Tip

The file */pbi-build/docs/HOWTO-MODULES* is well worth reading because it fully explains all the files contained in the archive and the PBI creation process. If you're curious about what commands are executed when building a PBI, read through the scripts in */pbi-build/scripts/*. You can also find some examples in */pbi-build/docs/module-examples*.

Building Your First PBI

Before building your PBI:

- Check that a PBI for that software doesn't currently exist at *pbidir.com* or *pbibuild.pcbsd.com*.
- Check that the Pbi-dev mailing list isn't currently testing a PBI for that software.
- Check to see if a module already exists at *http://trac.pcbsd.org/browser/pbibuild/modules*
- Search for the software at freshports.org. Some of the FreshPorts details for that software come in handy when you configure your PBI module.
- Download and untar the PBI Builder archive according to the instructions in the *Using the PBI Builder* (*http://wiki.pcbsd.org/index.php/Using_the_PBI_Builder*) document.

Now that your system is ready for building PBIs, download the PBI module template.

```
# cd /pbi-build/modules
 # fetch http://www.pcbsd.org/files/templates/module-
template.tgz
 # tar xzvf module-template.tgz
```

Create a directory structure for your module that represents the port's category and name. Copy the contents of the template directory to your new directory. We use the example of creating a module named */pbi-build/modules/irc/conspire*.

```
# mkdir -p irc/conspire
# cp -R template/* irc/conspire/
# ls -F irc/conspire
build.sh    kmenu-dir/    overlay-dir/
preportmake.sh
copy-files    mime-dir/    pbi.conf
```

Most PBIs can be successfully built after modifying a few lines in *pbi.conf* and kmenu-*dir/0mymenu*. This section shows you how to make those changes, and the next section demonstrates more advanced configurations.

To successfully configure your module, you must modify the following variables in the pbi.conf file.

- *PROGNAME*: The name of the PBI. This should be the same name as the FreeBSD port. Don't include the version number unless there is already another PBI for a different version.
- *PROGWEB*: The Main Web Site URL for the port as listed at Freshports.
- *PROGAUTHOR*: Most software is maintained by a project rather than an individual. Examples of suitable values are The Mozilla Foundation (for Firefox) or the BitchX team (for bitchx).
- *PROGICON*: Check the pkg-plist in the CVSWeb for the port to find the path to the png file representing the icon for the application. If there is more than one, look for the png with the same name as the port. If there is no png for the software, check the software's website to see if it has an icon image. If there is an image available, download the image, convert it to png if it is in another format, save the png to the module's overlay-dir directory, and provide only the name of the png.
- *PBIPORT*: The full path to the port to be built.

Here is an example of the changes made to */pbi-build/modules/irc/conspire/pbi.conf*: see Listing 1.

Next, you must modify the first three variables in `kmenu-dir/0mymenu`.

- *ExePath*: The path to the executable that should start when the application is launched. You can find the correct path name at Freshports. Click the CVSWeb link for the port, and then click the pkg-plist. The binary has *bin* somewhere in the path. If there are multiple binary paths, select the binary that seems the most reasonable name for the application.
- *ExeIcon*: The same path you used in `PROGICON= in pbi.conf`. This allows the icon to show in the KDE menu.
- *ExeDescr*: A short (2 – 3 words) description that shows up in the KDE menu.

The example for `/pbi-build/modules/irc/conspire/kmenu-dir/0mymenu` looks like this:

```
ExePath: bin/conspire
ExeIcon: share/pixmaps/conspire.png
ExeDescr: IRC Client
```

When you finish making your changes, ensure that the system is connected to the Internet because you require connectivity to build the underlying port.

You're now ready to cd into the `/pbi-build` directory and start the pbibuld.sh script. Include the name of the module you wish to build. If you don't provide any arguments, the script builds every module that exists in the modules directory. The script provides some messages as the build progresses, as seen in the following example: see Listing 2.

If you want to watch the details of the build process, you can monitor the build log using tail `-f /pbi-build/outgoing/irc/conspire/build.log` and substitute the pathname for your PBI.

Although the PBI build process is completely automated and should just work, it does take time. The amount of time

Listing 1. An example of the changes made to /pbi-build/modules/irc/conspire/pbi.conf

```
#Program Name
PROGNAME="conspire"

#Program Website
PROGWEB="http://confluence.atheme.org/display/CON/Home"

#Program Author
PROGAUTHOR="Conspire Team"

#Default Icon: (Relative to overlay-dir)
#Please only use PNG files for the program icon PROGICON="share/pixmaps/
conspire.png"

#FreeBSD Port we want to build
PBIPORT="/usr/ports/irc/conspire"
```

Listing 2. The script messages as the build progresses

```
# cd /pbi-build
# ./pbibuild.sh irc/conspire
Running portsnap to update ports tree
Starting module traversal...
Copying /pbi-build/buildworld to /pbi-build/pbisandbox
Copying /pbi-build/ports to /pbi-build/pbisandbox/usr/ports
Starting build of irc/conspire
Rebuilding module irc/conspire...
Found preportmake.sh, running it...
Running port build...
SUCCESS! Build finished for irc/conspire
#
```

depends upon the size of the application, the number of dependencies, and the speed of your build system.

When the build is finished, you receive your prompt back and the PBI is placed in a subdirectory of `/pbi-build/outgoing/` with the same name as the module you built. In this example, the PBI is found in: `/pbi-build/outgoing/irc/conspire/conspire4.0.35-PV0.pbi`.

Advanced Module Configuration

Most PBIs can be built by simply modifying the variables mentioned in the previous section. This section provides an overview of the more advanced configurations that are possible through modifying the other variables and files that come with the modules template.

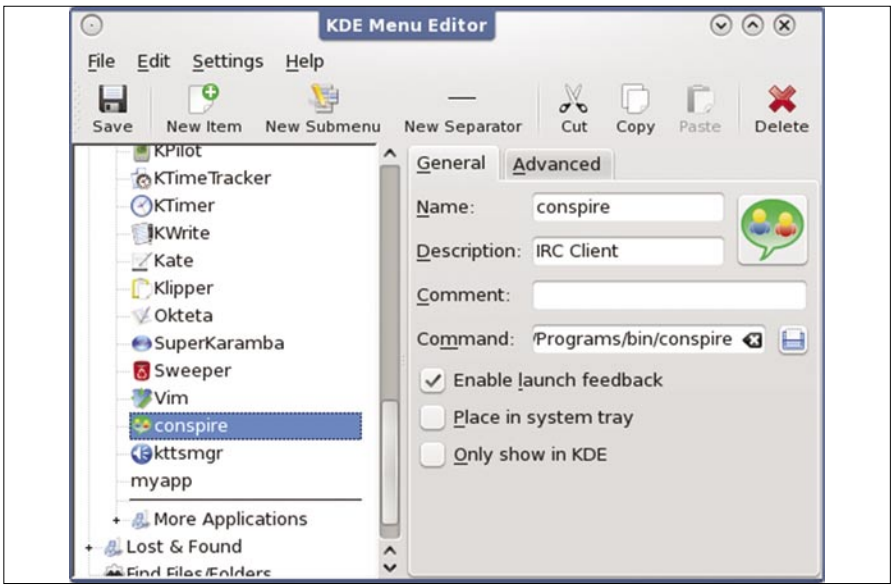


Figure 14-4. KDE menu settings for the conspire PBI

Table 14-2. Variables that Control a PBI's Appearance in the KDE Menu

Variable	Description
ExePath	The path to the application's executable as listed at Freshports.
ExeIcon	The path to the application's icon as listed at Freshports or the name of the custom icon you have created in overlay-dir/.
ExeDescr	A brief description of the application.
ExeNoDesktop	Set to 0 if you want a desktop icon and to 1 if you don't.
ExeNoMenu	Set to 0 if you want an icon in the KDE menu and 1 if you don't.
ExeRunRoot	Some applications require superuser access to run correctly. Set this to 1 to require
	the user to enter the administrative password when the application launches.
ExeRunShell	Set to 0 if the application should run in a GUI and set to 1 if the application is.
	command-line based and should be executed in a Konsole session.
ExeNotify	Set to 0 to disable the bouncy application loading icon and set to 1 to enable it (the preferred setting).
ExeLink	Set to 1 to open the ExePath value in Konqueror, and set to 0 to launch the ExePath value as an executable.
ExeWebLink	If the ExePath value is an URL, set to 1 to open the URL in Konqueror; otherwise, leave it set as 0.
ExeTaskbar	Places application in system tray; this feature is currently unimplemented.
ExeOwndir	0 places the application name in top level directory of KDE menu, 1 places the application name in its own directory under the category indicated by ExeKdeCat, and 2 places the application name in the category indicated by ExeKdeCat.
ExeKdeCat	Set to one of the category names listed in *Kickoff->Applications*.

build.sh

This script is run after all the files have been copied to the PBI's directory and can contain any commands you wish to run at that time. The PBI Module Builder Guide (*http://wiki.pcbsd.org/index.php/PBI_Module_Builder_Guide*) provides an example that modifies the version number.

copy-files

It is rare to need this file, but you can use it to modify where certain files get populated to.

kmenu-dir/0mymenu

The variables in this file control how the application appears in the KDE menu. Table 14-2 provides a description of each variable.

mime-dir/00mymime

Some applications require their MIME types to be listed to work correctly. See the PBI Module Builder Guide for usage examples and gotchas.

overlay-dir/

PBI builder automatically populates all the files needed by the PBI, according to the underlying port's instructions. If you have an additional file you would like to include (for example a README for the PBI) or a customized graphic, include it here. You can also customize the PBI scripts that came with this directory but should only do so if you have a good reason to make the change.

pbi.conf

Table 14-3 summarizes the remaining variables in this file.

preportmake.sh

Allows you to execute commands needed for the port to build properly. See the PBI Module Builder Guide for an example.

If you right-click Kickoff and select Menu Editor, you can see the settings that come with every application in the KDE menu. Comparing the General and Advanced tab of an application should

give you a better understanding of the effect that the variables in Table 14-3 have on the KDE menu. Figure 14-4 shows a screenshot for the installed conspire PBI.

Table 14-3 briefly describes the remaining variables that can be set in `pbi.conf`.

Troubleshooting

As long as there isn't a problem with the underlying FreeBSD port and assuming you have followed all of the steps in the section on *Building your First PBI*, PBI Builder should just work. If the build fails, double-check Freshports to confirm that the port isn't broken, forbidden, or restricted.

If the port looks fine, check the error message that appeared when you received your prompt back. It contains the number of the script that failed. The 2.1 in the following example indicates that `/pbi-build/scripts/2.1.startmake.sh` failed. Any script with a lower number is successful, and any script with a higher number has not run yet.

ERROR: 2.1 Build failed of irc/conspire!!!

When PBI Builder exits, it compresses the log of the PBI build process, so you need to uncompress it with the `bunzip2` command. Take the time to go through the build log, starting at the end, because this is where the error occurred. Usually, the problem is obvious from the error. If it is not, work your way backwards to see what happened successfully before the error occurred. If the error indicates that the port build was unable to fetch a required file, double-check your Internet connectivity.

After you resolve the error, remove the *.lock* file and rerun `buildpbi.sh`. The build starts over again to ensure that your sandbox environment is clean.

If you are stuck, send an email to the Pbi-dev mailing list that includes the error and enough contextual information to enable other developers to help you figure out what went wrong.

Testing and Submitting Your PBI

After you have a PBI, you want to test it yourself before making it available for others to test. From Dolphin, navigate to your PBI, right-click it, and select Open with PBI Launcher. The PNG for your PBI should show in the PBI's icon within Dolphin. As the PBI installs, check the initial installation screen to ensure that the Vendor (PROGAUTHOR variable) and URL (PROGURL variable) are displayed correctly. After the PBI is finished installing, start the application to make sure that the correct binary starts. After the application launches, try out all the screens in the program to make sure that nothing is missing and none of the menus causes the application to crash. Finally, find your PBI in Menu Editor, and make sure that all the desired features show for the KDE menu.

If you find a typo or need to fix a configuration file in your module, you don't have to rebuild the underlying port. Simply run `/pbi-build/scripts/3.makepbi.sh` after making your configuration change. This rerolls the PBI so you can test your changes.

Tip pbibuild.sh creates a clean environment every time it runs. This means that it removes everything that was previously built and starts over again. If your build successfully finished, you don't have to rebuild to reroll the PBI with your new configurations. You can save a lot of time by running the `3.makepbi.sh` script.

When you are satisfied that your PBI works correctly, create a compressed archive of its directory. The following example creates a compressed archive named `/conspire.tar.bz2`.

```
# cd /pbi-build/modules/irc
# tar cvf /conspre.tar
# bzip2 /conspire.tar
```

Upload the archive to a publicly available server. If you don't have a server of your own, contact the leader of the PBI development team for credentials to the PBI ftp server. After the PBI is uploaded, send an email to the Pbi-dev mailing list. Your email should include a subject line of *submit module category/portname* (for example, submit module irc/conspire). The body should contain the location where testers can download the archive for the module to build and test it.

Table 14-3. Remaining Variables for pbi.conf

Variable	Description
PBIVERSION=	Enables you to override the PBI version if the build fails to automatically detect it.
PROGLIBS=	Leave at AUTO; otherwise, you have to manually populate the PBI's directory. If you need to override a file that is populated, use copy-files instead.
PBIUPDATE=	Leave as-is as needed by the PBI build server.
OTHERPORT=	If you want to include another port in your PBI (besides the dependencies listed in the port's Makefile), add its category and portname; this is useful for applications that have additional plugins or skins that are available as separate ports.
MAKEOPTS=	Enables you to pass make targets that are used when the PBI is built; Chapter 10 discusses targets.
BUILDKEY=	Committers can temporarily change this number to force the build server to rebuild the PBI.
PBIDISABLE FONTLINK=	Use this if you want to use the application's internal fonts instead of the system fonts.
PBIKEEPGL=	Use this to use the applications internal libGL libraries instead of the system libraries.
PBIPRUNE*	Several prune variables allow you to keep include directories, python files, perl
	files, or doc files that were created during the PBI build.
BUILDINMATE=	Uncomment this line if you are building an inmate file instead of a PBI.
INMATEVER=	Uncomment and set a version number for the inmate; increment the number for each later version.

BSD File Sharing – Part 2. SAMBA

Petr Topiarz

Last time I wrote about NFS on different BSD's. This time I am going to dedicate this article of the series to SAMBA.

Why SAMBA? Well, while samba is far from being a reliable well secured tool for sharing, it definitely is very usable in terms of sharing files with various versions of MS Windows.

As samba is supported by all unices I have come across, I find it rather important for any network administrator to be able to configure it and make it work. This can be very beneficial in situations such as when a person visits your office or home and now has the ability to connect to your network shares regardless of the operating system they are running on their laptop. Samba is also a very easy way to share printers.

First I am going to describe the way in which we setup a simple samba server with various BSD systems, then I will try to give you an account of some of the other samba features that can be used to extend its usability and finally I will show how to access a samba share.

Starting samba on BSD

With all the BSD's variants, samba comes as a third-party package in `ports/pkgsrc`. The currently used version is version 3 or 3.3 depending on your BSD system. After installing the package you need to configure your system to start Samba automatically on boot-up as well as configure your shares and determine who may access them.

NetBSD

Install samba:

```
# cd /usr/pkgsrc/net/samba
# make install clean
```

Edit `/etc/inetd.conf` and uncomment the next two lines:

```
netbios-ssn stream tcp nowait root /usr/pkg/sbin/smbd
netbios-ns dgram udp wait root /usr/pkg/sbin/nmbd
```

add the following lines to `/etc/rc.conf`:

```
smbd=YES
nmbd=YES
samba=YES
```

copy starting scripts to its place:

```
# cp /usr/pkg/share/examples/rc.d/samba /etc/rc.d/
#cp /usr/pkg/share/examples/rc.d/smbd /etc/rc.d
#cp /usr/pkg/share/examples/rc.d/nmbd /etc/rc.d/
```

reboot and samba is up and running.

You can restart samba daemons any time:

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

OpenBSD

Install samba:

```
# cd /usr/ports/net/samba
# make install clean
```

Edit `/etc/rc.local` and add these lines:

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

```
        echo -n ' nmbd'
        /usr/local/libexec/nmbd
fi
```

reboot and samba is up and running.

Listing 1. Configuring SAMBA

```
$ cat /etc/samba/smb.conf
[global]
  workgroup = BSD
  server string = clipper
  smb passwd file = /etc/smbpasswd
  encrypt passwords = yes
  load printers = yes
  printing = cups
  printcap name = cups
  show add printer wizard = Yes
  use client driver = yes

[HP-Laser]
  comment =  HP LaserJet 2300L
  path = /var/spool/samba/printing
  printer = HP-Laser
  public = yes
  writable = no
  printable = yes

[HP-PSC]
  comment =  HP PSC 1510
  path = /var/spool/samba/printing
  printer = HP-PSC
  public = yes
  writable = no
  printable = yes

[print$]
    comment = Printer Drivers
    path = /etc/samba/drivers
    browseable = no
    guest ok = no
    read only = yes
    write list = root

[shared]
  comment = shared space
  browseable = yes
  path = /home/samba
  public = yes
  readonly = no
  writable = yes
  create mask = 0777
```

You can manually restart samba any time:

```
kill -HUP `cat /var/run/smbd.pid`
kill -HUP `cat /var/run/nmbd.pid`
```

FreeBSD

Install samba:

```
# cd /usr/ports/net/samba3
# make install
```

Add the following lines to the `/etc/rc.conf` file:

```
nmbd_enable="YES"
smbd_enable="YES"
```

reboot and samba is up and running.

You can restart samba daemons any time:

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

Configuring samba

With NFS, the file `/etc/exports` is used for share configuration, similarly the Samba equivalent is a file called `smb.conf`. The file `smb.conf` is often stored in `/etc/samba` or simply in `/etc`. I use the same `smb.conf` file on various Linux distros as well as on OpenBSD and NetBSD. After checking FreeBSD man page I found out that the same `smb.conf` file would work with FreeBSD as well. This one is very simple, and it's main feature is that it makes the shares easily discoverable and usable to anyone (to an attacker as well, of course). It is suitable for non-important public shares on well secured nets see (Listing 1).

The above file will work well on most systems. Now we will look at the file structure. The first paragraph with the *global* options tells us very basically what we want the group and server to be called and how it appears when viewed via samba browsers. Next it specifies where the system should look for samba user passwords. This is a very important option, as it can cause a lot of trouble if incorrectly changed by you or an application. Next there is information about using cups for printing. The following paragraphs define the printers we use at our company. One of the most important variables, is the path to the spooler, that can vary on different systems. The section called `[print$]` defines a directory where you can place window drivers that can be loaded should a connecting system be

missing a specific printer driver for one of your printers. Finally there is the `[shared]` section that tells us the (path) where the shared files are and that they are fully accessible to anyone.

Configuring Samba can probably be easier when using an administration tool such as SWAT. However I never use it. I prefer editing config files on my own. If you are one of the people who prefer using graphical configuration tools, have a look at the following url: *http://samba.org/samba/docs/man/Samba-HOWTO-Collection/SWAT.html*

Security

Now I believe many people would like to make their shares a little more secure, or at least not so easily accessible by anyone who plugs in a cable. There are security features in samba that can assist in securing your shares, for example setting a variable `security = user` will make the share accessible by specific samba share users only. Setting up a samba share password is done easily. Here is an example of adding a system user *david* to the samba password list:

```
# smbpasswd -a david
```

the system will then prompt you for a password, accept it and save it in the file defined in the global options section. After choosing the `security = user` feature you have to add a line to your samba shares, similar to this:

```
valid users = david liz
```

The line above will allow users `david` and `liz` to access the chosen samba share. Of course there can be various shares defined in the smb.conf with different security options. There are more security options and features available in the Samba configuration. You can decide whether read only or read-write access is given to the `user` or `public`, you can pick a domain, a group of users to access the share and so on. However there is one interesting fact to realize:

SAMBA IS VERY INSECURE – and to prove it, hackers offer a tool for everyone to try out. It is called *smbsniff*, and its description is as follows: Smbsniff is a LanManager packet sniffer that will write to your disk all the files shared and the documents printed in a LanManager (understand *samba*) environment.

Accessing samba

Windows machines will find your shares via the network environment icon on their desktops. Unix desktop environments such as Gnome and KDE have integrated samba clients in their file browsers. The same is true for Mac. Samba shares can also be accessed and mounted in a command line environment. The command line samba client uses smbfs to access the share. The commands are rather easy:

To see which shares are available on a given host, run:

```
# smbclient -L host
```

You can also browse the content from a windows machine with NetBSD smbclient:

```
# smbclient //host/shared_name_resource
```

Then you can also mount the chosen share on your machine as if it was a part of your local filesystem:

```
# mount_smbfs -W workgroup -u david
//host/share /mnt/samba
```

The above example will try to mount a remote share to your local directory `/mnt/samba`. While accessing the remote file system it will give the user name of david and workgroup work group.

Of course the *host* in the examples above is a name or IP address of a machine in your network.

Summary

Samba is a way to share your files and printers with MS Windows, Mac, Linux or other BSD's. Samba is a universal and very practical tool for everyday file sharing and printer sharing. However samba is insecure and it is not recommended for production, important, or confidential shares especially in a large network environments.

Sources

- *http://wiki.netbsd.se/How_to_set_up_a_Samba_Server*
- *http://www.freebsd.org/doc/handbook/network-samba.html*
- *https://calomel.org/samba.html*
- *http://www.samba.org/*
- *http://www.hsc.fr/ressources/outils/smbsniff/*
- *http://www.openunix.eu/*

And a lot of practice … :-)

Running VirtualBox OSE with VNC under FreeBSD 8.0

Rob Somerville

VirtualBox is a type 2 hypervisor that sits directly on top of the host-server OS and is suitable for server, desktop and embedded applications. It will run most OS's as guest with few exceptions, and like Vmware * there are many pre-built VM's available.

While VirtualBox is generally very stable, there are a few *gotcha's* that are specific to certain versions and hardware configurations which are covered later in this article. A VirtualBox enterprise support license is available from Oracle* for a number of platforms, but at the time of writing there was no specific offering for the BSD community, so we will be using the VirtualBox Open Source Edition for this installation. There seems to be little functional difference between the two products other than support for VNC and USB in the enterprise version.

Installation requirements

While VirtualBox will itself require little in way of disk space (and will run in 512Mb of RAM) depending on the size of your VM images, and the types of VM you intend to roll out will play a large part in dictating what hardware you will require. If more than 4GB of RAM will be required, the 64 bit version of FreeBSD would be preferred rather than the i386 version FreeBSD 8.0 which was used for this setup. All tests were carried out on a 2.6 GHZ dual core AMD 64 with 2GB of RAM and 80GB of storage.

In this example once FreeBSD was installed the server was run headless and all updates etc. were carried out via SSH. A working internet connection will be required for patching/downloading packages etc. For this installation, we will be using BlackBox as the Window Manager and Tightvnc for accessing the desktop, but due to inherent security issues with VNC based applications, it is recommended that the Tightvnc traffic is run through a secure tunnel in a production environment.

Packages or Ports?

Depending on how *bleeding edge* you want your installation to be, and if you have the time available, VirtualBox should install OK from ports, but there have been problems in the past that are documented at *http://wiki.freebsd.org/VirtualBox*. For this install I used packages and the server was up and running with multiple VM's in under 90 minutes using some pre-prepared ISO's.

Part 1 – Commissioning VirtualBox

Perform a clean basic FreeBSD install, with no ports or packages. You can use DHCP as the IP address for the server if preferred. Enable SSH, and create a guest account with membership of the wheel group. Ensure Internet connectivity is present. During testing, I used a script file to load the drivers when needed, but if you prefer they can be added to `loader.conf` – see *Improving this configuration at the end of this article*.

If running headless, login with the guest account and su to root. Patch the box:

```
freebsd-update fetch
freebsd-update install
```

Install the required packages and check for vulnerabilities. If you want to run VirtualBox headless, install TIGHTVNC:

```
pkg_add -r xorg blackbox virtualbox
pkg_add -r tightvnc
pkg_add -r portaudit
/usr/local/sbin/portaudit -Fda
```

Modify RC.CONF to support Xorg and VirtualBox.

```
# Added for VirtualBox support
```

```
dbus_enable="YES"
hald_enable="YES"
```

Add PROCFS support for VirtualBox in `/etc/fstab`

```
# Added for VirtualBox support

proc   /proc    procfs   rw       0
0
```

Create a test VBOXload script in `/usr/local/sbin`: see Listing 1.

To allow an unprivileged user to mount CDROM in VirtualBox add the following to `/etc/devfs.conf`:

```
# Added for VirtualBox support
perm  cd0      0666
perm  xpt0     0666
perm  pass0    0666
```

If you intend to run VirtualBox at the server console edit `/home/vboxuser/.xinitrc`:

```
exec blackbox
```

If you want to run headless, now is the time to SSH into the box with your guest account. Add a custom user vboxuser and ensure they join the `vboxusers` group

```
su
adduser
```

Next we need to configure vncserver for the `vboxuser` account:

```
su vboxuser
vncserver
```

When prompted enter your password and say n to view only password.

Edit `/home/vboxuser/.vnc/xstartup` script to support BlackBox:

```
#!/bin/sh
xrdb $HOME/.Xresources
blackbox &
```

Restart vncserver on the host to pull in the changes:

```
vncserver -kill :1
vncserver
```

Create a directory for ISO images:

```
mkdir /home/vboxuser/.VirtualBox/ISO
```

Either copy some ISO's across to the newly created directory or roll your own from an OS CD/DVD. Insert the OS of your choice into the CDROM drive then fall back to your root account:

```
exit
cd /home/vboxuser/.VirtualBox/ISO
dd if=/dev/acd0 of=image.iso bs=2048
chown vboxuser:vboxuser image.iso
```

Listing 1. Creating a test VBOXload script

```
#!/bin/sh

# NOTE: Under certain circumstances VirtualBox KM's can cause the server
# to panic.
#
# KM's should be OK with later releases, but until stable & tested I
# prefer to manually load the modules on new machines.
# See http://wiki.freebsd.org/VirtualBox
#
# You cannot create a VM from CDROM media over a VNC session using this
# script - load the drivers via loader.conf if you want to do this.

echo Loading VirtualBox kernel module support ...
kldload atapicam
kldload vboxdrv
kldload vboxnetflt
kldload vboxnetadp

Make it executable:
 chmod 550 /usr/local/sbin/VBOXload
```

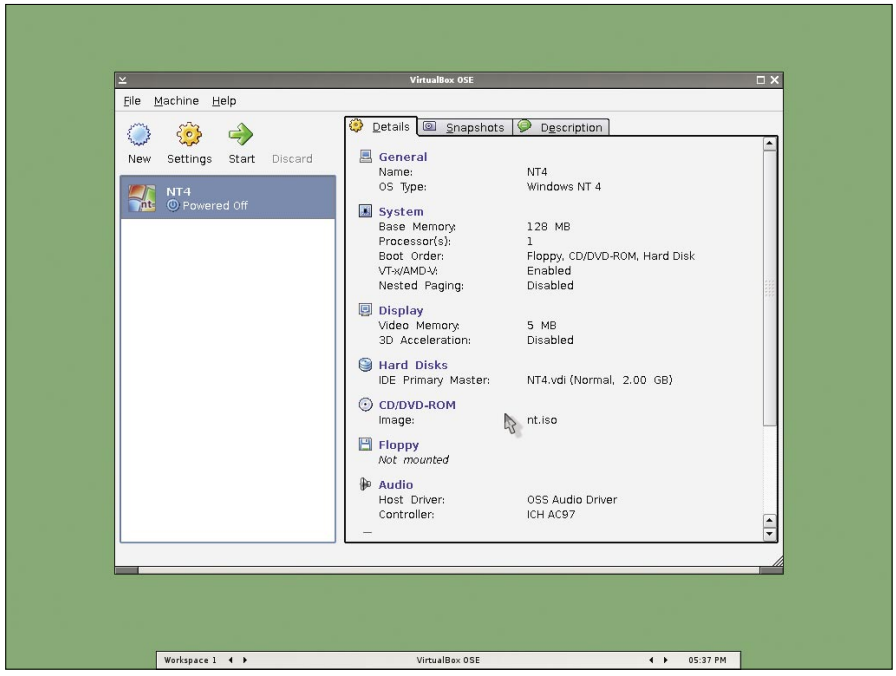


Figure 1. A Microsoft NT4 virtual machine ready to run

Repeat as necessary for each distribution, remove the CD/DVD and reboot to pick up the `RC.CONF` and DEVFS changes:

```
reboot
```

Part 2 – Testing VirtualBox

Note: You can install the O/S software from CDROM with a VirtualBox session run from the server console provided you run VirtualBox as the root user – as a security measure X will not allow you to run VirtualBox as root via VNC. Alternatively, load the drivers from loader.conf at boot time rather than using the VBOXload script and log in via VNC as vboxuser. If you decide to use the root account to load CDROM's from the server console, change the default hard disk and machine folders in VirtualBox to a mounted filesystem with sufficient space for your disk images to expand – the root partition on FreeBSD is ~ 496Mb by default which is not sufficient capacity.

To access Blackbox on the server at 192.168.0.130 from another host use xvncviewer (or the client of your choice):

```
xvncviewer 192.168.0.130:1
```

Alternatively login to the server console as vboxuser and type startx.

Once your remote VNC session is established, run an xterm session by right-clicking on the Blackbox desktop and selecting xterm. Load the VBOXload script as root and check that the atapicam and 3 vbox kernel modules are loaded:

```
su name_of_your_guest_account
su
VBOXload
kldstat
exit
exit
```

If all the modules are loaded successfully, run VirtualBox from your xterm session:

```
VirtualBox
```

You should be greeted by the registration screen (See Step 1)

Part 3 – Building Virtual Machines

VM's can be built from virtually any OS provided the processor architecture is supported e.g. you cannot run an OS designed for a SPARC * box on an i386 version of Virtualbox, but you can run FreeBSD i386 under an 64 bit Intel or AMD environment. BSD, Linux, Minux, FreeDOS and the various offerings from Microsoft *, Sun * (OpenSolaris) and Apple * (i386) should run without any problem.

Novell Netware * seems to have problems though, but I didn't have a copy to test to confirm this.

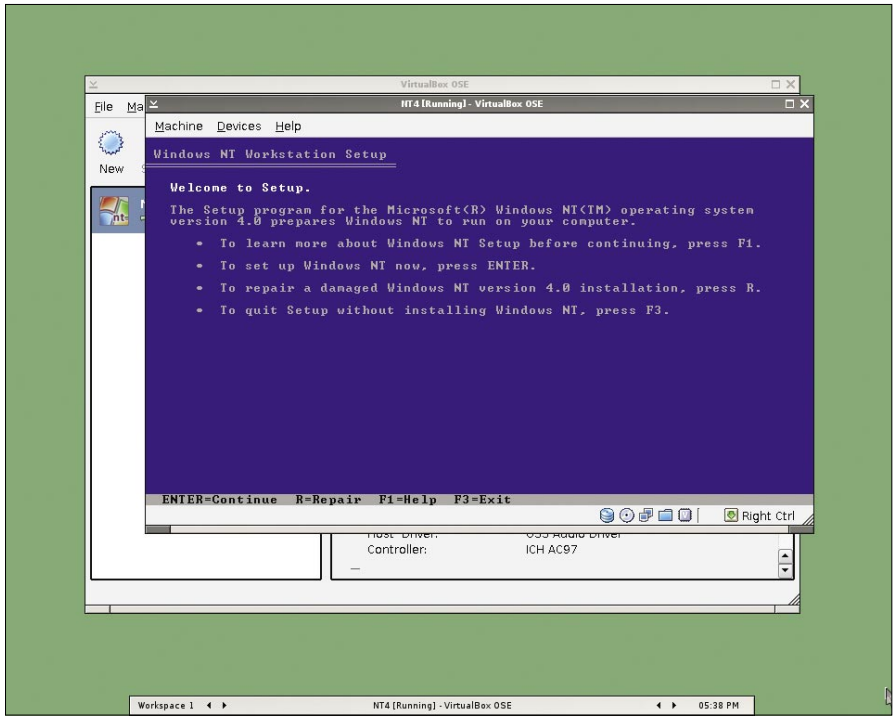


Figure 2. NT4 install screen

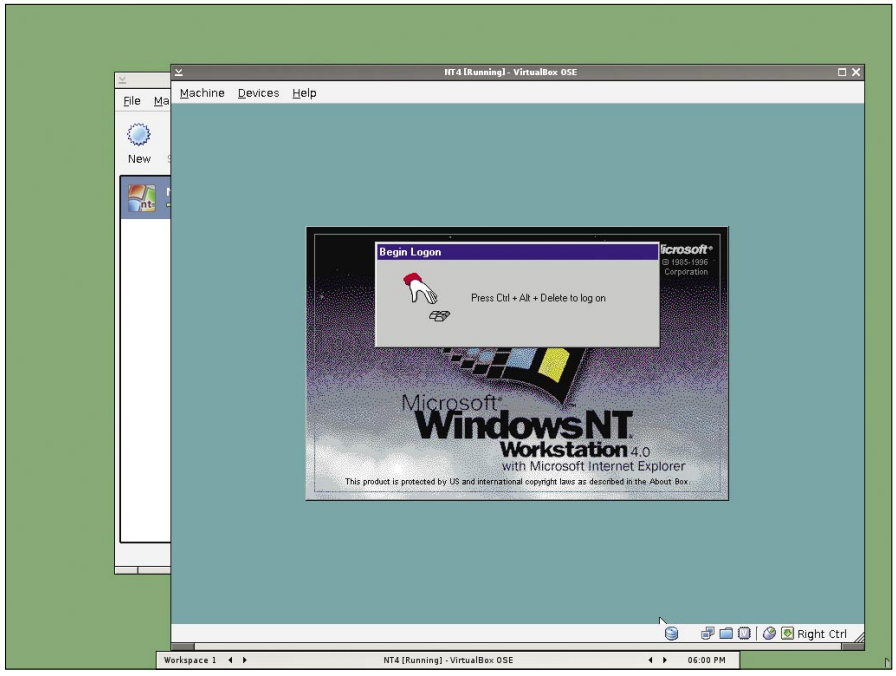
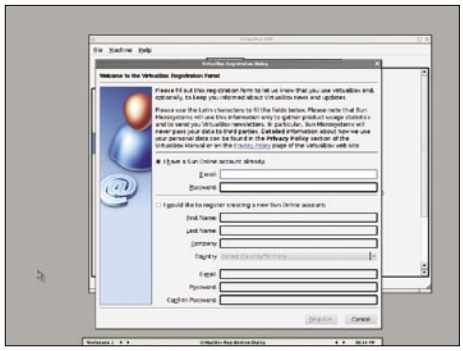
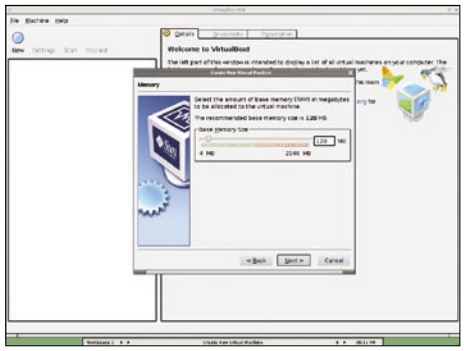


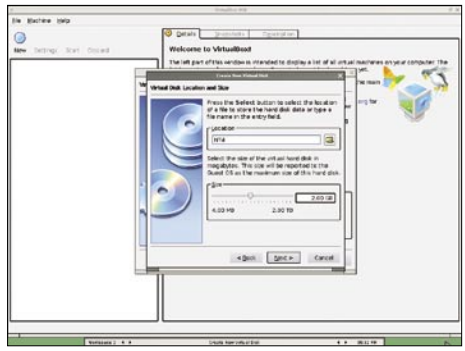
Figure 3. NT4 in glorious 16 colours using the stock video support



Step 1. Initial registration screen

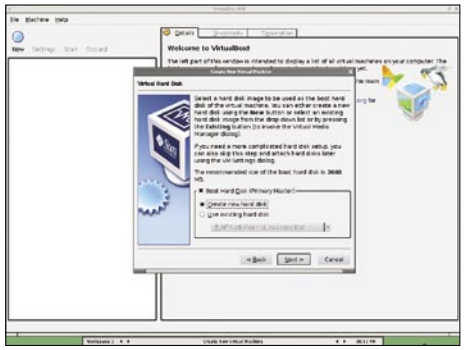


Step 5. Allocating memory to the guest OS



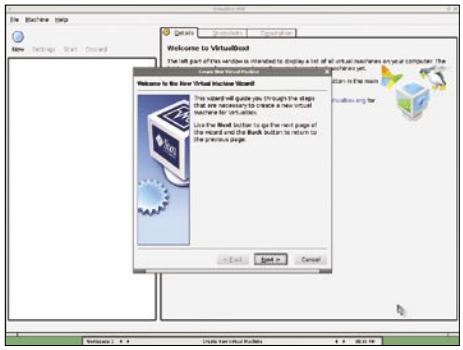
Step 9. Location and selecting size of virtual hard disk

Step 2. VirtualBox GUI with no VM's loaded



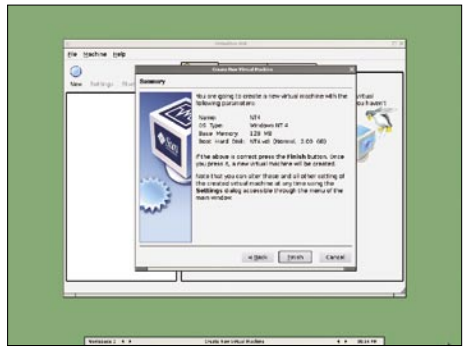
Step 6. Creating a new hard disk image

Step 10. Disk wizard summary

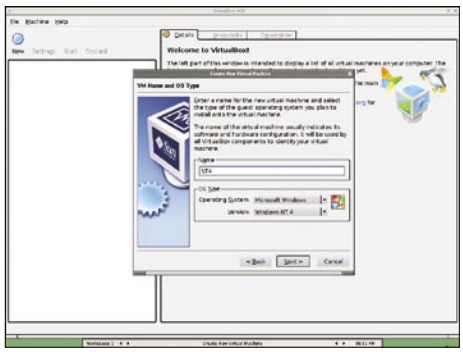


Step 3. Creating a new VM

Step 7. Disk wizard

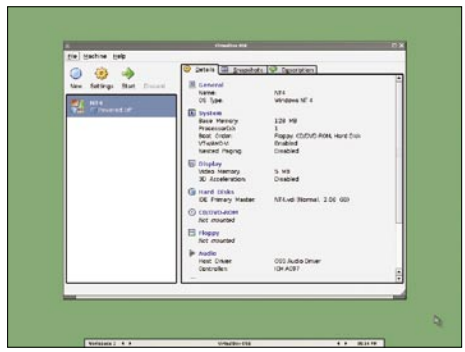


Step 11. Virtual Machine summary



Step 4. Naming and selecting the guest OS and version

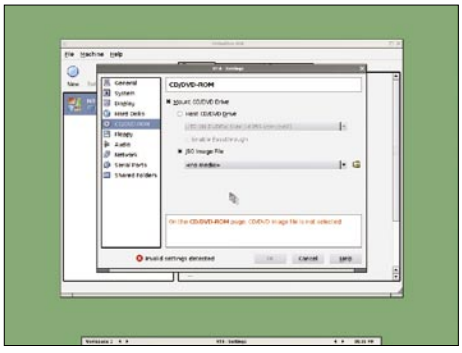
Step 8. Selecting dynamic or fixed size storage



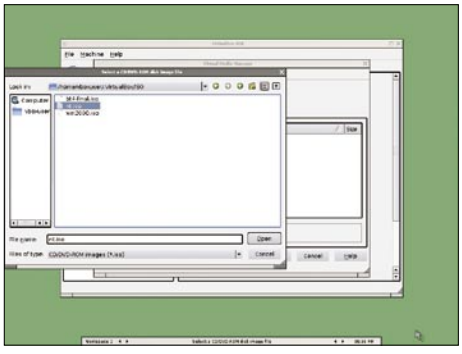
Step 12. A newly created VM in the GUI – No valid boot device available

Key points to note when building VM's:

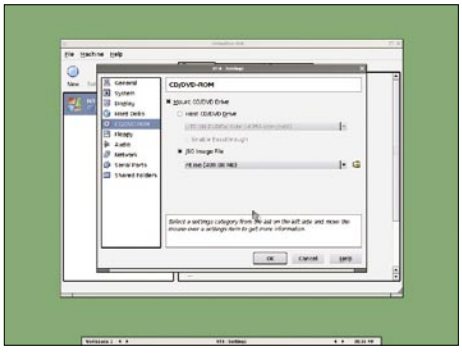
1. Ensure the VM meets the minimum (or maximum!) requirement of the target OS for memory, disk space etc. For instance NT4 * baulked at the 8GB default partition that VirtualBox provides.
2. Ensure you have enough storage on your VM host. On older hard disks, at 8GB per chunk space can be eaten up quickly. While the VM may start OK, if you have chosen dynamic storage, the image grows according to the demands of the OS and this can lead to capacity issues.
3. Most OS's play well with the default hardware provided by VirtualBox, but sometimes video drivers may need to be tweaked with older OS's (NT4 is an example). This caveat also applies to some versions of Windows 7 * that do not recognise the network card, but this was not found in the later versions I tried
4. Guest additions are not mandatory, all the VM's shown in Figure 5 were installed without them. Your mileage may vary though.



Step 13. Selecting the CDROM as the boot device for new VM



Step 14. Choosing an ISO image



Step 15. ISO image selected

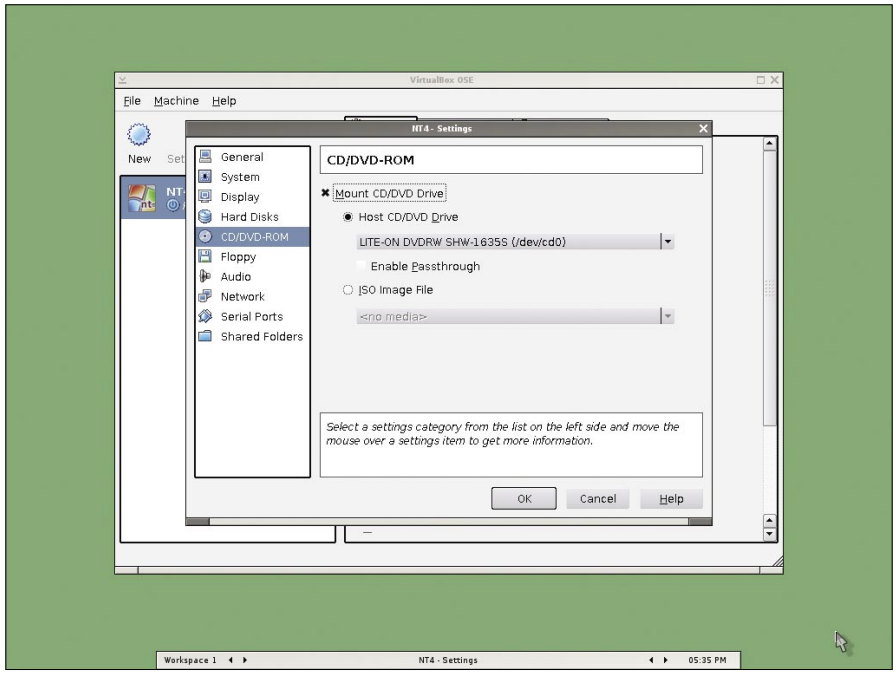


Figure 4. Booting the guest VM from CDROM

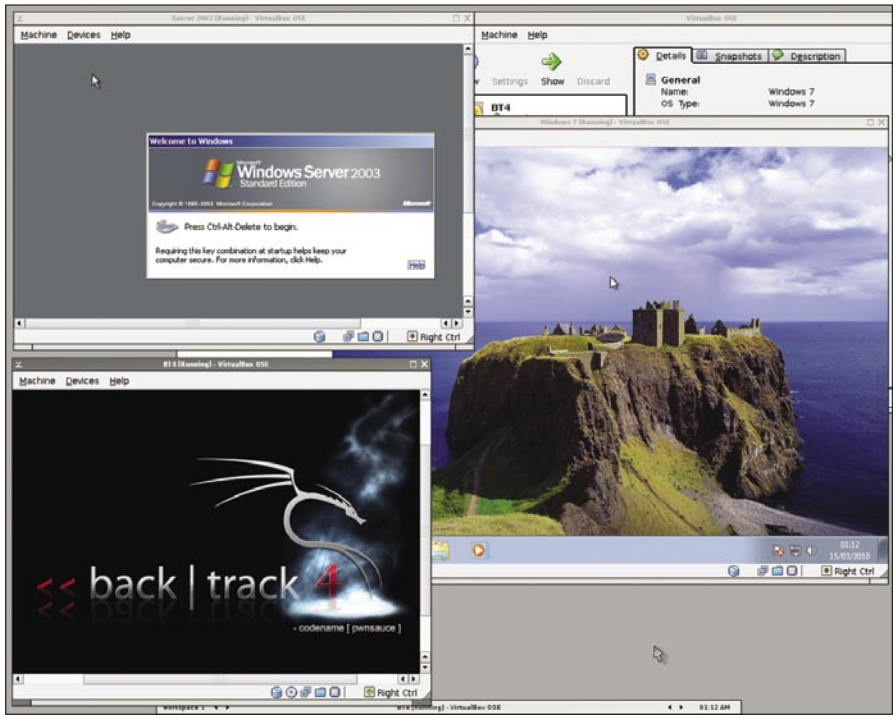


Figure 5. Windows Server 2003, Windows 7 and BackTrack 4 VM's running simultaneously

5. To build your VM using an ISO image, follow the diagrams in Steps 1 – 15. If you want to use the vendor supplied CDROM, refer to Figure 4.

Improving this configuration

When you are happy with the way VirtualBox performs, add support at boot by adding these lines to `/boot/loader.conf`:

```
atapicam_load="YES"
vboxdrv_load="YES"
vboxnetflt_load="YES"
vboxnetadp_load="YES"
```

You will no longer need to run VBOXload prior to loading VirtualBox after a reboot.

It would be trivial to autostart VBOXload and vncserver via an rc script at boot, but this would probably be undesirable unless a firewall was installed on the server to limit access to the vncsession to certain clients. As VNC traffic is not encrypted (with the exception of the password), and the password is effectively limited to 8 characters, al least an SSL tunnel or equivalent form of encryption would be required in a production environment.

At time of writing, the guest additions were not available from the website (so could not be downloaded via VirtualBox), but they are available as a separate port from the freshports website.

Further reading

Virtualbox

- *http://wiki.freebsd.org/VirtualBox*
- *http://www.virtualbox.org/*

Downloadable VM images – treat all downloads as untrusted or sandbox accordingly

- *http://virtualboximages.com/*
- *http://sourceforge.net/projects/virtualboximage/files/*
- *http://virtualboxes.org/*

* All trademarks and copyrights acknowledged

FreeBSD Firewall with Transparent Proxy Server, DHCP Server and Name Server

Joshua Ebarvia

If you need Internet-sharing to be available to share allow your network to access the web using only one public IP Address, you need to setup a gateway.

Operating System: FreeBSD 8.0-RELEASE
Name Server: DNSMasq 2.52
DHCP Server: ISC DHCP Server 3.1
Proxy Server: Squid 3.1
URL Redirector: SquidGuard 1.4 using the black list of *www.shallalist.de*
Firewall: PF (OpenBSD Packet Filtering)

FreeBSD has all the tools and packages for your network to be able to access the Internet. It has also the services to filter the traffic requests to the web and block sites which are not appropriate according to your corporate IT rules. In short, all you need is a Firewall plus services that will make your network secure and easy to manage in terms of network configurations.

I will assume that you have a fully functional machine running FreeBSD 8.0 with two network interfaces, one with public IP address and the other one with a private IP address. Here is what your setup may look like see Figure 1.

I will not go on to details on installing FreeBSD and setting up both it's interface cards. Let's start!

Getting and updating the Ports Collection

The FreeBSD ports collection contains the list of packages that can be installed into Freebsd. If you don't have it yet, do the following:

```
# portsnap fetch
# portsnap extract
```

If you want to update it because you already have it,

```
# portsnap update
```

To learn more about the Ports Collection, read the FreeBSD Handbook

Setting up PF

PF will be our firewall for our setup. It is included in the FreeBSD base installation, but it is not enabled by default. There are two ways to enable it, using kldload and recompiling your kernel. I prefer the latter.

To enable PF in the kernel, you have to include the lines in the `/usr/src/sys/[$ARCH]/conf/GENERIC` where $ARCH may be i386, amd64 or whatever architecture you use. Use your favorite text editor to add the entries below at the end of the file.

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

After making changes, you have to recompile your kernel and reboot your system.

```
# cd /usr/src
# make buildkernel KERNCONF=GENERIC
# make installkernel KERNCONF=GENERIC
# reboot
```

Take note that the FreeBSD source tree should be available for you to be able to build your customized kernel. The entire operation time depends on your hardware, just be patient and wait.

Next, we have to create our `/etc/pf.conf` or the so called firewall rules to enable traffic from the entire network accessing the web to go the proxy server first for filtering. Your minimal pf.conf may look like this see Listing 1.

The above configuration file allows the clients to access the web, ftp sites, and https sites. Take note that only access to the web goes

to the proxy server first. Https should not be redirected. The following entries should also be appended to your `/etc/rc.conf`

```
gateway_enable="YES"
pf_enable="YES"
pf_rules="/etc/pf.conf"
pflog_enable="YES"
```

Make sure to restart PF every after changing your `/etc/pf.conf` by

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

or

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

To learn more about PF, visit *http://www.openbsd.org/faq/pf/*.

Installing and configuring DNSMasq

DNSMasq is a lite name server. It has been said that it works on a thousand clients very well. DNSMasq will serve as our resolver for name resolution and IP address lookups.

To install it, navigate to the *Ports Collection* (You need root privileges)

```
# cd /usr/ports/dns/dnsmasq
# make install clean
```

By default, it has a sample config file at `/usr/local/etc/`. Make a copy of it and name it `dnsmasq.conf`. You don't have to make changes to it unless it is necessary.

To start DHCP server, you have to add `dnsmasq_enable="YES"` to your `/etc/rc.conf`. and

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

You may change start to stop or restart, depending on what operation you want.

Installing and configuring ISC DHCP Server

In your network, you want to have an automatic configuration of your clients' networking device. This is the job of a DHCP server. We will be using ISC DHCP Server to set the ipv4 address, default gateway, DNS server, and netmask of the client inside the network.

To install ISC DHCP Server, navigate to the Ports Collection (you need root privileges)

```
# cd /usr/ports/net/isc-dhcp31-server
# make install clean
```

A sample configuration file is included in the installation and is located at `/usr/local/etc/` directory. You may want to copy it so that when you mess things up, you have a file to work with. The configuration file of the DHCP server is named dhcpd.conf and should be located at `/usr/local/etc/dhcpd.conf`. The entries there are straightforward, but for a simple setup, your file may look something like this see Listing 2.

The domain-name specifies the domain name that will be given to the clients. The domain-name-servers may be one or more IP addresses, separated by a whitespace. In our case, we want our gateway to be also the DNS server used by the clients in the network. The lease-times are in seconds. Then we define the subnet. With an entry of 192.168.0.10 to 192.168.0.100, meaning only addresses in that range will be given to the clients.

To start DHCP server, you have to add `dhcpd_enable="YES"` to your `/etc/rc.conf`. Then

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

You may change start to stop or restart, depending on what operation you want.

To learn more about ISC-DHCP, visit *http://www.isc.org/software/dhcp*.

Installing and configuring Squid and SquidGuard

In a typical network setting, a system administrator would setup a proxy server in which clients within the network will use to access the web and it's services. Squid is a caching proxy for the web. It is capable of optimising the clients' connection to the web by caching and reusing frequently visited web pages. Not just that, it can also act as a transparent proxy, meaning you don't have to setup all your clients' browser to enter details such as the IP Address or hostname and ports for a specific proxy server to use.

With Squid, you will be able to do a transparent proxy that will eliminate your work on manually setting all the clients' broswers to a specific proxy server.

Another important thing about Squid is it's site blocking/redirecting feature. Although you can setup Squid to block sites using its configuration file and a text file containing the desired sites to block, it is generally recommended to use another program for that purpose. Here comes SquidGuard. It will be the redirection program that will be used by squid.

To install Squid and SquidGuard, navigate to the ports collection directory (you need to have root privileges)

```
# cd /usr/ports/www/squid31
# ./configure --enable-pf-transparent
# make && make install
# cd /usr/ports/www/squidguard
# ./configure
# make && make install
```

The key configuration files used by Squid and SquidGuard are squid.conf and squidGuard.conf respectively and are located at `/usr/local/etc/squid/` directory. There you will find a sample configuration file for each. You may want to make a copy of it first so that when you mess things up, you'll be having a default config file to work on.

To configure Squid as a transparent proxy and use SquidGuard, you have to

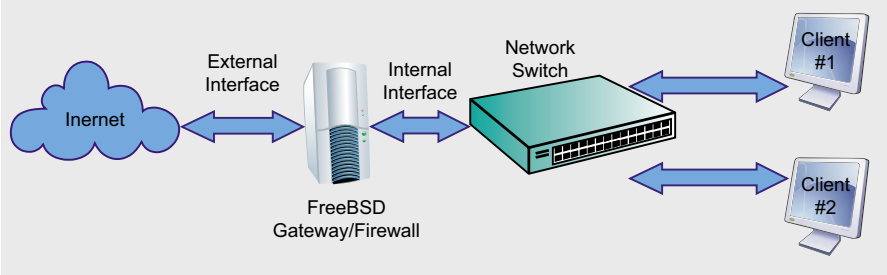


Figure 1.

edit `/usr/local/etc/squid/squid.conf`. Use your favorite text editor for this.

The sample configuration file supplied is straightforward. You just have to add and edit a few lines to make transparent proxy and blocking work. First you have to add your network in the acl list. Find the line `# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS`. Right after it insert your network using something like this,

Listing 1. Minimal pf.conf

```
int="em1"          #internal interface change the device to fit your setup
ext="em0"          #internal interface change the device to fit your setup
lan=$int:network
gw="127.0.0.1"
tcp_services = „{www, ftp-proxy, ftp-data, ftp}"
udp_services = „{ domain, ntp}"
icmp_types = „{ echoreq, unreach }"
www="{ 80:83, 1080, 8080:8081, 8088, 11523}"

nat-anchor „ftp-proxy/*"
rdr-anchor „ftp-proxy/*"

#-----NAT on $ext on traffic from $int to $ext
nat on $ext from $lan to any -> $ext

# Redirect ftp traffic to ftp-proxy
rdr on $int inet proto tcp from $lan to any port ftp -> $gw port ftp-proxy

# Redirect all www traffic to squid proxy server
rdr on $int inet proto tcp from $lan to any port $www -> $gw port 3128

# Blocks all in and out traffic and logs them via pflog0
block log all

# This is needed for FTP proxy
anchor „ftp-proxy/*"

antispoof quick for {lo $int}

# Allow ping IN and OUT
pass inet proto icmp all icmp-type $icmp_types

#-------------Squid Transparent Proxy----
pass in on $int inet proto tcp from $lan to $gw port 3128
pass out on $ext inet proto tcp from $gw to any port 3128

#------------- HTTPS Access  ----
pass in on $int inet proto {tcp, udp} from $lan to any port https
pass out on $ext inet proto {tcp, udp} from $lan to any port https

#-------------FTP Access -------
pass in on $int inet proto {tcp, udp} from $lan to any port ftp:ftp-proxy
pass out on $ext inet proto {tcp, udp} from $ext to any port ftp:ftp-proxy

#------------------Make udp services work----
pass inet proto {tcp, udp} from $lan to $gw port $udp_services
```

```
acl my_network src 192.168.0.0/24
http_access allow my_network
```

You have to change the network address to fit your needs. `my_network` is the name given to your acl or the *access control list*.

Next, you have to look for the line `http_port 3128`. You have to change it to

```
http_port 127.0.0.1:3128 transparent
```

Port 3128 is the default port used by squid and the word *transparent* is needed to use Squid in transparent mode.

You have to add other options to make things a little bit hidden. At the end of the configuration file, append the following lines:

```
forwarded_for off
visible_hostname localhost
cache_mgr administrator@your.domai
n.com
```

`forwarded_for` off makes your private IP address invisible to the outside world

`visible_hostname` specifies your proxy servers hostname to the outside world

`cache_mgr` specifies the email address of the administrator

To make SquidGuard the program for redirection, append this line at the end.

```
url_rewrite_program /usr/local/
bin/squidGuard -c/usr/local/etc/squid/
squidGuard.conf
```

This specifies the path of squidGuard command and its configuration file.

It is recommended to change the sample black list that came with SquidGuard. I recommend the black list from *http://www.shallalist.de*. To download it use the fetch command and extract it to `/var/db/squidGuard`

```
# fetch http://www.shallalist.de/
Downloads/shallalist.tar.gz
# gzip -d shallalist.tar.gz
# tar -xvf shallalist.tar
```

You will have a directory named BL after extracting the archive. You have to move all the contents of BL to `/var/db/squidGuard`

```
# mv BL/*  /var/db/squidGuard
```

There are lots of *categories/directories* on your black list. You need to compile them in order for SquidGuard to use them. But it takes time, depending on your system. So, suppose we just want to block all that is in porn and adv, we have to compile it. But before compiling it, we have to edit the configuration file of SquidGuard which is `/usr/local/etc/squid/squidGuard.conf`. Use your favorite text editor. Your minimal configuration file should look like this see Listing 3.

The dest block specifies the name of the category you want to block. The name should correspond to the directory inside `/var/db/squidGuard`. The acl block specifies which ones to pass and which ones to block, in our case the `!porn` and `!adv` means that all of the sites on the domainlist and urllist of dest porn and adv are blocked. The all keyword means that everything else not included in porn and adv is allowed. The redirect line specifies the address where the client will be redirected when accessing the blocked sites. You may change it to any address you want.

You have to compile the files inside the `/var/db/squidGuard/porn` and adv

```
# squidGuard -C all
```

After compiling you will see two new additional files namely `urls.db` and `domains.db`. Make sure they are executable and that it is owned by squid

```
# cd /var/db/squidGuard/porn
# chmod g+x *.db
# chown squid:squid *.db
# cd /var/db/squidGuard/adv
# chmod g+x *.db
# chown squid:squid *.db
```

Here are the steps to test Squid and SquidGuard.

Make sure that you have the `squid_enable="YES"` in your `/etc/rc.conf`

To launch Squid, you can change start with stop or restart.

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

To check if Squid uses Squidguard,

```
# ps ax | grep squid
```

and you will see five (5) lines similar to this

```
67913  ??  S 0:04.99 (squidGuard) -c
/usr/local/etc/squid/squidGuard.conf
```

To check if the redirector is working try an entry from the domains in adv

```
#echo "http://ads.inet.co.th / - -
GET" | squidGuard -c /usr/local/etc/
squid/squidGuard.conf
   http://www.wheredoyouwant2meredirect
.sample.com -/- - GET
```

If the redirector is working you will see the URL of the one you have specified in your `squidGuard.conf`. If you change *http://ads.inet.co.th* to let say *http://www.google.com*, then the output should be a blank line meaning that access *www.google.com* is allowed.

You can learn more about Squid at *http://www.squid-cache.org/* and Squidguard at *http://www.squidguard.org/*.

Remember that every change in the squid.conf or the `squidguard.conf` needs the service squid to be restarted. You may do so by

```
# squid -k reconfigure
```

or

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

The former is recommended as it doesn't stop the service and applies the new changes on the fly.

Your gateway is ready to go.

Listing 2. Configuration file of the DHCP server

```
option domain-name "my_network.intranet";
option domain-name-servers 192.168.0.1;
default-lease-time 18000;
max-lease-time 36000;
authoritative;
ddns-update-style none;
log-facility local7;
subnet 192.168.0.0 netmask 255.255.255.0{
   range 192.168.0.10 192.168.0.100;
   option routers 192.168.0.1;
}
```

Listing 3. Minimum configuration file

```
# SAMPLE CONFIG FILE FOR SQUIDGUARD
dbhome /var/db/squidGuard
logdir /var/log
dest porn {
        domainlist porn/domains
        urllist porn/urls
}
dest adv{
        domainlist adv/domains
        urllist adv/urls
}
acl {
     default {
         pass     !porn !adv all
         redirect http://www.wheredoyouwant2meredirect.sample.com
     }
}
```

The Squid and the Blowfish

Daniele Mazzocchio

We have grown so much accustomed to Internet access on our work computers, that we can hardly imagine what people ever did all day long on their workplace before!

By providing access to a virtually endless amount of information, the Internet has quickly turned into an essential working tool. So essential that most companies can't do without it anymore. But besides providing a huge amount of information, the Internet has also turned into the main virus vehicle (together with e-mail) and doesn't exclusively provide content in line with corporate policies. That's why a proxy server is often as necessary as the Internet connection itself.

The main benefits of web proxying are:

- content filtering: the proxy can be configured to filter out virus files, ad banners and requests to unwanted websites;
- network bandwidth conservation: cached pages are served by the proxy itself, thus saving bandwidth and offering faster access times;
- authentication: Internet access can be authorized (and filtered) based on username/password, IP address, domain name and much more.

The following is the list of the pieces of software we will use:

- OpenBSD *http://www.openbsd.org/* – a robust, security-oriented operating system, with *only two remote holes in the default install, in a heck of a long time!*;
- Squid *http://www.squid-cache.org/* – a *caching proxy for the Web supporting HTTP, HTTPS, FTP, and more*;
- SquidGuard *http://www.squidguard.org/* – a *combined filter, redirector and access controller plugin for Squid*;
- ClamAV *http://www.clamav.net/lang/pl/* – a fast and easy-to-use open-source virus scanner;
- SquidClamav *http://www.darold.net/projects/squidclamav/* – an *open source (GPL) anti-virus toolkit for UNIX*;
- AdZapper *http://adzapper.sourceforge.net/* – a *redirector for squid that intercepts advertising (banners, popup windows, flash animations, etc), page counters and some web bugs (as found).*

The choice of using free software prevented me from using *DansGuardian* (*http://dansguardian.org/*), an Open Source web content filter, running on many OSes and filtering the actual content of pages based on many methods including phrase matching, PICS filtering and URL filtering. Fine and dandy, but it is *not* free for commercial use (*http://dansguardian.org/?page=copyright2*).

A good knowledge of OpenBSD is assumed, since we won't delve into system management topics such as OS installation and base configuration, packages/ports installation or PF syntax.

Squid

Squid is a *a full-featured HTTP/1.0 proxy* and it *offers a rich access control, authorization and logging environment to develop web proxy and content serving applications.*

Installation

Let's start with the location of the cache server in the network: according to the documentation (*http://www.deckle.co.za/squid-users-guide/Squid_Configuration_Basics#DMZ*), the most suitable place is in the DMZ; this should keep the cache server secure while still able to peer with other, outside, caches (such as the ISP's).

The documentation also recommends setting a DNS name for the cache server (such as `cache.mydomain.tld` or `proxy.mydomain.tld`) as soon as possible: *a simple DNS entry can save many hours further down the line. Configuring client machines to access the cache server by IP address is asking*

for a long, painful transition down the road.

Squid installation is as simple as it can be; you only have to add the Squid package. Available flavors are `ldap` (allowing for LDAP authentication) and `snmp` (including SNMP support) (see Listing 1).

Listing 1. Installation

```
# export PKG_PATH=/path/to/your/favourite/OpenBSD/mirror
# pkg_add squid-x.x.STABLExx-snmp.tgz
squid-x.x.STABLExx-snmp: complete

--- squid-x.x.STABLExx-snmp -------------------
NOTES ON OpenBSD POST-INSTALLATION OF SQUID x.x

The local (OpenBSD) differences are:
configuration files are in              /etc/squid
sample configuration files are in       /usr/local/share/examples/squid
error message files are in               /usr/local/share/squid/errors
sample error message files are in       /usr/local/share/examples/squid/errors
icons are in                             /usr/local/share/squid/icons
sample icons are in                      /usr/local/share/examples/squid/icons
the cache is in                          /var/squid/cache
logs are stored in                       /var/squid/logs
the ugid squid runs as is                _squid:_squid

Please remember to initialize the cache by running „squid -z” before
trying to run Squid for the first time.

You can also edit /etc/rc.local so that Squid is started automatically:

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

#
```

Listing 2. Base Configuration

```
# Define the access log format
logformat squid  %ts.%03tu %6tr %>a %Ss/%03Hs %<st %rm %ru %un %Sh/%<A %mt
# Log client request activities ('squid' is the name of the log format to
use)
access_log       /var/squid/logs/access.log squid

# Log information about the cache's behavior
cache_log        /var/squid/logs/cache.log
# Log the activities of the storage manager
cache_store_log  /var/squid/logs/store.log
```

Base configuration

Squid configuration relies on several dozens of parameters, and thus can quickly turn into a very tricky task. Therefore, the best approach is probably starting with a very basic configuration and then tweaking the options, one by one, to meet your specific needs, while still making sure that everything keeps working as expected.

Actually, only a few parameters need to be set to get Squid up and running (theoretically, you could even run Squid with an empty configuration file): for all the options you don't explicitly set, the default values are assumed. Anyway, at least one setting must certainly be changed: the default configuration file denies access to all browsers; and this may sound a bit ...too strict!

Our first configuration will be very simple: we will place our proxy server in the DMZ (172.16.240.0/24, below is the network layout) and allow only requests from the LAN (172.16.0.0/24). No ISP's parent proxy is taken into account (see Figure 1).

The main Squid configuration file is `/etc/squid/squid.conf`. Let's have a look at it.

The `http_port` option sets the port(s) that Squid will listen on for incoming HTTP requests. There are three forms: port alone (e.g. `http_port 3128`), hostname with port (e.g. `http_port proxy.kernel-panic.it:3128`), and IP address with port (e.g. `http_port 172.16.240.151:3128`); you can specify multiple socket addresses, each on a separate line. If your Squid machine is multi-homed and directly accessible from the internet, it is strongly recommended that you force Squid to bind the socket to the internal address. This way, Squid will only be visible from the internal network and won't proxy the whole world! Squid's default HTTP port is 3128, but many administrators prefer using a port which is easier to remember, such as 8080.

```
http_port   3128
```

The `cache_dir` parameter allows you to specify the path, size and depth of the directories where the cache swap files will be stored. Squid allows you to have multiple `cache_dir` tags in your config file.

```
cache_dir   ufs /var/squid/cache 100
16 256
```

The above line sets the cache directory pathname to `/var/squid/cache`, with

a size of 100MB and 16 first-level subdirectories, each containing 256 second-level subdirectories. The cache directory must exist and be writable by the Squid process and its size can't exceed 80% of the whole disk. For further details, please refer to the documentation (*http://www.deckle.co.za/squid-users-guide/Squid_Configuration_Basics#Where_to_Store_Cached_Data*).

The `cache_mgr` parameter contains the e-mail address of the Squid administrator, which will appear at the end of the error pages; e.g.:

```
cache_mgr    webmaster@kernel-panic.it
```

The `cache_effective_user` and `cache_effective_group` options, allow you to set the UID and GID Squid will drop its privileges to once it has bound to the incoming network port. The package installation has already created the `_squid` user and group.

```
cache_effective_user    _squid
cache_effective_group   _squid
```

The `ftp_user` option sets the e-mail address that Squid will use as the password for anonymous FTP login. It's a good practice to use an existing address:

```
ftp_user    webmaster@kernel-panic.it
```

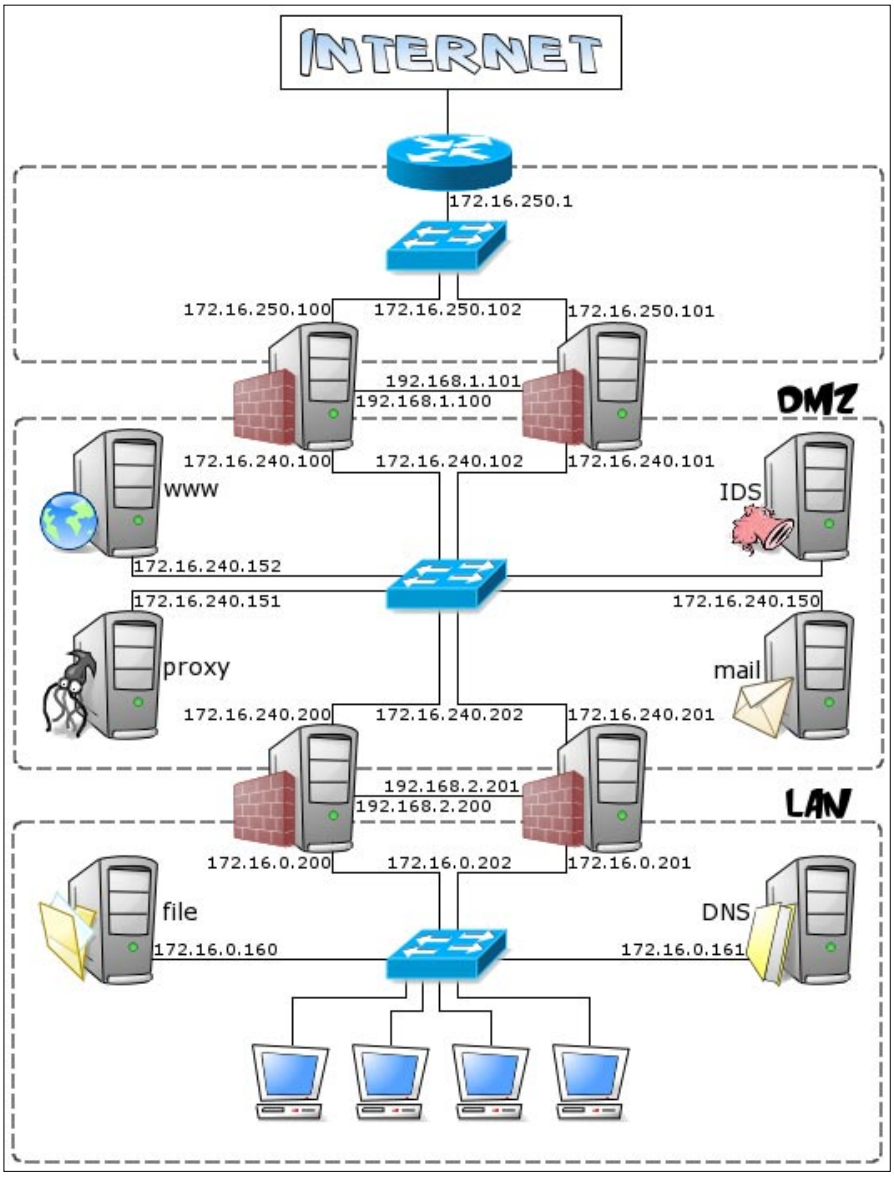


Figure 1. No ISP's parent proxy is taken into account

The following options set the paths to the log files; the format of the access log file, which logs every request received by the cache, can be specified by using a `logformat` directive (please refer to the documentation (*http://devel.squid-cache.org/customlog/logformat.html*) for a detailed list of the available format codes): see Listing 2.

And now we come to one of the most tricky parts of the configuration: *Access Control Lists*. The simplest way to restrict access is to only accept requests from the internal network. Such a basic access control can be enough in small networks, especially if you don't wish to use features like username/password authentication or URL filtering.

ACLs are usually split into two parts: acl lines, starting with the `acl` keyword and defining classes, and acl operators, allowing or denying requests based on classes. Acl-operators are checked from top to bottom and the first matching wins. Listing 3 is a very basic ruleset.

Starting Squid

Now our cache server is almost ready for a first run, just one last step to go. We first need to create the cache-swap directories where Squid will store cached pages. The `squid -z` command will create all the required directories, according to the `cache_dir` parameter in `squid.conf` (see above), as the user and group specified by the `cache_effective_user` and `cache_effective_group` parameters.

```
# /usr/local/sbin/squid -z
2009/05/15 18:04:35| Creating Swap
Directories
#
```

We are now ready to start Squid. Starting it in debug mode (`-d 1` flag) and in foreground (`-N` flag) will make it easier to see if everything is working fine (see Listing 4).

Once you get the *Ready to serve requests* message, you should be able to use the cache server. Once it is up and running, Squid reads the cache store:

the first time you should see all zeros, as above, because the cache store is empty.

Now, to make sure everything is working fine, we will configure our browser to use our fresh new proxy and we will try to access our favourite web site. In the `/var/squid/logs/access.log` file, you should see something like: see Listing 5.

For a detailed description of each field in the access.log file, please refer to the documentation (*http://www.deckle.co.za/squid-users-guide/Starting_Squid#Access.log_basics*). Anyway, `TCP_MISS` means that the requested page wasn't stored in the cache (either it was not present or it had expired); `TCP_HIT`, instead, means that the page was served from the cache. The second field is the time (in milliseconds) that Squid took to service the request: as you can see, it is much shorter when the page is cached. The page size is the fifth field: cached pages may be a little larger because of the extra headers added by Squid.

If everything is working fine, we can stop Squid:

```
# /usr/local/sbin/squid -k shutdown
```

and configure the system to start it on boot.

```
/etc/rc.local
if [ -x /usr/local/sbin/squid ]; then
    echo -n ' squid'
    /usr/local/sbin/squid
fi
```

You may also wish to start Squid through the `RunCache` script, which automatically restarts it on failure and logs both to the `/var/squid/squid.out` file and to syslog. Just remember to background it with an `&`, or it will hang the system at boot time.

Further Squid configuration

In many cases, the basic configuration we've seen in the previous chapter can be sufficient for accelerating web access and protecting the network, but Squid can do much more. Below are a just few of the many things Squid can do.

More on Access Control Lists

Though most people implement only very basic access control, Squid's access system is very powerful and flexible, allowing for in-depth filtering of access to cache resources. So far we have mainly dealt with ACLs that filter based on source IP address or destination port, but there are many other ACL types. In this paragraph, we will take a brief look at the main ones, just to get an idea of what Squid ACLs can do; for a more detailed and comprehensive description of Squid ACLs, please refer to the documentation (*http://www.deckle.co.za/squid-users-guide/Access_Control_and_Access_Control_Operators*).

A Squid ACL is made up of at least four fields: the `acl` keyword, followed by a (possibly descriptive) unique name, the ACL type and one or more decision strings. Thus, the overall syntax of Squid ACLs looks like:

```
acl name type (string|"filename")
[string2] [string3] ["filename2"]
```

An ACL containing multiple decision strings will return true if any of the decision strings matches (i.e. decision strings are ORed together). To avoid cluttering the configuration file with hundreds of ACL lines, you can specify the full pathname of a file (in double quotes) containing the decision strings one per line.

Listed below are the most commonly used ACL types:

- Source/Destination IP address – Filtering based on source IP address (`src` type) or destination IP address

Listing 3. Base Configuration. Basic Ruleset

```
# Classes
acl  all           src    all              # Any IP address
acl  localhost     src    127.0.0.0/8      # Localhost
acl  lan           src    172.16.0.0/24    # LAN where autorized clients
reside
acl  manager       proto  cache_object     # Cache object protocol
acl  to_localhost  dst    127.0.0.0/8      # Requests to localhost
acl  SSL_ports     port   443              # https port
acl  Safe_ports    port   80 21 443        # http, ftp, https ports
acl  CONNECT       method CONNECT          # SSL CONNECT method

# Only allow cachemgr access from localhost
http_access  allow  manager localhost
http_access  deny   manager

# Deny requests to unknown ports
http_access  deny   !Safe_ports

# Deny CONNECT to other than SSL ports
http_access  deny   CONNECT !SSL_ports

# Prevent access to local web applications from remote users
http_access  deny   to_localhost

# Allow access from the local network
http_access  allow  lan

# Default deny (this must be the last rule)
http_access  deny   all
```

Listing 4. Starting Squid

```
# /usr/local/sbin/squid -d 1 -N
2009/10/30 18:05:19| Starting Squid Cache version 2.7.STABLE6 for i386-
unknown-openbsd4.6...[ ... ]
2009/10/30 18:05:19| Accepting HTTP connections at 0.0.0.0, port 3128, FD 10.
2009/10/30 18:05:19| Accepting ICP messages at 0.0.0.0, port 3130, FD 11.
2009/10/30 18:05:19| Accepting SNMP messages on port 3401, FD 12.
2009/10/30 18:05:19| WCCP Disabled.
2009/10/30 18:05:19| Ready to serve requests.
2009/10/30 18:05:22| Done scanning /var/squid/cache (0 entries)
2009/10/30 18:05:22| Finished rebuilding storage from disk.
2009/10/30 18:05:22|         0 Entries scanned
2009/10/30 18:05:22|         0 Invalid entries.
2009/10/30 18:05:22|         0 With invalid flags.
2009/10/30 18:05:22|         0 Objects loaded.
2009/10/30 18:05:22|         0 Objects expired.
2009/10/30 18:05:22|         0 Objects cancelled.
2009/10/30 18:05:22|         0 Duplicate URLs purged.
2009/10/30 18:05:22|         0 Swapfile clashes avoided.
2009/10/30 18:05:22|   Took 5.1 seconds (   0.0 objects/sec).
2009/10/30 18:05:22| Beginning Validation Procedure
2009/10/30 18:05:22|   Completed Validation Procedure
2009/10/30 18:05:22|   Validated 0 Entries
2009/10/30 18:05:22|   store_swap_size = 0k
2009/10/30 18:05:22| storeLateRelease: released 0 objects
```

Listing 5. Starting Squid. Acces to Website

```
/var/squid/logs/access.log
1242419601.435    6735 172.16.0.13 TCP_MISS/200 11810 GET http://www.kernel-
panic.it/ - DIRECT/62.149.140.23 text/html1242419849.536     14 172.16.0.13
TCP_HIT/200 11820 GET http://www.kernel-panic.it/ - NONE/- text/html
[...]
```

Listing 6. Further Squid Configuration

```
# „Traditional” notation
acl  myNet1  src 192.168.0.0/255.255.255.0
# Address range with CIDR notation
acl  myNet2  src 172.16.0.0-172.16.2.0/24

# Filtering on destination address
acl  badNet  dst 10.0.0.0/24
```

Listing 7. Further Squid Configuration

```
# Match a specific site
acl  badDomain   dstdomain  forbidden.site
# Match the IP address of „forbidden.site”
acl  badDomainIP dst        1.2.3.4
```

(`dst` type). Both the traditional "IP/Netmask" and CIDR "IP/Bits" notations are allowed. E.g.: see Listing 6.

- Source/Destination Domain – Squid can allow/deny requests to or from specific domains (`dstdomain` and `srcdomain` types, respectively). If you want to deny access to a site, don't forget to also deny access to its IP address, or the rule will be easily bypassed. E.g.: see Listing 7. Regular expressions can also be used for checking the source domain (`srcdom_regex` type) and destination domain (`dstdom_regex` type) of a request. E.g.: see Listing 8.
- Words in the requested URL – Squid can use regular expressions to filter URLs matching specific patterns (`url_regex` type); if you don't care about the URL-type and the hostname, you can use the `urlpath_regex` type instead (Listing 9).
- Current day/time – Squid can allow/deny access to specific sites by time. The syntax is: `acl name time [day-list] [start_hour:minute-end_hour:minute]` where `day-list` is a list of single characters representing the days that the acl applies to (Sunday, Monday, Tuesday, Wednesday, THhursday, Friday, SAturday). E.g.:

```
acl workhours time  MTWHF 08:00-18:00
acl weekend   time  SA
acl morning   time  07:00-13:00
```

- Destination port – Squid can filter based on destination ports. E.g:

```
acl SSL_ports  port  443 563
acl Safe_ports port  80 21 443 563 70
210 280 488 591 777 1024-65535
```

- Protocol (FTP, HTTP, SSL) – The proto acl type allows Squid to allow/deny access based on the request protocol. E.g.:

```
acl www proto HTTP SSL
acl ftp proto FTP
```

- Method (HTTP GET, POST or CONNECT) – The `method` ACL type allows you to restrict access based

on the request HTTP `method`, i.e. `GET` (used for downloading), `POST` (used for uploading) and `CONNECT` (used for SSL data transfers). E.g.:

```
# Deny CONNECT to other than SSL ports
acl connect method CONNECT
http_access deny connect !SSL_ports
```

It is very important that you stop `CONNECT` type requests to non-SSL ports. The `CONNECT` method allows data transfer in any direction at any time, regardless of the transport protocol used. As a consequence, a malicious user could `telnet(1)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=telnet&sektion=1*)to a (very) badly configured proxy, enter something like: see Listing 10.

- Browser type – The `browser` acl type allows you to specify a regular expression that can be used to allow/deny access based on the `User-Agent` header. E.g.:

```
# Deny access to MS Internet Explorer
acl MSIE browser MSIE
http_access deny MSIE
```

- Username/Password pair – User authentication allows you to track Internet usage and collect per-user statistics. The simplest authentication scheme is the basic scheme, with username/password pairs stored in a file. To create this file, you can use the `htpasswd(1)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=htpasswd&sektion=1*)command:

```
# /usr/bin/htpasswd -c /etc/squid/
squid.passwd danix
New password: dAn1x
Re-type new password: dAn1x
Adding password for user danix
#
```

Authentication parameters are set using the `auth_param` tag; then, to actually activate authentication, you need to make use of ACLs based on login name in `http_access` (`proxy_auth` or `proxy_auth_regex`) or `external_acl_type` with `%LOGIN` used in the format tag. E.g.: see Listing 11.

Listing 8. Further Squid Configuration

```
# Match domains containing the word „sex” and a „.com” TLD (the match is case
# insensitive because of the ‘-i’ flag)
acl  badSites  dstdom_regex -i sex.*\.com$
```

Listing 9. Further Squid Configuration

```
# Match the most common video files extensions
acl  movies  urlpath_regex -i  \.avi$ \.mpg$ \.mpeg$ \.wmv$ \.asf$ \.mov$

# Match JPG images from URLs containing the word „sex”
acl  sexImg  url_regex -i  sex.*\.jpg$
```

Listing 10. Further Squid Configuration

```
$ telnet bad.proxy.tld 3128
Trying 1.2.3.4...
Connected to bad.proxy.tld.
Escape character is '^]'.
CONNECT telnet.server.tld:23 HTTP/1.1
and end up connected to the remote server, as if the connection was
originated by the proxy.
```

Listing 11. Further Squid Configuration

```
# Configure traditional (basic) proxy authentication
auth_param basic program /usr/local/libexec/ncsa_auth /etc/squid/squid.passwd

# Number of authenticator processes to spawn
auth_param basic children 5

# Realm to be reported to the client
auth_param basic realm Squid proxy-caching web server

# Usernames are case insensitive
auth_param basic casesensitive off

# Credentials time to live
auth_param basic credentialsttl 12 hours

# Using REQUIRED will accept any valid username
acl AUTH proxy_auth REQUIRED

# Don’t require authentication to localhost
http_access allow localhost

# Only allow authenticated requests coming from the LAN
http_access allow AUTH lan

# Default deny
http_access deny all
```

- SNMP Community – Squid can restrict SNMP queries based on the requested SNMP community. E.g.: see Listing 12.

Http-accelerator mode (reverse proxy)

According to the documentation (*http://www.deckle.co.za/squid-users-guide/Accelerator_Mode#When_to_use_Accelerator_Mode*), enabling Squid's Accelerator Mode can be useful only in a limited set of circumstances:

- accelerating a slow server;
- replacing a combination cache/web server with Squid;
- transparent caching;
- protecting an insecure web server.

Besides these cases, enabling the accelerator mode is strongly discouraged. The configuration is very simple; below is a sample configuration of a Squid server accelerating requests to a slow web server (see Listing 13).

Listing 12. Further Squid Configuration

```
# Address of the cache administrator
acl snmpManager  src  172.16.0.100

# Non-sensitive information
acl SNMPPublic  snmp_community public
# Allow any request from the cache administrator
snmp_access  allow  snmpManager

# Clients on the LAN can only query non-sensitive information
snmp_access  allow  SNMPPublic lan

# Default deny
snmp_access  deny  all
```

Listing 13. Http-accelerator mode (reverse proxy)

```
/etc/squid/squid.conf
# In accelerator mode, Squid usually listens on the standard www port
http_port   80 accel vhost

# Do the SSL work at the accelerator level. To create the certificates, run:
#   openssl req -x509 -newkey rsa:2048 -keyout squid.key -out squid.crt \
#   -days 365 -nodes
https_port  443 cert=/etc/ssl/squid.crt key=/etc/ssl/private/squid.key

# Accelerated server address and port
cache_peer  172.16.1.217 parent 80 0 no-query originserver

# Do not rewrite 'Host:' headers
url_rewrite_host_header off
# Process multiple requests for the same URI as one request
collapsed_forwarding    on

# Allow requests when they are to the accelerated machine AND to the
# right port
acl webSrv    dst   172.16.1.217
acl webPrt    port  80
acl all       src   0.0.0.0/0.0.0.0
http_access   allow webSrv webPrt
http_access   allow all
always_direct allow webSrv
```

Transparent caching

Transparent caching means having a filtering device, such as a router or a firewall, silently redirecting web traffic to the cache server. Clients ignore the presence of the proxy between them and the web server and think they're talking directly to the server.

As a consequence, transparent caching doesn't require any configuration on the client side, thus making maintenance much easier and faster. On the other hand, however, a transparently intercepting proxy can't use authentication or transparently proxy the HTTPS protocol.

Before configuring Squid, we will need to enable web traffic redirection on our firewalls (the involved firewalls are those between the LAN, where clients reside, and the DMZ, where the cache server is placed). Below are some sample rules for the `pf.conf(5)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=pf.conf&sektion=5*) file: see Listing 14.

Squid configuration is quite simple:

```
/etc/squid/squid.conf
# Port on which connections are
redirected
http_port  3128 transparent
```

SNMP

SNMP is a set of protocols for network management and monitoring. If you installed the *snmp* flavor of the Squid package, the proxy will be able to serve statistics and status information via SNMP.

SNMP configuration is rather simple: see Listing 15.

You can test whether SNMP is working with the `snmpwalk` program (`snmpwalk` (*http://net-snmp.sourceforge.net/docs/man/*

snmpwalk.html) is part of the NET-SNMP *http://net-snmp.sourceforge.net/* project). E.g.: see Listing 16.

Please refer to the documentation (*http://wiki.squid-cache.org/Features/Snmp?action=show&redirect=SquidFaq/SquidSnmp#head-edb6affeb8aa4364a710048e20f0ce125e5b8244*) for a detailed explanation of the output from the snmpwalk command.

Content filtering with SquidGuard

SquidGuard (*http://www.squidguard.org/*) is a *combined filter, redirector and access controller plugin for Squid*. We will use it to block access to specific categories of unwanted sites, based on IP addresses, URLs and regular expressions. SquidGuard comes with a very comprehensive list of commonly-banned web sites, divided into categories such as *porn*, *drugs*, *ads* and so on, making configuration rather simple and fast.

Installation

SquidGuard is available through OpenBSD's packages and ports system (*http://www.openbsd.org/faq/faq15.html*) and requires the installation of the following packages:

- db-x.x.x.tgz
- squidGuard-x.x.x.tgz

The installation places a copy of the blacklists tarball (`blacklists.tar.gz`) in `/usr/local/share/examples/squidguard/dest/`. We will extract it into the `/var/squidguard/db` directory:

```
# cd /usr/local/share/examples/
squidguard/dest/
# mkdir -p /var/squidguard/db
# tar -zxvC /var/squidguard/db -f
blacklists.tar.gz
[...]
```

Configuration

SquidGuard's configuration file is `/etc/squidguard/squidguard.conf`; it is logically divided into six sections (please refer to the documentation (*http://www.squidguard.org/Doc/*) for a more in-depth look at squidGuard's configuration options):

Listing 14. Transparent Caching

```
/etc/pf.conf
[...]
# LAN interface
lan_if     = rl1

# Cache server and port
cache_srv  = proxy.kernel-panic.it
cache_port = 3128

# Transparently redirect web traffic to the cache server
rdr on $lan_if proto tcp from $lan_if:network to any port www -> \
    $cache_srv port $cache_port
[...]
```

Listing 15. SNMP Configuration

```
/etc/squid/squid.conf
# By default, Squid listens for SNMP packets on port 3401, to avoid
conflicting
# with any other SNMP agent listening on the standard port 161.
snmp_port  3401

# Address to listen on (0.0.0.0 means all interfaces)
snmp_incoming_address 0.0.0.0

# Address to reply on (255.255.255.255 means the same as snmp_incoming_
address)
# Only change this if you want to have SNMP replies sent using another
address
# than where Squid listens for SNMP queries.
# snmp_incoming_address and snmp_outgoing_address can't have the same value
# since they both use port 3401.
snmp_outgoing_address 255.255.255.255

# Configuring access control is strongly recommended since some SNMP
# information is confidential
acl  all              src             0.0.0.0/0.0.0.0
acl  lan              src             172.16.0.0/24
acl  snmpManager      src             172.16.0.100
acl  publicCommunity  snmp_community  public
snmp_access           allow           snmpManager
snmp_access           allow           publicCommunity lan
snmp_access           deny            all
```

Listing 16. SNMP Configuration

```
# snmpwalk -c public -v 1 proxy.kernel-panic.it:3401 .1.3.6.1.4.1.3495.1.1
SNMPv2-SMI::enterprises.3495.1.1.1.0 = INTEGER: 356
SNMPv2-SMI::enterprises.3495.1.1.2.0 = INTEGER: 744
SNMPv2-SMI::enterprises.3495.1.1.3.0 = Timeticks: (540791) 1:30:07.91
#
```

- Path declarations – Specify the path to the logs and blacklists directories:

```
logdir /var/squidguard/log
dbhome  /var/squidguard/db
```

- Time space declarations – SquidGuard allows you to have different access rules based on time and/or date. A short example will probably best illustrate the flexibility of these rules (Listing 17).
- Source group declarations – SquidGuard allows you to filter based on source IP address, domain and user (users credentials are passed by Squid along with the URL); e.g.: see Listing 18.
- Destination group declarations – One of the main features of SquidGuard is certainly its ability to filter based on destination address or domain. And this is where the pre-built databases we extracted before come in handy. The `domainlist` parameter specifies the path to a file containing a list of domain names (later on, we will see how to create the db files to speed up SquidGuard startup time): this must be a relative path rooted in the directory specified by the `dbhome` parameter. Similarly, the `urllist` and `expressionlist` parameters specify the (relative) path to files containing a list of URLs and regular expressions respectively. E.g.: see listing 19.
- Access control rule declarations – Finally, we can combine all the previous rules to build Access Control Lists: see Listing 20.

Listing 17. SquidGuard Configuration

```
time workhours {
    weekly  mtwhf  08:00-18:00
}

time night {
    weekly  * 18:00-24:00
    weekly  * 00:00-08:00
}

time holidays {
    date    *.01.01                 # New Year's Day
    date    *.05.01                 # Labour Day
    date    *.12.24 12:00-24:00     # Christmas Eve (short day)
    date    *.12.25                 # Christmas Day
    date    *.12.26                 # Boxing Day
}
```

Listing 18. SquidGuard Configuration

```
src admin {
    ip      172.16.0.12               # The administrator's PC
    domain  lan.kernel-panic.it       # The LAN domain
    user    root administrator        # The administrator's login names
}

src lan {
    ip      172.16.0.0/24             # The internal network
    domain  lan.kernel-panic.it       # The LAN domain
}
```

Listing 19. SquidGuard Configuration

```
dest porn {
    domainlist     blacklists/porn/domains
    urllist        blacklists/porn/urls
    expressionlist blacklists/porn/expressions
    # Logged info is anonymized to protect users' privacy
    log anonymous  dest/porn.log
}
```

The redirect rule declares the URL where to redirect users requesting blocked pages. SquidGuard can include some useful information in the URL by expanding the following macros:

- `%a`: the IP address of the client.
- `%n`: the domain name of the client or *unknown* if not available.
- `%i`: the user ID or *unknown* if not available.
- `%s`: the matched source group or *unknown* if no groups were matched.
- `%t`: the matched destination group or *unknown* if no groups were matched.
- `%u`: the requested URL.
- `%p`: the path and the optional query string of %u but without the leading `/`.
- `%%`: a single `%`.

Now that squidGuard is configured, we can build the Berkeley DB files for domains, URLs and regular expressions with the command:

```
# squidGuard -u -C all
# chown -R _squid /var/squidguard/
```

You can test that squidGuard configuration is working properly by simulating some Squid requests from the command line; squidGuard expects a single line on stdin with the following format (empty fields are replaced with `-`):

```
URL client_ip/fqdn user method urlgroup
```

and returns the configured redirect URL (if the site is blocked) or an empty line; for example: see Listing 21.

If everything is working as expected, we can configure Squid to use squidGuard as the redirector, by editing a few parameters in the `/etc/squid/squid.conf` file see Listing 22.

Virus scanning with SquidClamav

SquidClamav (*http://www.darold.net/projects/squidclamav/*) is a *ClamAV antivirus redirector for Squid*. It will help us filter out malicious software from web traffic.

Installation

We already covered the installation procedure of the *Clam AntiVirus* (*http://www.clamav.net*) in a previous document (*http://www.kernel-panic.it/openbsd/mail/mail6.html#mail-6.2*), so we won't dwell on this topic now and proceed directly to the installation of SquidClamav. We will assume that ClamAV resides on the same machine as Squid, though you may wish to create a separate antivirus server, possibly serving both the cache and the mail server.

SquidClamav relies on the cURL (*http://curl.haxx.se/*) library to download the files to scan, so we need to add the following packages first:

- `libiconv-x.x.tgz`
- `gettext-x.x.x.tgz`
- `libidn-x.x.tgz`
- `curl-x.xx.x.tgz`

Then we can download (*http://www.darold.net/projects/squidclamav/*), extract and compile the SquidClamav tarball: see Listing 23.

Configuration

The configuration file is `/etc/squidclamav.conf`. SquidClamav can be configured to scan or ignore requests based on regular expressions. The `regex` and `regexi` keywords allow you to specify the files you want to scan (the former is case-sensitive while the latter is not). E.g: see Listing 24.

Listing 20. SquidGuard Configuration

```
acl {
    admin within workhours {
        # The following rule allows everything except porn, drugs and
        # gambling sites during work hours. '!' is the NOT operator.
        pass !porn !drugs !gambling all
    } else {
        # Outside of work hours drugs and gambling sites are still blocked.
        pass !drugs !gambling all
    }
    lan {
        # The built-in 'in-addr' destination group matches any IP address.
        pass !in-addr !porn !drugs !gambling all
    }
    default {
        # Default deny to reject unknown clients
        pass none
        redirect  http://www.kernel-panic.it/error.html&ip=%a&url=%u
    }
}
```

Listing 21. SquidGuard Configuration

```
# echo „http://www.blocked.site 1.2.3.4/- user GET -” | squidGuard \
> -c /etc/squidguard/squidguard.conf -d
[ ... ]
2008-12-14 09:57:04 [27349] squidGuard ready for requests (1197622624.065)
http://www.kernel-panic.it/error.html&ip=1.2.3.4&url=http://www.blocked.site
1.2.3.4/- user GET
2008-12-14 09:57:04 [27349] squidGuard stopped (1197622624.067)
# echo „http://www.good.site 1.2.3.4/- user GET -” | squidGuard \
> -c /etc/squidguard/squidguard.conf -d
[ ... ]
2008-12-14 10:30:24 [12046] squidGuard ready for requests (1197624624.421)

2008-12-14 10:30:24 [12046] squidGuard stopped (1197624624.423)
```

Listing 22. SquidGuard Configuration

```
/etc/squid/squid.conf
# Path to the redirector program
url_rewrite_program   /usr/local/bin/squidGuard

# Number of redirector processes to spawn
url_rewrite_children  5

# To prevent loops, don't send requests from localhost to the redirector
url_rewrite_access    deny  localhost
and reload Squid configuration:
# squid -k reconfigure
```

The `abort` and `aborti` keywords, `instead`, tell SquidClamav to skip checking files matching specific paterns. You may also use the `whitelist` keyword to ignore a given URL or domain.E.g.: see Listing 25.

The `content` keyword allows virus scanning based on the request content type. E.g.:

```
# Scan all files with a media type of
"application"
content  ^.*application\/.*$
```

Listing 26 is a sample configuration file.

As you can see, the `squidguard` parameter allows you to chain SquidClamav with another redirector, typically `squidGuard`; the chained program is called before the antivirus scanner.

Now we only have to modify the value of the `url_rewrite_program` parameter in Squid's configuration file:

```
/etc/squid/squid.conf
url_rewrite_program  /usr/local/bin/
squidclamav
```

and reload Squid.

```
# squid -k reconfigure
```

Note: to scan a file, SquidClamav needs to download it first; so make sure your Squid ACLs allow localhost to access the web:

```
/etc/squid/squid.conf
http_access allow localhost
```

You can check that everything is working fine by trying to download the Eicar (*http://eicar.org/anti_virus_test_file.htm*) anti-virus test file. In the log file, you should get something like: see Listing 27.

Listing 23. Virus Scanning with SquidClamav

```
$ tar -zxvf squidclamav-x.x.tar.gz
[...]
$ cd squidclamav-x.x
$ env LDFLAGS=-L/usr/local/lib/ CPPFLAGS=-I/usr/local/include/ ./configure
[...]
$ make
[...]
$ su
Password:
$ make install
[ ... ]
# cp squidclamav.conf.dist /etc/squidclamav.conf
# touch /var/log/squidclamav.log
# chown _squid /var/log/squidclamav.log
```

Listing 24. Virus Scanning with SquidClamav

```
# Check against the ClamAV antivirus all files with case insensitive
# extension .exe, .com or .zip
regexi    ^.*\.exe$
regexi    ^.*\.com$
regexi    ^.*\.zip$
```

Listing 25. Virus Scanning with SquidClamav

```
# Don't virus scan .gif, .png and .jpg images and .html and .htm documents
aborti    ^.*\.gif$
aborti    ^.*\.png$
aborti    ^.*\.jpg$
abort     ^.*\.html$
abort     ^.*\.htm$

# Don't virus scan trusted web sites
whitelist www.kernel-panic.it
```

Ad Zapping with AdZapper

AdZapper (*http://adzapper.sourceforge.net/*) is a *redirector for squid that intercepts advertising (banners, popup windows, flash animations, etc), page counters and some web bugs (as found).* It will help users to get rid of those annoying popup windows, flash animations and malicious cookies and will help you save bandwidth and cache resources.

We will make use of three scripts:

- `squid_redirect`, which performs the actual ad zapping;
- `zapchain`, which chains multiple redirectors together (this is necessary because Squid accepts only one `redirector_program`);
- `wrapzap` which is a very simple wrapper script that sets environment variables useful to the redirector and then runs it.

Installation

The installation procedure is very simple. Download (*http://adzapper.sourceforge.net/#download*) and extract the tarball, then copy the `squid_redirect`, `wrapzap` and `zapchain` scripts to `/usr/local/bin`, or wherever you prefer.

```
# tar -zxvf adzap-xxxxxxxx.tar.gz
[...]
# cd adzap-xxxxxxxx/scripts
# cp squid_redirect wrapzap zapchain
/usr/local/bin/
```

The `zaps` directory contains the images that will replace the zapped ads: copy them to where the web server can find them. They're not really works of art, so feel free to customize them.

```
# scp -r ../zaps root@www.kernel-
panic.it:/var/www/icons/
```

Configuration

AdZapper configuration takes place in the `wrapzap` script; below is a sample configuration script: see Listing 28.

Now we only have to update the `url_rewrite_program` in Squid's configuration file:

Listing 26. Virus Scanning with SquidClamav

```
/etc/squidclamav.conf
# IP address and port of the Squid proxy
squid_ip      127.0.0.1
squid_port    3128

# Path to the log file
logfile       /var/log/squidclamav.log

# URL where to redirect a request when a virus is found. SquidClamav will
# append the original URL and the virus name to this URL.
redirect      http://www.kernel-panic.it/viruswarn.php

# Disable virus scanning if the requested file hits squid cache
trust_cache   1

# Timeout when downloading files
timeout       60

# Set this to '1' for more verbose logging
debug         0

# Set this to '1' to force virus scan of URLs whose content-type can't be
# determined by libcurl
force         1

# Set this to '1' to show time statistics of URL processing
stat          0

# Don't follow more than 10 redirects
maxredir      10
# Uncomment to make cURL pretend to be Internet Explorer
#useragent     Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
# IP address and port of the clamd daemon
clamd_ip      127.0.0.1
clamd_port    3310
# Uncomment if you're using the unix socket to communicate with clamd
#clamd_local   /tmp/clamd
# Check rules
aborti        ^.*\/cgi-bin\/.*$
aborti        ^.*\.pdf$
aborti        ^.*\.html$
aborti        ^.*\.css$
aborti        ^.*\.xml$
abortcontenti ^.*application\/json.*$
regexi        ^.*\.exe
regexi        ^.*\.zip
regexi        ^.*\.gz
content       ^.*application\/.*$
whitelist     www.kernel-panic.it
# Call another redirector (usually squidGuard) before the antivirus scanner
squidguard    /usr/local/bin/squidGuard
```

Listing 27. Virus Scanning with SquidClamav

```
/var/log/squidclamav.log
[...]
Fri May 15 19:26:49 2009 [29028] DEBUG received from Clamd: stream: Eicar-Test-Signature FOUND
Fri May 15 19:26:49 2009 [29028] LOG Redirecting URL to: http://www.kernel-panic.it/viruswarn.php?url=http://
www.eicar.org/download/eicar.com.txt&source=192.168.1.14/-&user=-&virus=stream:+Eicar-Test-Signature+FOUND
Fri May 15 19:26:49 2009 [29028] DEBUG End reading clamd scan result.
Fri May 15 19:26:49 2009 [29028] DEBUG Virus found send redirection to Squid.
```

Listing 28a. Ad Zapping wit AdZapper. Configuration

```
/usr/local/bin/wrapzap
#!/bin/sh

squidclamav=/usr/local/bin/squidclamav
zapper=/usr/local/bin/squid_redirect

# Setting ZAP_MODE to „CLEAR” will cause the zapper to use transparent images,
# thus completely hiding ads. This may, however, hide useful markup.
ZAP_MODE=

# Base URL of the directory containing the replacement images
ZAP_BASE=http://www.kernel-panic.it/icons/zaps
ZAP_BASE_SSL=https://www.kernel-panic.it/icons/zaps

# The following variables contain the path to extra pattern files.
# ZAP_PREMATCH patterns are consulted before the main pattern list. Use it to
# prevent overzapping by some erroneous patterns in the main pattern file.
ZAP_PREMATCH=

# ZAP_POSTMATCH patterns are consulted after the main pattern list. Use it to
# add extra patterns
ZAP_POSTMATCH=

# ZAP_MATCH patterns are consulted instead of the main pattern list. Use it to
# fully customize AdZapper
ZAP_MATCH=

# Should you use Apache2 instead of Squid, set this to „NULL”
ZAP_NO_CHANGE=

# Placeholder images names. „Clear” versions have „-clear” appended to the root
# portion of the file name; e.g. „ad.gif” becomes „ad-clear.gif”.
STUBURL_AD=$ZAP_BASE/ad.gif
STUBURL_ADSSL=$ZAP_BASE_SSL/ad.gif
STUBURL_ADBG=$ZAP_BASE/adbg.gif
STUBURL_ADJS=$ZAP_BASE/no-op.js
STUBURL_ADJSTEXT=
STUBURL_ADHTML=$ZAP_BASE/no-op.html
STUBURL_ADHTMLTEXT=
STUBURL_ADMP3=$ZAP_BASE/ad.mp3
STUBURL_ADPOPUP=$ZAP_BASE/closepopup.html
```

Listing 28b. Ad Zapping wit AdZapper. Configuration

```
STUBURL_ADSWF=$ZAP_BASE/ad.swf
STUBURL_COUNTER=$ZAP_BASE/counter.gif
STUBURL_COUNTERJS=$ZAP_BASE/no-op-counter.js
STUBURL_COUNTERHTML=$ZAP_BASE/no-op-counter.html
STUBURL_WEBBUG=$ZAP_BASE/webbug.gif
STUBURL_WEBBUGJS=$ZAP_BASE/webbug.js
STUBURL_WEBBUGHTML=$ZAP_BASE/webbug.html

# Set this to „1” to use the rewrite facility to get the printer-friendly
# version of some pages
STUBURL_PRINT=

export ZAP_MODE ZAP_BASE ZAP_BASE_SSL ZAP_PREMATCH ZAP_POSTMATCH ZAP_MATCH ZAP_NO_CHANGE
export STUBURL_AD STUBURL_ADSSL STUBURL_ADJS STUBURL_ADHTML STUBURL_ADMP3      \
        STUBURL_ADPOPUP STUBURL_ADSWF STUBURL_COUNTER STUBURL_COUNTERJS        \
        STUBURL_COUNTERHTML STUBURL_WEBBUG STUBURL_WEBBUGJS STUBURL_WEBBUGHTML \
        STUBURL_PRINT STUBURL_ADHTMLTEXT STUBURL_ADJSTEXT


# Exec the real zapper (chained with SquidClamav)
exec /usr/local/bin/zapchain „$zapper” „$squidclamav”
```

Listing 29. Appendix. Server-side Configuration

```
remote# pkg_add stunnel-x.xx.tgz
[...]
remote# openssl req -x509 -newkey rsa:2048 -keyout /etc/ssl/private/stunnel.key \
> -out /etc/ssl/stunnel.crt -days 365 -nodes
[...]
remote# chmod 600 /etc/ssl/private/stunnel.key
```

Listing 30. Appendix. Server-side Configuration

```
/etc/stunnel/stunnel.conf
cert = /etc/ssl/stunnel.crt
key = /etc/ssl/private/stunnel.key

chroot = /var/stunnel/
setuid = _stunnel
setgid = _stunnel
pid = /var/run/stunnel.pid

socket = l:TCP_NODELAY=1
socket = r:TCP_NODELAY=1

[https]
accept  = 443
connect = 22
TIMEOUTclose = 0
```

```
/etc/squid/squid.conf
redirect_program  /usr/local/bin/
wrapzap
```

and reload Squid.

```
# squid -k reconfigure
```

Now ads should magically disappear from web sites!

Appendix

Tunneling through Squid

So you have finally configured your proxy server, allowing only requests to a few standard ports, blocking blacklisted sites, ads and viruses. The HTTP `CONNECT` method is restricted to the standard HTTPS port. Your LAN firewalls rules are very strict and block everything but requests to port 3128 of the proxy. Therefore, you feel pretty confident that users won't be able to do anything on the Internet you didn't explicitly allow.

But Squid is an ugly beast, and if you don't pay very close attention to its configuration (and log files), your users could end up getting around most of your blocking rules. Let's have a look at a practical example.

Stunnel (*http://www.stunnel.org/*) is a program that allows you to encrypt arbitrary TCP connections inside SSL. It is mainly used to secure non-SSL aware daemons and protocols (like POP, IMAP, LDAP, etc) by having Stunnel provide the encryption, requiring no changes to the daemon's code.

Listing 31. Appendix. Client-side Configuration

```
local$ tar -zxvf stunnel-4.05.tar.gz
[...]
local$ patch -p0 < connect-proxy.mwald.patch
[...]
local$ cd stunnel-4.05
local$ ./configure
[...]
local$ ln -s /usr/sbin/openssl /usr/bin/openssl
local$ make
[...]
local$ su
Password:
local# make install
[...]
local#
```

Listing 32. Appendix. Client-side Configuration

```
/etc/stunnel/stunnel.conf
chroot = /var/stunnel
setuid = _stunnel
setgid = _stunnel
pid = /var/run/stunnel.pid

client = yes

[https]
accept = 1443
connect = web-proxy:3128
httpsproxy_dest = stunnel-server:443
httpsproxy_auth = username:password
```

Basically, Stunnel establishes an encrypted and persistent connection between two separate machines. One machine acts as the server and forwards any connection Stunnel receives to a user-defined port. The other machine acts as the client, binding to an arbitrary port and forwarding any connection it receives on that port to the server machine.

We will use Stunnel and Squid to bypass firewall rules and `ssh(1)` to a remote server (e.g. your home computer) from a local computer in the corporate LAN. The OpenBSD ports and packages archives include a few similar tools for tunneling network traffic through proxy servers, such as:

- Corkscrew (*http://www.agroman.net/corkscrew/*), a tool for tunneling `ssh(1)` through HTTP proxies;
- gotthard (*http://www.nazgul.ch/dev.html*), a daemon which tunnels `ssh(1)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=ftp&sektion=1*)sessions through an HTTPS proxy;
- httptunnel (*http://www.nocrew.org/software/httptunnel/*), which creates a bidirectional virtual data connection tunnelled in HTTP requests.

However, Stunnel is probably the most versatile and comprehensive tunneling solution, since it can forward any type of network traffic (not only `ssh(1)` *http://www.openbsd.org/cgi-bin/man.cgi?query=ftp&sektion=1*) and provides an additional SSL cryptography layer, thus protecting clear text protocols such as `telnet(1)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=telnet&sektion=1*) or `ftp(1)` *http://www.openbsd.org/cgi-bin/man.cgi?query=ftp&sektion=1*.

Server-side configuration

The remote computer will necessarily have to act as the server. Install stunnel (*http://www.stunnel.org/*) from the packages and create the SSL certificate: see Listing 29.

Then configure Stunnel to bind to port 443 (HTTPS) and forward incoming connections to port 22 (ssh). The configuration file is `/etc/stunnel/stunnel.conf`: see Listing 30.

References

- OpenBSD *http://www.openbsd.org/*, the secure by default operating system
- Squid *http://www.squid-cache.org/*, a full-featured Web proxy cache designed to run on Unix systems
- Squidguard *http://www.squidguard.org/*, an ultrafast and free filter, redirector and access controller for Squid
- ClamAV *http://www.clamav.net/*, a GPL anti-virus toolkit for UNIX
- SquidClamav *http://www.darold.net/projects/squidclamav/*, a Clamav Antivirus Redirector for Squid
- AdZapper *http://adzapper.sourceforge.net/*, a redirector for squid that intercepts advertising, page counters and some web bugs
- DansGuardian *http://dansguardian.org/*, true web content filtering for all
- Stunnel *http://www.stunnel.org/*, the universal SSL wrapper
- HTTP Connect-style proxy patch for Stunnel *http://www.stunnel.org/patches/desc/connect-proxy.mwald.html*
- Corkscrew *http://www.agroman.net/corkscrew/*, a tool for tunneling ssh(1) *http://www.openbsd.org/cgi-bin/man.cgi?query=ssh&sektion=1* through HTTP proxies
- gotthard *http://www.nazgul.ch/dev.html*, a daemon which tunnels ssh(1) *http://www.openbsd.org/cgi-bin/man.cgi?query=ssh&sektion=1* sessions through an HTTPS proxy
- httptunnel *http://www.nocrew.org/software/httptunnel/*, a tool for creating a bidirectional virtual data connection tunnelled in HTTP requests

Bibliography

- The Squid Documentation Project *http://squid-docs.sourceforge.net/*
- Squid Frequently Asked Questions *http://old.squid-cache.org/Doc/FAQ/FAQ.html*
- Squid Wiki *http://wiki.squid-cache.org/*
- Squid configuration manual *http://www.visolve.com/squid/index.htm*
- Squid-Book oltre le FAQ (Italian only) *http://www.merlinobbs.net/Squid-Book/HTML/index.html*
- Configuring squidGuard *http://www.squidguard.org/config/*
- Meeting the Challenges of Web Content Filtering *http://www.squidguard.org/config/*

Now we can start it and go to work to have some fun with our tunnel:

```
remote# /usr/local/sbin/stunnel
```

Client-side configuration

So now we come to the local computer, which will act as the client. The SSL tunnel needs to go through Squid to get around the firewall rules but, by default, Stunnel doesn't support web proxies. Fortunately, a few patches are available that add SSL-proxy support to Stunnel. The most recent available (*http://www.stunnel.org/patches/desc/connect-proxy.mwald.html*) applies to Stunnel version 4.05, so I suggest that you download (*http://ftp.bit.nl/mirror/stunnel/obsolete/4.x/*) and install this version on the client machine see Listing 31. The patch introduces two additional configuration parameters: `httpsproxy_dest` (name or address of the Stunnel server) and `httpsproxy_auth` (proxy authentication credentials). We will configure the client to accept connections on an arbitrary port (e.g. 1443) and forward them to port 443 of the remote Stunnel server (which, in turn, will forward them to port 22). In other words, when you will connect to port 1443 on the local computer, you will actually get connected to port 22 on the remote computer.

The client configuration file looks like: see Listing 32.

Ok, everything is ready, let's give it a try:

```
local# /usr/local/sbin/stunnel
local# ssh localhost -p 1443
root@localhost's password:
remote#
```

As you can see, despite firewall rules and Squid ACLs, we have successfully connected to the remote computer. Once the tunnel is up, you could even do the opposite and connect from the remote server to the local client by simply opening a reverse ssh from the local client:

```
local# ssh -NR 2443:localhost:22 -p
1443
```

This way, every connection received by the remote server on port 2443 will be forwarded to port 22 of the local client:

```
remote# ssh localhost -p 2443
root@localhost's password:
local#
```

You could even allow X11 forwarding on the remote server and have your whole remote graphical environment available on the local machine (for instance to surf the web with no proxy filters).

Anyway, this paragraph only meant to point out how much careful Squid configuration must be. Usually, however, the stricter your corporate policy, the more determined your users will be to evade it.

By the way, using whitelists is probably the best solution to prevent tunneling, but, if they are too restrictive, get ready to get your car keyed by a crowd of angry users!

About the Author

Daniele Mazzocchio a Unix system administrator from Italy, working for a major telco where he manages HP/UX, Solaris, Tru64 and Linux machines. He is a BSD user since 2003 and maintain the www.kernel-panic.it website, which contains various documents on OpenBSD. He also maintains BowlFish, a script for installing OpenBSD on embedded devices, and he is currently working on py-PF, a Python module for managing Packet Filter.

Hosting Environment Network and Firewall Redundancy with the BSDs

Chris Buechler

With many large websites and hosting providers relying on BSD operating systems to power their businesses, it only makes sense that many smaller providers take the same path.

In these smaller environments, BSD systems are also frequently relied upon to perform all of the routing, firewalling and load balancing for the environment. This article covers the network and firewall redundancy and load balancing options available with the BSDs, from the author's experience implementing such solutions in numerous environments around the world ranging from a partial rack to a few dozen rack cabinets.

Overview

As a summary of the type of environment being discussed here, this is generally where you are renting a small portion of a large colocation facility, and building a hosting environment. This can be to offer hosting services to customers, or for your company's own web sites, email and other services. The colocation provider brings a network drop or two into your cabinet, and the remaining infrastructure is up to you to design and implement. This article is about the process of designing that infrastructure and where various capabilities of the BSD operating systems can be employed to create a highly available network while minimizing cost.

For public IP assignments from the provider, unless you are working with a very small environment, you will usually get two IP blocks. One /29 or /30 block for the WAN side of your firewall(s), and a larger block that gets routed to an IP in your WAN block for use inside the firewall. This will be covered further in the next section.

Various levels of redundancy in each portion of the network will be discussed. The level of which to use in your environment will depend on your availability requirements, and budget. Some of the practices to enhance redundancy require doubling of equipment such as firewalls and switches, and while this is typically a small portion of the overall budget to build and maintain such an environment, budgetary constraints may guide your design decisions and lead to sacrifices in certain areas.

Also note that being the co-founder of the pfSense project and it being the primary firewall I help deploy, I have a considerable bias on the firewall side towards PF, CARP and pfsync. Depending on your BSD of choice, there are other alternatives, though PF and friends will be the focus of this article on the firewall side.

Network Perimeter

The edge of your network in these environments will usually be your firewall, or in my preferred deployment, a pair of redundant firewalls running PF, CARP and pfsync. Two network drops from the provider, one into each firewall's WAN port, protects you from switch port failure or cabling issues on the provider's side of the network. In the case of redundant firewalls, a /29 subnet from the provider is most commonly used on the WAN side, with one IP for each firewall's WAN interface, one for the provider's router on that segment, and three remaining to be assigned as shared CARP IPs. The CARP IPs are shared between the two firewalls, with only the firewall having master status answering on those IPs. The CARP IPs can be used in combination with NAT with systems behind the firewall. In environments large enough to justify more IP space, a second public IP subnet is usually assigned. The second subnet, typically a /28 or larger, is routed by the provider to one of your CARP IPs so it will always go to the firewall currently holding master status. This second public IP subnet is usable in a number of ways. Some people prefer to assign public IPs directly to systems, and configure an internal firewall interface with the second public IP subnet. If you prefer using strictly NAT, you can use that subnet with NAT on the firewall as well. In some environments you use a mix of both, with the public IP subnet configured on an internal interface, but

some of the IPs in that subnet used by the firewall for NAT rather than being directly assigned.

In addition to the connection to the provider network, and one or more connections to the internal network, the firewalls have a connection between them for pfsync traffic. pfsync synchronizes the state table between the firewalls, allowing for failover while retaining all active connections. The pfsync traffic does not require a dedicated interface, however it is recommended for security and performance reasons.

The following diagram illustrates the basic layout of the firewall setup described Figure 1.

Internal Network

With the network perimeter defined, this section covers the network devices connected inside the firewalls. For switch redundancy, usually a minimum of two internal switches are used, with one firewall plugged into each switch. Port bonding, as discussed later in this article, can be used as well to connect both firewalls to both internal switches for additional redundancy. However, because the number of interfaces on the firewalls may be limited because of the hardware platform in use or the expense of multi-port NICs, and the fact that you already have switch redundancy by being able to fail over to the secondary firewall, frequently port bonding is not configured on the firewalls.

A VLAN trunk is used in most environments as the internal network interface of the firewalls. In combination with 802.1Q VLAN capable switches, this allows the firewall to carry numerous internal networks over one physical NIC. The switch ports are configured as members of the appropriate VLAN for the attached device, with the ports to the firewalls and those connecting a switch to another switch configured with tagged VLANs. VLAN interfaces are then configured on the firewalls, which are functionally equivalent to adding another physical interface.

The following diagram illustrates a typical VLAN deployment. The green lines indicate tagged VLAN trunks carrying all VLANs. The blue and orange lines connecting the two servers shown are on separate VLANs, which is functionally equivalent to having them plugged into

References

Firewalls

- PF – *http://www.openbsd.org/faq/pf/*

Books

- The following books on PF are available from Amazon and many other booksellers.
- The Book of PF: A No-Nonsense Guide to the OpenBSD Firewall
- The OpenBSD PF Packet Filter Book
- Building Firewalls with OpenBSD and PF

pfSense

- Home – *http://www.pfsense.org*
- Documentation – *http://doc.pfsense.org*
- Forum – *http://forum.pfsense.org*
- Book – pfSense: The Definitive Guide – *http://pfsense.org/book*

Port Bonding

- FreeBSD lagg(4) – *http://www.freebsd.org/cgi/man.cgi?query=lagg&apropos=0&sektion=0&manpath=FreeBSD+8.0-RELEASE&format=html*
  http://www.freebsd.org/doc/handbook/network-aggregation.html
- NetBSD agr(4) – *http://netbsd.gw.com/cgi-bin/man-cgi?agr*
- OpenBSD trunk(4) – *http://www.openbsd.org/cgi-bin/man.cgi?query=trunk&sektion=4*

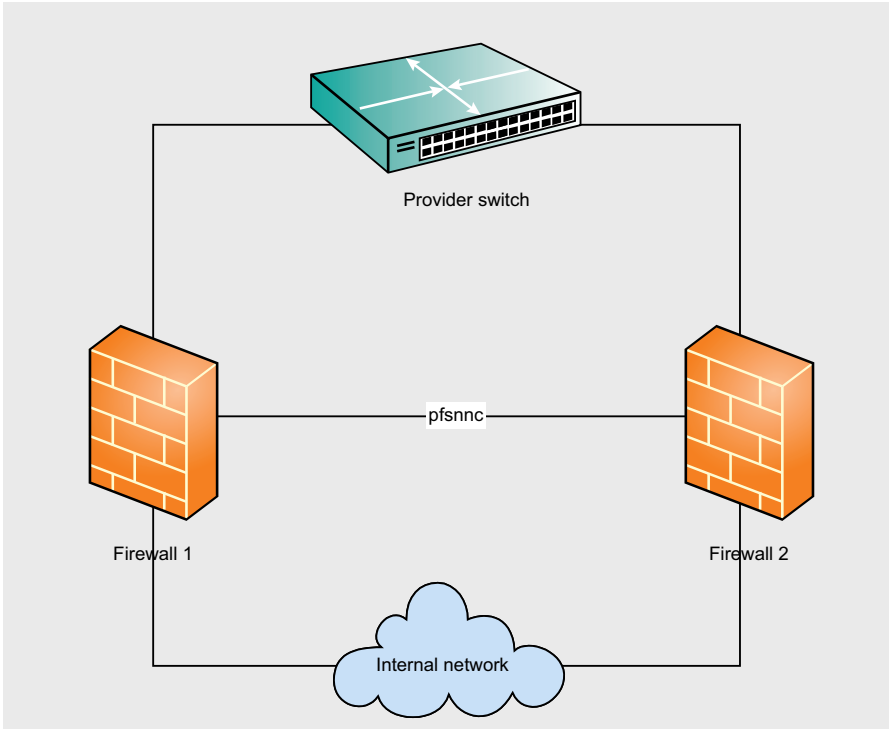


Figure 1. The illustration the basic layout of the firewall setup

two different switches. All communication between the VLANs must be routed, usually by the firewall in networks such as this (Figure 2).

References

Load Balancing

- haproxy – *http://haproxy.1wt.eu/*
- nginx – *http://nginx.org/*
- pound – *http://www.apsis.ch/pound/*
- relayd – *http://www.openbsd.org/cgi-bin/man.cgi?query=relayd&sektion=8&format=html*
- slbd – *http://slbd.sourceforge.net/*
- Varnish – *http://varnish-cache.org/*

The built in server load balancer in pfSense is covered in pfSense: The Definitive Guide. *http://pfsense.org/book*

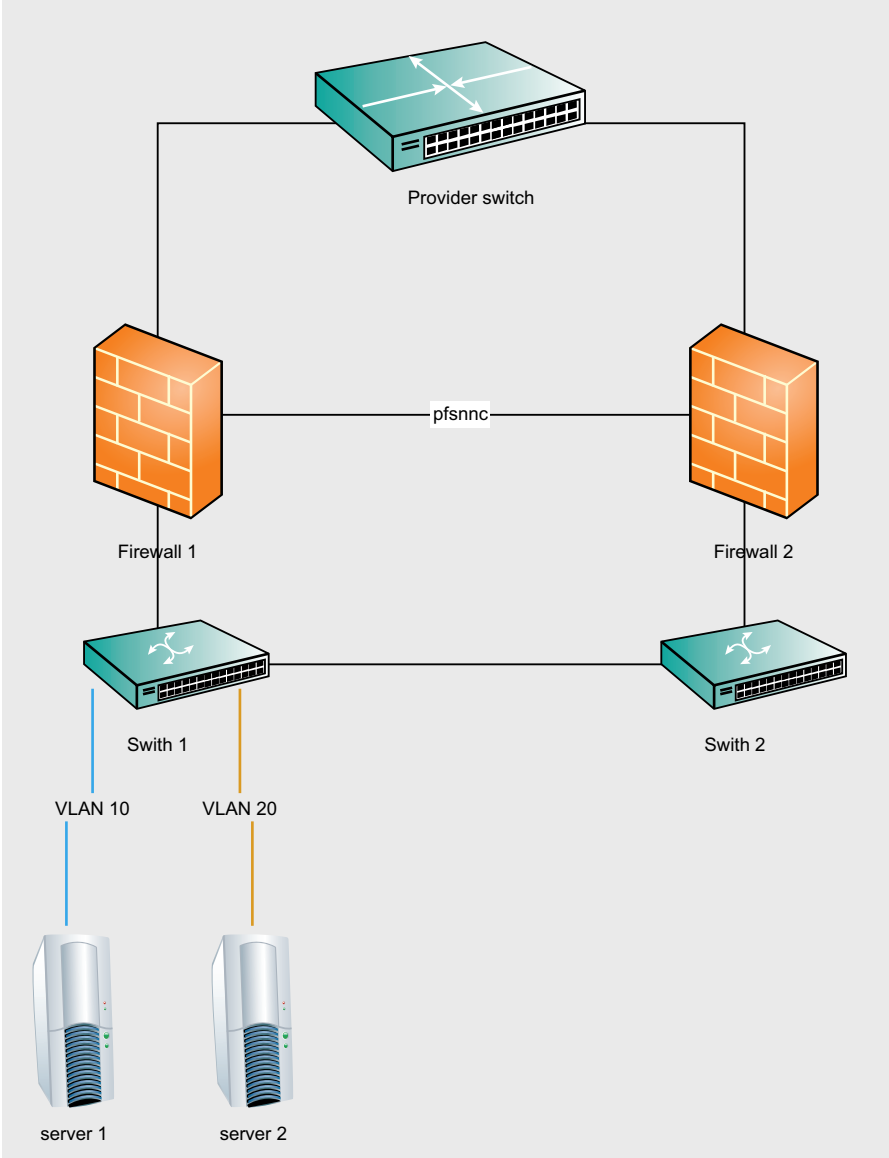


Figure 2. A firewall in networks for routing all communication between the VLANs

Server Network Connectivity Redundancy

Now with the perimeter and routing redundancy handled, the servers can also be accommodated. The usual one NIC per server leaves the servers susceptible to NIC, switch or switch port failures. Each of the three most widespread BSDs offers a solution here, with port bonding – lagg(4) on FreeBSD, agr(4) on NetBSD, and trunk(4) on OpenBSD. Sorry DragonFly BSD fans – while ng_one2many(4) provides somewhat similar functionality, it's not truly comparable to lagg/agr/trunk and DragonFly lacks anything equivalent to those three.

Port bonding enables you to combine multiple physical NICs into a single logical NIC. Depending on the configuration type chosen, this may provide only redundancy in case of NIC or switch failure, or increased bandwidth as well as redundancy. The failover mode of lagg and trunk allow for sending traffic only over the primary NIC, and failing over to the secondary NIC if the first loses its Ethernet link. The loadbalance and roundrobin modes balance outgoing traffic across all active interfaces, and disable interfaces if they lose link. With all of these modes, if the NIC has a link light, the interface is considered up. That may not always be the case, so these methods are more limited in their ability to detect failures.

The 802.3ad *Link Aggregation Control Protocol* (LACP) standard provides a way for the servers and switches to detect each other and bond the ports appropriately. This has the advantage of being able to detect failures that the other modes will not, since LACP requires the connected device to communicate successfully with the switch, while the others simply check the NIC's link state. The downside to LACP is it generally is not supported across switches on low end to mid range equipment, so it usually leaves you stuck using a single switch.

While named differently, the implementations in FreeBSD and OpenBSD are essentially the same as lagg(4) in FreeBSD is a port of trunk(4) in OpenBSD. NetBSD's agr(4) is more limited than lagg and trunk, as it only supports the LACP standard.

For purposes of switch redundancy, because of the limits of LACP across switches, the failover, loadbalance or roundrobin modes of lagg or trunk have most frequently been chosen in the environments where I have assisted with the design.

The following diagram illustrates the above infrastructure, with server network redundancy added in (Figure 3).

Load Balancing

With network and firewall redundancy covered, there is a remaining gap with service availability. If a web server or other service has died or otherwise malfunctioned, there are load balancing options to direct traffic to a different server. In other instances where a specific application is resource intensive on the server, the load can be distributed to multiple internal servers, automatically removing failed servers from the pool.

There are a number of options for load balancing, with varying capabilities. In smaller networks, many times the load balancing functionality is deployed on the firewalls. Two options that integrate into PF are slbd and relayd. slbd is deprecated, so for new deployments on a stock BSD, you'll likely want to use relayd instead. pfSense 1.2.x provides a load balancing GUI for slbd, which works well, but is more limited in its ability to detect failures. If the load balancer can connect to the server on the service's port, it is considered up, which may not always be the case. relayd provides enhanced service checking capabilities. pfSense 2.0, slated for stable release sometime in 2010, replaces slbd with a GUI-configured relayd.

The options that integrate into PF lack some of the more advanced capabilities provided by other load balancing services. Four of the most commonly used alternatives are haproxy, nginx, pound, and Varnish. They each provide somewhat different capabilities, and the best fit for your environment will depend on your specific needs. Review the links provided in the references section of this article to determine which best fits your environment. For a GUI-managed option, pfSense offers a haproxy package.

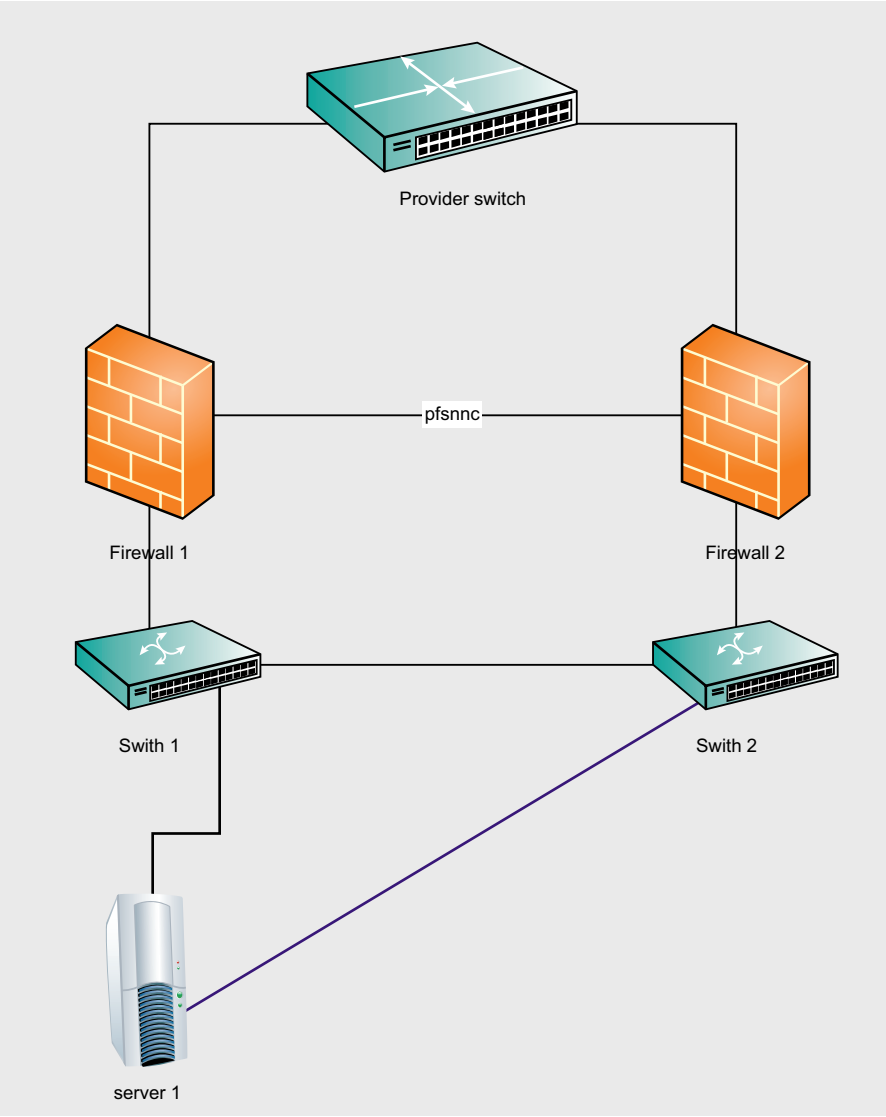


Figure 3. The illustration the above infrastructure, with server network redundancy added in

Summary

The BSDs and related tools are a great fit and proven solutions for building a highly reliable and redundant network infrastructure for hosting environments. While specific configuration examples are beyond the scope of this article, hopefully the content here and references provided will help you evaluate the solutions available and design a solid infrastructure for your hosting environment.

About the Author

Chris Buechler is the co-founder of the pfSense open source firewall distribution, and Chief Technology Officer of BSD Perimeter LLC, the corporate arm of the pfSense project. His most recent book is pfSense: The Definitive Guide. Chris and the rest of the pfSense team provide a variety of network and security services primarily related to BSD systems via https://portal.pfsense.org. Chris can be reached at cmb@pfsense.org. Thanks to Jim Pingle (jimp@pfsense.org), the co-author of pfSense: The Definitive Guide, for reviewing this article.

Comparison of FreeBSD And OpenBSD:
Not One Cake But The Two Ones

Juraj Sipos

The purpose of this article is to highlight some differences between the two BSD operating systems – FreeBSD and OpenBSD.

It is because there is a significant lack of such information, as BSD systems somewhat keep hidden in seclusion. To help readers understand what the term BSD means, some terminological and historical aspects are presented too.

There are several types of BSD Unix systems such as FreeBSD, OpenBSD, NetBSD, and few other ones, too. These BSD's, however, do not differ from one another in the Linux-like fashion. Every BSD system is a cake made with different ingredients and thus with its own taste.

Linux has one skeleton only – all its distributions use the same kernel; this OS is comparable to a cake made with identical ingredients but with different fruits on its top.

BSD systems are enveloped with myths some people believe are true. A guy in a BSD forum once said that Linux packages intended for one Linux distribution cannot be always easily installed in other Linux distros, which is the same complication as with OpenBSD packages, for example, which you cannot install in FreeBSD, and that it is thus unfair to compare Linux and BSD systems by criticizing bad installability of Linux packages in a Linux these packages are not intended for (end of the myth).

To understand the differences between OpenBSD and FreeBSD, we must also get some picture about differences between BSD systems and Linux. In the light of this, I have to disprove the myth outlined in the above statement, because some people think that BSD systems are like Linux. They are not. If we look at what now the word BSD means, it is, above all, the abbreviation of Berkeley Software Distribution and presently this term only identifies the family of operating systems with common history, the same way of handling particular tasks (like compilation of a kernel), or the same terminology like, for example, the *base system*, which is an installable BSD-style OS with its kernel and system utilities such as ifconfig, mount, chmod, etc., but without packages (like AbiWord, MPlayer, etc.). There is not such a thing as a sole (one) BSD system.

Unlike Linux (Slackware, Ubuntu, Debian, RedHat, etc.), FreeBSD and OpenBSD differ in their base system more manifestly – whether it is the kernel or the system commands, both OS's use different source codes; in addition, some commands used for the same task are named differently, or they exist in one system only (the `sysinstall` command in FreeBSD, for example), or have the same name but both offer a little different options. We can say the same thing if we juxtapose NetBSD and BSD/OS. Anyone can say with certainty that OpenBSD is not FreeBSD and that Ubuntu, Slackware, or SuSE is Linux.

A sole member of the BSD family is OpenBSD, FreeBSD, NetBSD – not Free BSD (although such a naming convention is occasionally used). The word *Linux* is always written separately (Slackware Linux, Debian Linux, etc.).

The above-mentioned terminological convention mirrors the fact that Linux always consists of one body, which means that in its environment you will always use the same system commands – for example, `modprobe` to load modules into the kernel. In FreeBSD, you will use `kldload` and in OpenBSD `modload` – both commands, unlike `modprobe`, have a different source code. There is less compatibility between OpenBSD and FreeBSD (and between NetBSD and BSD/OS, etc.) than between Slackware and Ubuntu Linux.

OpenBSD, FreeBSD, or NetBSD are as separate islands – each with its own beach and climate. The common feature of the BSD family is its organization, too – for example, the *base system*, or the division of distributions into CURRENT (developmental and unstable version), RELEASE (for normal use), and STABLE. Every system from the BSD family is a separate operating system under its own roof.

History

Unlike Linux, the present BSD systems stem from the real Unix. Berkeley University in California controlled the development of BSD Unix.

Some very long time ago, the company AT&T (1960-1970) had started developing an operating system we today know as Unix. However, at that time the company had been prohibited from selling software, so it licensed its code to universities for a small fee. The universities continued developing this source code and exchanged patches of it, but under coordination of *Computer Science Research Group* (CSRG) at Berkeley University. Because of this, the patches received the name of BSD Unix (Berkeley Software Distribution).

After many accomplishments and troubles, in the beginning of the 1990's the BSD code went public. It was NetBSD that first slithered to light as a free OS early in 1993; then followed FreeBSD (December 1993), and finally, but much later, OpenBSD (1995). OpenBSD appeared as a Theo de Raadt's (the founding member of NetBSD) protest against the NetBSD developers.

FreeBSD vs OpenBSD

FreeBSD: is much more user-friendly for newcomers, but some criticism appears that it is too robust and therefore a little less transparent. Some features that users have been accustomed to change more frequently over time, or rather shift away from what this OS looked like before. During the development period of many years, users have come across a number of things that more visibly steered away from the historical FreeBSD coastline – for example, the kernel in older versions of FreeBSD was in the root directory, or the syntax in the kernel compilation file (*/sys/i386/conf/GENERIC*) has kept changing more dramatically then in OpenBSD over the period of many years.

OpenBSD: is slimmer; if you demand something, you will almost always succeed (fewer problems with unresolved library dependencies). After many years, this OS has remained much the same. If you download a new version of OpenBSD, it will not startle you (that some names of devices changed, for example). Ports will always install successfully. However, OpenBSD may appear a little bit more difficult for people who got accustomed to GUI administration, as users must configure everything manually (for example, by editing scripts in /etc).

FreeBSD: installation takes place in the text graphics with intelligible wizards. Today, FreeBSD also has its installation DVD (a downloadable ISO image), which contains many binary packages such as KDE, GNOME, etc.

OpenBSD: the user will burn the *install46.iso* CD image that is available on a number of FTP servers (the number *4.6* indicates the current version). Unless you buy the official OpenBSD CD's or DVD's, you must always download binary packages separately, or install them over the Internet. Contrary to FreeBSD, the *install46.iso* image contains only the base system. The installation process is in a pure text (not text graphics). If you want to manipulate your disks, you will have to deal with cylinders – a ghostly approach for most users. OpenBSD will always keep you knowing that you descended deep into the basement of the real Unix.

FreeBSD: partitioning of disks is a straightforward and easy job like a breeze. Easy partitioning is available to you even after you finished installing the system – run the `sysinstal` command (a text graphic wizard for system administration), and then choose *Do post-install configuration of FreeBSD* from the menu, then *The disk slice (PC-style partition) editor*. That's it.

OpenBSD: partitioning of disks requires more knowledge. The system does not have any central configuration tool such as `sysinstall` in FreeBSD. You must do all your system configurations manually by editing the relevant scripts (with vi or any other editor).

FreeBSD: has more applications (packages) available than OpenBSD; the number has exceeded 20,000, which you may confirm at *www.freebsd.org/ports* website. The FreeBSD ports tree is one of the biggest in the BSD world.

OpenBSD: has fewer packages and newer ones appear with delays; *OpenOffice.org* was ported some years after it had been already available in FreeBSD.

FreeBSD: has better virtualization possibilities (as the host system); you may use older versions of VMware and packages such as WIN4BSD or VirtualBox. Wine works very well, too. It smoothly runs Microsoft Office. Linux emulation also supports compatibility for 2.6.x kernels. In FreeBSD, many Linux applications run like a breeze including Skype.

OpenBSD: deployment of virtualization strategies (with OpenBSD as the host system) is thorny – you can only use Qemu. Except for Qemu, there are other emulators like DOSBox or Bochs, but Quemu and Bochs are slow and cannot compete with the power of VMware or VirtualBox. However, OpenBSD supports running FreeBSD binaries. Win32 applications run much worse under Wine than in FreeBSD. Linux emulation is excellent, but a little bit outdated (compatible with Fedora Core 4). You will run a fewer Linux applications in OpenBSD than in FreeBSD.

FreeBSD: the base system does not include software that underwent paranoic security auditing. If you want to use the Apache Web Server, you must install it from ports (packages). In FreeBSD, the OpenBSD *Packet Filtering* (PF) is available as a kernel module.

OpenBSD: the developers put a secure Apache Web Server into the base system and you do not need to install the Apache Web Server separately (from ports). OpenBSD implements other smart security policies, too.

FreeBSD: you do not have to set your terminal type after you log in; history of your shell commands is available even after you reboot your PC. If you want to implement some security policies (like swap encryption), you must configure them additionally.

OpenBSD: every time after you type your login name and password, the system will prompt you for a terminal type (xterm, vt220, etc.); history of commands is always erased after the next reboot (for security reasons); the system comes with security measures (like swap encryption,

etc.) that users do not need to employ additionally.

FreeBSD: works with kernel modules like Linux. The FreeBSD kernel with modules is in the /boot/kernel directory (modules have the **.ko* extension like, for example, `snd_driver.ko`). User modules like kqemu (used to speedup the Qemu accelerator) are placed into the /boot/modules directory (after you `kldload` them).

OpenBSD: supports modules, too, but does not use them. The kernel has the name *bsd* and is monolithic (from one piece that does not need anything more); you will always find it in the root directory (/bsd).

FreeBSD: names of devices differ from OpenBSD. For example, physical disks are referred to as `/dev/ad0`, `dev/ad1`, etc.; USB disks as `/dev/da0`, `/dev/da1`, etc., global partitions (*slices* in the BSD terminology – visible by all partitioning tools) are referred to as `/dev/ad0s1` (first partition on the first hard drive – `/dev/hda1` in Linux), `/dev/ad1s1` (first partition on the second hard drive – `/dev/hdb1` in Linux), etc.; FreeBSD partitions (visible only by FreeBSD) are referred to as `/dev/ad0s1a`, `/dev/ad0s1b`, `/dev/ad1s1e`, etc.

OpenBSD: whether you deal with global (slices) or OpenBSD partitions, OpenBSD follows one naming convention only – partitions have letters – `/dev/wd0j`, `/dev/wd0a`, etc. USB disks are referred to as `/dev/sd0`, `/dev/sd1i`, etc.

FreeBSD: in addition to excellent manual pages, this OS has a comprehensive documentation kept in the `/usr/share/doc` directory.

OpenBSD: except for searchable documentation and FAQ on the Internet (*www.openbsd.org*), it does not have such a comprehensive documentation as FreeBSD in its /usr/share/doc directory.

FreeBSD: to switch between consoles, you need to press [*Alt-F1*], [*Alt-F2*], etc. If you have the X environment running, you will get back to it (from text consoles) by pressing [*Alt-F9*]. If you want to go to a text console from your X environment, you need to press [*Ctrl-Alt-F1*] (or [*Ctrl-Alt-F2*], [*Ctrl-Alt-F3*], etc.).

OpenBSD: the user switches between text consoles always with three keys – [*Ctrl-Alt-F1*], [*Ctrl-Alt-F2*], etc. You will get back to the X Window system, if it is running, by pressing [*Ctrl-Alt-F5*].

FreeBSD: the community is more open to ideas and questions to which anybody will probably always get a response. If you have a general question, you can send an email to the addresses listed at *www.freebsd.org* website.

OpenBSD: the community (http://*www.openbsd.org*) is closed, uncommunicative; some people say that it does not like questions. To say it better, the community wants you to thoroughly read all the manual pages and will barely answer questions like how to mount a Linux partition, or how to boot your OpenBSD box into a single user mode. To get some help, the best policy for you is to read OpenBSD forums where questions are welcome.

FreeBSD: although a few tweaks are necessary to implement this (look at `<a href="http://www.freebsd.nfo.sk/bsd2.htm"> www.freebsd.nfo.sk</a>`), in version 7.1 and higher, you can use Adobe Flash 9 with native FreeBSD Internet browsers (Seamonkey, Firefox, Opera) and watch youtube.com videos.

OpenBSD: you can only use the archaic Flash 7.0 for Opera Browser, which, unfortunately, does not work very well (the video is good, but the sound disobeys).

FreeBSD: was primarily developed for the i386 platform.

OpenBSD: supports more platforms than FreeBSD (but fewer than NetBSD).

FreeBSD: is probably a little more responsive in the X environment with default settings (without tuning). In my FreeBSD box, OpenOffice.org 2.4 writer opens in 16 seconds in KDE 3 and many other apps, too, launch a little bit faster.

OpenBSD: applications launch a little slower in X in the default installation (without tuning). OpenOffice.org 2.4 writer opens in 34 seconds in KDE 3 (tested on the same computer as with FreeBSD). To speed up your OpenBSD box, you need to tweak it (disable swap encryption, for example, enlarge the number of files that can be opened, etc.).

FreeBSD: none of its features dominate – stability, security, and usability, as well as desktop or server/network deployment, stand proudly alongside each other. The latest software and drivers (WiFi, KDE4, etc.) appear in the system much sooner than in OpenBSD; thus it is possible to say that FreeBSD fits better for desktops.

OpenBSD: its priority is security; only then do follow stability, slimness, and usability. You may deploy a secure server (firewall, gateway, etc.), which is a priority for a lot of companies. The software giant Adobe Systems runs OpenBSD on its systems. OpenBSD (not packages) is thoroughly audited as the base system for security holes and some security experts use it as a honeypot (the system designed with the purpose of drawing attacks from hackers). OpenBSD has the best firewall (PF) ever to be seen in the computer world. But this OS is extremely conservative – with exception of important fixes related to security, many other things get implemented much later than in FreeBSD.

The differences and advantages summarized jointly

Although FreeBSD did not have the boot menu in the past, its greater user-friendliness is apparent already upon the first contact with it – the OS welcomes you with an intelligent boot menu, which offers you quite a few possibilities to choose from including the single user mode (useful for cases you forgot your password). With OpenBSD, many things including the information on how to start the OS in the single-user mode may, especially for beginners, appear hard to find.

If you want to choose one of these two operating systems and words such as slimness, security, or invariability (if you are conservative) are not foreign to you, than OpenBSD is for you.

If you have USB sticks, USB hard drives, standard network cards – that is, the commonly used hardware (known in the BSD world), there is nothing to fear with both systems, but before buying any hardware you must be always more careful than with Linux. Do you use OpenBSD and plan to buy a new hardware? Then you must be even more careful than with FreeBSD. FreeBSD has a

little better hardware support for i386 PC's than OpenBSD and it can be deployed anywhere where greater flexibility and more software are needed. If the software for you does not exist, you can always try Linux programs with the FreeBSD Linux emulation.

Are you more conservative or perhaps paranoid? Then it is absolutely the best decision for you to choose OpenBSD because of its purity and fewer changes in user interaction to be expected in the future. OpenBSD, even today, is very easy to install on old (legacy) computers. The Linux emulation is excellent, but in FreeBSD it is somewhat more up-to-date.

FreeBSD and OpenBSD, including NetBSD, are three different cakes. They all come from one workshop. Any IT expert will tell you that workers of this workshop were real masters!

About the Author

Juraj lives in Slovakia and works in a library in an educational institute (school of psychology). Some time in the past he was fortunate enough to travel around the world and spend a bit of time in India and Australia. Juraj's hobbies are computers, mostly Unix and also spirituality. He has also translated several books from English, for example - Zen Flesh, Zen Bones by Paul Reps. He started with FreeBSD in 1997. He wrote the Xmodmap Howto „http://tldp.org/HOWTO/Intkeyb/" In addition to computers, he is very interested in Hinduism but not really the guru side of things, but more-so freedom and self actualization. His website has more information: http://www.freebsd.nfo.sk/

Introducing Beastie to Strangers

Jesse Smith

When PC-BSD 8 first came out back in February, I installed the operating system on two of my machines and was very impressed with the new release.

It was fast, powerful, flexible and worked well with my hardware. Not only was I thrilled with the latest release from the PC-BSD team, but I wanted to share my experience with others. I had visions of an army of Beasties peacefully invading homes, public access terminals, schools and businesses. And while I felt this BSD product had earned a place on my desktop machine, I was curious to see how other people would react to it – not just people in the IT field or people who were already open source enthusiasts, but everyday Joe and Jane Users. With that in mind, I burned several copies of the PC-BSD DVD, created a short survey form and handed both items out to anyone willing to participate.

The survey asked each volunteer some questions about how comfortable they were with computers, which common operating systems they had used previously and what sort of hardware they were using.. After all, as any technical support agent can tell you, not all computers and customers are created equal. Each person was additionally asked what aspects of the system worked for them, what did not work, what their first impressions were and how they felt about some of PC-BSD's key abilities on the desktop. The software testers were then given two weeks to experiment with PC-BSD without any assistance or direct technical support aside from the project's user's manual. After two weeks, I collected the survey forms and set about finding out what everyone else thought about PC-BSD's latest offering.

A disclaimer is probably in order here: this wasn't a scientific experiment. There weren't any control groups, the participants weren't monitored and all the volunteers filled out their own forms. The information gathered to make this report was from a fairly small population size (just under twenty people) and what it represents is closer to collective anecdotes than scientific finding.

From my perspective, one of the more interesting things about the survey results I got back was seeing how people ranked their abilities with computers and how that compared with their range of experience with various operating systems. On a scale of 1-10, the lowest anyone ranked their abilities with a computer was *2*. That person claimed the only operating system they had ever used (prior to trying PC-BSD) was Microsoft Windows. In fact, about a third of the people questioned said they had only ever used Windows prior to the experiment. The highest self-ranking score was an *8*, and that person stated they had used every OS on the form (Windows, OS X, DOS, Linux, Solaris and FreeBSD).

Getting a firm idea of what hardware was being used to test PC-BSD was difficult. Some people were able to provide detailed information, stating the type of processor, CPU speed, memory and hard drive size. But about half of the respondents claimed to have no knowledge of their computer's hardware other than it was *a three year old desktop*, or that it was a second hand Dell. Of the people who did provide specific information about their computers, the lowest-end machine had a 2.3GHz processor and 1GB of RAM with a 360GB hard drive. The highest-end machine had a dual core 2GHz CPU, 4GB of memory and a 320GB hard drive. Almost all of the test machines were reported to be desktops, with two volunteers reporting they were using a laptop for the experiment.

First impressions are always important and this is where the participants seemed most uncomfortable with the operating system. As one person remarked, *It reminded me of DOS booting up*. Most operating systems, including Linux and other members of the UNIX family, have moved to graphical start-up screens and seeing plain text scroll by tended to throw people off. As another volunteer observed, *I didn't like the black screen with white text*. Fortunately, once people arrived at the desktop, they felt more comfortable. One survey response summed up the over-all impressions of the desktop nicely by commenting, there were *nice colours* and a *slight adjustment to new terms*. On the flip side, it seems hardware problems were a serious issue for a number of people – nearly a third of the participants were unable to get PC-

BSD to boot as far as the desktop, reporting their screens went blank after the boot menu but before the reaching the desktop. Among those who were able to reach the desktop, one reported being unable to get on-line as the computer's modem wasn't detected.

Though some got off to a rocky start, the volunteers who arrived at the desktop reported mostly positive results. Of the group of people who got PC-BSD to boot, everyone reported their screens being set to a suitable resolution and their hardware (such as mice, keyboards and sound cards) working. For most of the group navigation through the system was easy, though one commenter mentioned having trouble adjusting to the KDE menu layout.

Often times when users give feedback it's in the form of complaints or bug reports. While the volunteers ran into the occasional problem, they also had some very positive things to say about the features built into PC-BSD. Everyone, for instance, was very happy to learn the FreeBSD-based system came with multimedia codecs and Flash installed right out of the box. Each member of the experiment who managed to get the operating system running expressed pleasure at the wealth of applications (such as Open Office) which came with the system free of charge, as opposed to *trial-ware and other half-functioning* apps. Though most of the volunteers didn't have enough time to use and appreciate some aspects of the system, a few respondents expressed a great deal of enthusiasm for ZFS snapshots and the concept of being nearly immune to malware. For the most part, the group was silent on the topic of package management with one user finding the package manager confusing and two people expressing how they liked the *software web browser*.

As far as using PC-BSD for their day-to-day tasks (such as e-mailing, web browsing, editing documents and playing media files), the users largely felt that the operating system was a good fit, with one person mentioning trouble working with MS-Office documents. Another user mentioned the system appeared to have *wonderful features* but wasn't sure how to make use of all of them, reducing their productivity.

So, at the end of the day, how did people view PC-BSD? About a quarter of the people surveyed said they liked PC-BSD and, over all, enjoyed the experience with one person considering making a switch from their previous OS. Another quarter of the respondents (mostly made up of people who either couldn't get the system to boot or get on-line) said they weren't happy with the product. Though one disappointed volunteer expressed an interest in trying a future version if it could be made to boot on their hardware. The remainder, about half, of the group said they were interested in PC-BSD, but were not planning on using it full-time. As one person put it, they'd like to try the OS again with *my IT guy beside me* to help explain the software.

Something I found surprising, looking at the results, was there didn't seem to be any correlation between people's confidence in their ability to use computers and getting PC-BSD to work for them. People who ranked their computer skills as being low were almost as comfortable getting the OS up and running as people who ranked themselves higher. The only serious hurdle to the testers appeared to be hardware, with some participants reporting their computers would lock up during the boot process. But once a tester reached the graphical desktop, it was generally smooth sailing from there. While the sample size involved is too small to draw any concrete conclusions, these findings suggest to me that the applications and layout of PC-BSD are mature and ready for the desktop. The bottleneck to adoption appears to be with a combination of hardware drivers and inertia. The latter because even volunteers who had very good experiences with PC-BSD showed a reluctance to switch operating systems, being comfortable with their existing set up.

Another thing I took away from this experiment is PC-BSD is providing a desktop environment which makes BSD easily available to a broader audience. Even to people who haven't used a member of the UNIX family before. With its wide range of software and well-considered defaults, the latest release of PC-BSD is mixing the power of FreeBSD with the novice-friendly desktop with wonderful results.

BSDCan 2010

The Technical BSD Conference

High value. Low cost. Something for everyone.

BSDCan, a BSD conference held in Ottawa, Canada, has quickly established itself as the technical conference for people working on and with 4.4BSD based operating systems and related projects. The organizers have found a fantastic formula that appeals to a wide range of people from extreme novices to advanced developers.

BSDCan 2010 will be held on 13-14 May 2010 at University of Ottawa, and will be preceded by two days of Tutorials on 11-12 May 2010.

There will be related events (of a social nature, for the most part) on the day before and after the conference.

http://bsdcan.org/

BSDCan 2010

HAKIN9

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

WWW.HAKIN9.ORG/EN

NOW FEATURING THE INTEL® XEON® PROCESSOR 5600 SERIES

Orion II iX-N4236

Powerful *4U* Orion II Storage Series

- ✔ Outstanding Performance
- ✔ Excellent Cooling Efficiency
- ✔ Up to 24 Processing Cores with Hyper-Threading
- ✔ Up to 72TB in 4U, Unparalleled Storage Density
- ✔ Up to 432TB in 20U of Rack Space Utilizing Optional Orion II JBOD Expansion Units

To order today call: **1-800-820-BSDi**

Notable features include:

- Dual Intel® 64-Bit Socket 1366 Six-Core, Quad-Core, or Dual-Core, Intel® Xeon® Processor 5600/5500 Series
- 4U Storage Server Chassis with up to 72 TB storage capacity
- 36 x 3.5 Hot-Swap SAS/SATA HDDs (24 front side + 12 rear side)
- 1400 W (1+1) Redundant High Efficiency Power Supply (Gold level 93%+ power efficiency)
- Dual Intel® 5520 chipsets with QuickPath Interconnect (QPI) up to 6.4 GT/s
- Up to 192GB DDR3 1333/1066/800 MHz ECC Registered DIMM/24 GB Unbuffered DIMM
- 2 (x16) PCI-E 2.0, 4 (x8) PCI-E 2.0 (1 in x16 slot), 1 (x4) PCI-E (in x8 slot)
- Intel® 82576 Dual-port Gigabit Ethernet Controller

iXsystems Introduces the Orion II 4U Storage Solution

The iX-N4236 boasts energy efficient technology and maximum, high density storage capacity, creating a 4U powerhouse with superior cooling.

The Orion II has **thirty-six hot-swappable SAS/SATA drive bays**, providing 50% more storage density than its predecessor. By delivering high-end storage density within a single machine, iXsystems cuts operating costs and reduces energy requirements.

Storage sizes for the iX-N4236 are customizable, with 250GB, 500GB, 750GB, 1TB, and 2TB hard drives available. For environments requiring maximum storage capacity and efficiency, 2TB Enterprise-class drives are available from Western Digital®, Seagate®, and Hitachi. These drives feature technologies to prevent vibration damage and increase power savings, making them an excellent choice for storage-heavy deployment schemes.

The Intel® Xeon® Processor 5600 Series (Six/Quad-Core) and Intel® Xeon® Processor 5500 Series (Quad/Dual-Core) have a light energy footprint, while creating a perfect environment for intense virtualization, video streaming, and management of storage-hungry applications. Energy efficient DDR3 RAM complements the other power saving components while still providing 18 slots and up to 192GB of memory overall.

100% cooling redundancy, efficient airflow, and intelligent chassis design ensure that even under the heaviest of workloads, the Orion II remains at an optimal temperature, while still drawing less power than other servers in its class. With a 1400 W Gold Level (93%+ efficient) power supply, the entire system works together to efficiently manage power draw and heat loss.

For more information or to request a quote, visit:
http://www.iXsystems.com/Orion2

intel Xeon inside™
Powerful. Intelligent.