Interested in learning
more about security?

SANS Institute
InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Aggressive Patching and the Use of a Standard Build: An OpenBSD Example

This paper starts with a brief general discussion of the importance of a standard build and defines Aggressive Patching as a vital part of defense in depth. It then goes on to demonstrate how to implement Aggressive Patching by creating a Standard Build internet server farm and support structures that allow for automated patching and rapid deployment of hardened servers. The general part of this paper is intended for anyone in the IT field who is interested in security in depth. The more hands-on part is aimed at Syste...

Copyright SANS Institute
Author Retains Full Rights

AD
Investigate Insider Threats
Protect your company against corporate IP Theft
Learn More
SpectorSoft

Aggressive Patching and the use of a Standard Build:
An OpenBSD example

Michael Sullenszino
April 5, 2002
Practical 1.3 for GSEC Certification

Abstract

This paper starts with a brief general discussion of the importance of a standard build and defines Aggressive Patching as a
vital part of defense in depth. It then goes on to demonstrate how to implement Aggressive Patching by creating a Standard
Build internet server farm and support structures that allow for automated patching and rapid deployment of hardened servers.
The general part of this paper is intended for anyone in the IT field who is interested in security in depth. The more
hands-on part is aimed at System Administrators with some Unix background working for small to medium sized companies with an
active internet presence. The system detailed in the pages below has been successfully deployed in a small ISP and a medium
sized virtual web hosting company. My hope is that while this is OpenBSD specific, it can work as something of a general
model for small to medium sized businesses to use.

Aggressive Patching and Security Policy

The overwhelming number of security holes found on a daily basis can cripple a small IT staff. From January to April 5, 2002
we have seen 17 advisories for Microsoft related products, 42 for RedHat linux Version 7.2 (and related software) and 17 for
the OpenBSD core. Even if you only ran one of those platforms you would be patching, on average 1-3 times per week, and this
does not include many third party software packages. Meanwhile, the speed of worm and virus development and replication

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

appears to be increasing. It is more important than ever to attempt to standardize your server platforms, thereby reducing your need to track multiple platform security issues, maintaining separate procedures, separate patch levels and separate competencies. Having a host of ad hoc procedures to patch the servers, or worse, having to reinvent the wheel when a difficult patch is released can be disaterous. And under these conditions, patching is often simply not done. The ability to patch easily and rapidly needs to be built into to very design of your server farm. We can no longer ignore the complexity and time it takes to keep servers and software patched. I have seen little in the literature on practical ways to do this effectively. Maximum Security Linux(Anonymous 2000) for instance, in its section on Malware does not even mention internet worms, nor is there a single reference in the book on the issue of keeping up to date with patches (which is how the vast majority of worms propagate). The strategy for dealing with malware is mostly reactive (host based intrusion detection, etc). Similarly, Internet Security: Professional Reference (Atkins, et.al. 1998) has no mention of systemizing patching security holes even though worms and advisories had been out for some time, even in 1998.

Past the invective to "patch, patch, patch!" there is little in the way of practical system, infrastructure builing and tools for the tracking and deployment of patches. Microsoft's Hfnetchk.exe has struggled to provide correct output. Solaris has had a few tools, and Debian Linux has developed a nice online patch system in their "apt-get" utility (http://www.debian.org). Even so, my own experience as a data security consultant, I have wasted countless hours trying to figure out which servers were patched to what patch level. Additionally, much of the damage of the last two years (Nimda, CodeRed, Li0n, Ramen) all relied on exploits for which there have been patches available for *months*. CodeRedII came almost 2 months after the availability of the patch that could have stopped it (Caida, 2001) and the Li0n and Ramen worms hit even longer than that after patches were available. Following Bruce Schneier (Schneier 2000), we are trying to close the window of exposure as rapidly as possible as the number of exploits and vulnerabilities increase in number and severity, and as the turn time for viruses and worms shortens. I try to inculcate a attitude of Aggressive Patching in my consulting clients and in my own systems. Patching is a crucial part of system life and is utterly crucial to system security. Aggressive Patching takes this concern the the deployment and whole life cycle of an internet server farm. This paper attempts to start to fill that gap for a particular demographic.

Aggressive Patching is defined as:
1) Patching is built into the design of your infrastructure.
2) Responsibility and time frames for patching is written into your security policy. See Appendix E for examples.
3) Patching systems is planned for the life cycle of the server farm, including servers that are held back from upgrade due to difficulty in migrating to a new or more recent platform.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

4) Having tools and checklists to assure that patches are completed in a timely manner and tracked to completion.

Finally, though I will not get into the details of the full disclosure debate (whether vendors and security consultants should post full details and even exploit code to the public), in my experience, if one is practicing Aggressive Patching, full disclosure allows you to make the fastest assesement of the situation and to most accurately test if the patch has fixed the problem, without having to have developers on site to write an exploit to test the fix. With a patch response time of 24-96 hours, full disclosure will generally not hurt us as virus writers, while speedy, are usually not that speedy, and the time gained by knowing everything and having tools to work with will help win the war with virus/worm writers.

With this in mind here is a tested architecture for small to medium sized businesses with an internet server farm (small ISP's, virtual hosting companies, online survey companies, etc) of around 4-50 machines. This exampe is OpenBSD based for reasons given below, but this can work easily with any Unix supporting ssh and cvs source code updates. It relies on a Master Server that will have the source tree repository for the Standard Build and will keep both the source and binary patches up-to-date. From there it is easy to build tools to roll out the patches to other servers, a few of which are included below. Additionally, it is easy to build and rollout new servers, cutting the time of deployment by over 50% in the test companies I have used this in. Included are original and tested OpenBSD hardening checklists, pf.conf examples and original perl and shell scripts to automate the patching procedure. Since this is aimed at small to medium sized business, it is relatively lightweight but effective. The overhead cost of this setup is next to nothing. By way of hardware you need a Master Server that does not need to be particularly robust, though it should be reliable (and ideally nearly identical to the standard build server hardware configuration). You will need a few hours (3-8) to set up the Master Server and Prototype. The rest is all free. This procedure should take less than a day to complete, but could save dozens or hundreds of hours of patch tracking, server maintenance, and new server deployment. Most of all, rapid patching greatly heightens over all security, reducing costs, downtime and overhead. With this system you can roll your servers over one at a time when they are ready to be moved from the old platform with minimum hassle as you have the ability to create a fully patched and configured server in less than an hour.

I think the advantages of having a Standard Build are clear. The barriers to deploying a Standard Build in a small/medium business environment are several, including:

Cost of buying standard hardware
Cost of setting up the Standard Build configuration
Lack of tools and models

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

Migrating any legacy apps to the new platform.

I cannot deal with all of these issues, though I hope to deal with the second and third items above.

The goals I have in mind are:
Ease of patching
Ease of deployment in a small to medium environment
Low capital oulays
Simplicity of design over advanced functionality or graphical user interface

I have chosen OpenBSD for six reasons:
1) It is flexible and open source. This allows me to make changes as needed to make my environment safe.
2) Their development model places a high value on security (rolling code audits, careful introduction of new features, etc).
3) Their installation is simple which allows for ease in standardization to my specifications.
4) The project has matured greatly in the last 2 years and includes a rich variety of packages and functions (MySQL, PHP, Radius, DNS, etc).
5) The use and understanding of OpenBSD can greatly contribute to a secure environment
6) It is free!

Structure of the Document:

The rest of the document is broken into three parts.
A short overview of the topology of the server farm we are looking at for this infrastructure.
A nearly step-by-step procedure on how to setup, deploy and maintain the patching system
Appendices that contain checklists, tools/scripts and configuration examples referenced in the main body, but would be
disruptive to include directly.

Server Farm overview:

We have an indeterminate number of internet servers serving different functions (web, dns, radius, database, etc). My
recommendation is between 4 and 50. Less than four and you loose the economies of scale that the extra overhead costs you to
set up the infrastructure. Over 50 and we would want to replace some of the flat text files with something a bit more dynamic
(awk script at least, or SQL query, to create the server lists, and more flexible scripts to do the patching).

There will be a server behind the corporate firewall that will act as a Master Server, executing remote jobs and being the

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

main source code repository for the OpenBSD release in question. This server will be the first to be patched and test the
new binaries. This machine will also have passwordless (dsa key) ssh root access to the entire server farm, and as such
needs special physical and virtual security considerations. Additionally, this server can have vpn access (OpenBSD comes with
built-in VPN) to the server farm, but we will restrict ourselves to NFS in this example.

If funds allow, we can have an additional prototype/dev server that replicates many of the functions of the server farm to
test the rolled out patches.

A firewall that can redirect either IPSec VPN or NFS (we will use NFS in this example) from the server farm to the Master Server.

Procedure:

The following does assume some basic Unix competence with a text editor, mounting volumes, using 'make' to compile software
and the rudiments of NFS. Basically, the process to set up and deploy the infrastructure is as follows:

Initial Setup of Master Server, Prototype, and Standard Build
Deployment
Maintenance
New Server Rollout

I. Set up the Master Server.

A. Make sure that this server is physically secure as it holds a good deal of power over your other servers.

B. Install the basic OpenBSD 3.0 standard build on the Master Server, using the floppy or CD set and following the directions on the OpenBSD site. (ftp://ftp3.usa.openbsd.org/pub/OpenBSD/3.0/, then find your architecture and look at the INSTALL.'arch' document.

C. Place or download the ports tree and source tree and ungzip them in their respective locations. (See http://www.openbsd.org/faq/faq8.html#Ports and http://www.openbsd.org/faq/faq5.html)

D. Run through the checklist from Appendix A

E. Set up an anonymous ftp server per the OpenBSD faq (http://www.openbsd.org/faq/faq10.html#AnonFTP)

F. Update the source code via CVS. Execute the following commands (this is included as a shell script, cvs-src.sh):

```
cd /usr/
export CVSROOT=anoncvs@anoncvs6.usa.openbsd.org:/cvs
export CVS_RSH=/usr/bin/ssh
/usr/bin/cvs up -rOPENBSD_3_0 -PAd src
```

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

G. Create a patch list: Make a directory /root/patch. Go over all of the released patches from the OpenBSD errata.html page. At the beginning of each patch is a list of directories that must be rebuilt. In a text file called /root/patch/patches.txt on the Master Server, make a list of those directories, one per line.

For example, patch_16 from ftp://ftp.openbsd.org/pub/OpenBSD/patches/3.0/common/016_approval.patch has three directories that need updating.

libexec/rshd
libexec/rexecd
libexec/atrun

Note the paths are relative to /usr/src/

Keep the list current as patches come out.

There is a current list at ftp://ftp.uptime.org:/pub/OpenBSD/3.0/patch/patchlist.txt (3.1 will be added by 06/01/02)

H. Create a standard build serverlist. In /root/patch/serverlist.txt put, one per line, a list of all of your standard build machines by IP or fqdn, not including the Master Server.

I. Create a patch-notes.txt file that notes:

If you need to use the Makefile.bsd-wrapper to build the binary for a particular directory

What services need to be restarted after an update.

This is just to remind you of oddities in the patching process over the months.

J. Add the following scripts to your /root/patch/ directory

remote-command.pl
patch.pl
cvs-src.sh
cvs-ports.sh

These can be found at ftp://ftp.uptime.org/pub/OpenBSD/3.0/patch/ or in Appendices D, F and G.

K. Configure a kernel for your Standard Build hardware and compile it according to the OpenBSD faq. (http://www.openbsd.org/faq/faq5.html)

L. Update the ports tree with the /root/cvs-ports.sh or these commands and BASH-like shell:

export CVSROOT=anoncvs@anoncvs6.usa.openbsd.org:/cvs
export CVS_RSH=/usr/bin/ssh
/usr/bin/cvs up -PAd ports

M. Build what ever ports are necessary for your environment. This would include all packages you would need for your various functions. Use *make install* if the Master Server needs it for local use or use *make package* if it is for one or more of the Standard Build servers.

N. Enable NFS sharing for the Master Server. Read only access for the /usr/src and /usr/obj directories to your server

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

farm. Also share out your compiled packages directory.
In /etc/exports (replace the ARCH, network and netmask as appropriate).
/usr/src /usr/obj /usr/ports -ro -maproot=daemon -network 192.168.0 -mask 255.255.255.0
/usr/ports/packages/ARCH/All -ro -maproot=daemon -network 192.168.0 -mask 255.255.255.0

O. Symlink your /usr/ports/packages/ARCH/All directory into your ftp root so you can download packages if necessary (replace ARCH with your architecture, i386, etc).

P. Generate ssh keys
ssh-keygen -t dsa

II. Create a prototype standard build

A. On standard build hardware, run through the standard like you did on the Master Server installation and pull your binaries from the Master Server's ftp site. Reboot into the new system.

B. Install, via ftp or nfs, any packages you need from the Master Server. The ports you install should be few and generic (basic text editors, system tools and gtar, etc) and not server specific (radius, MySQL, etc).

C. Install the Master Servers ssh keys for passwordless access in /root/.ssh/authorized_keys (see Appendix A)

D. Use the checklist in Appendix A to harden the box.

E. Copy the current working kernel (/bsd) to /bsd.old so you always have a working kernel to boot from.
cp /bsd /bsd.old

III. Patch the Master Server and the Prototype.

A. Use the patch.pl script in Appendix F to patch the Master Server and the Prototype.

B. Copy the new kernel from the Master Server to both the Master Server and Prototype.

C. Reboot both Master Server and Prototype.

D. Once both servers have been patched, create the standard build tarball. On the Prototype, (gnu)tar up the entire harddrive (you did install gtar from the /usr/ports/archivers port, yes?). Call it base30.tgz. Use this command:
cd / ; gtar -cxzv --exclude var/tmp/base30.tgz -f /var/tmp/base30.tgz /

E. Copy the standard build base30.tgz to the Master Server's anonymous ftp site. In the same directory, put the newly built kernel from /usr/src/sys/arch/... If you want to make life easy, place this in pub/OpenBSD/3.0/i386/ under your ftp root, that way you can easily use the standard OpenBSD install disks to create your customized Standard

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

Build. You will use the Master Server as your ftp server in the install and it will serve up the two files needed to replicate your server configuration, base30.tgz and bsd.

IV. Roll out the other servers

A. Using the basic OpenBSD floppy or CD install, you can use your new base30.tgz and kernel from your Master Server's ftp site to create pre-hardened and pre-patched standard servers with all your basic packages already installed.

B. Using NFS or FTP from the Master Server, add whatever packages each server needs to function.

C. Configure networking and services for the server and create and appropriate /etc/pf.conf to lock the box down, see Appendix B and Building Linux and OpenBSD Firewalls (Sonnereich and Yates, 2000) for examples.

V. Maintenance mode

As new patches become available, you can use the same script that patched the Master Server and Prototype (patch.pl) to keep the entire server farm up-to-date with no effort. When a patch comes out, update the Master Server. If you have the funds for a dev/prototype server, then update that also. Make sure the patch does not break anything and then deploy to all servers with the execution of the patch.pl script. In my experience, most patches from OpenBSD core are solid and do not break other apps.

Before you begin, you will need to let NFS through your firewall (at least temporarily). This is a bit of a pain and you can try to use the built in VPN in OpenBSD to get around this, but for now here is the basics of opening ports for NFS.

Assuming you have a NATing firewall, you need to allow ports tcp 2049 and udp 111 to be redirected to the Master Server from the outside interface of the firewall to the Master Server. Then there are two more ports that mountd requires and those change with each reboot of the Master Server. To find out what they currently are, as root, run *rpcinfo -p* on the Master Server and find the two ports that mountd needs. One will be tcp one will be udp. For example:

```
program vers proto   port
 100000    2   tcp    111  portmapper
 100000    2   udp    111  portmapper
 100005    1   udp    697  mountd
 100005    3   udp    697  mountd
 100005    1   tcp    826  mountd
```

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

100005 3 tcp 826 mountd

You would also need to forward port 697 udp and 826 tcp. So four ports will need forwarding. If you are using an OpenBSD firewall, see Appendix C for an OpenBSD pf.conf and nat.conf that will do this.

There are three kinds of updates:
A) Core system (things in /usr, /bin, /sbin, ...)
B) Kernel
C) Packages from the ports tree

A. Core System

1. On the Master Server, run patch.pl and it will walk you through the process. Make sure you have looked at the patch itself via the web interface (http://www.openbsd.org/errata.html). This will tell you the directories that need patching. patch.pl will download, compile and install the patches onto the Master Server. If you are confident the patch will not break anything, you can simply deploy from there. (See the policy in Appendix E for suggestions on how to decide if a patch is safe or not.) Add the new directories, one per line, to /root/patch/patchlist.txt file for future new builds.
2. Restart any services that are needed (the patch will tell you which to restart). Use remote-command.pl to do this on all relevant servers.
3. Done. For the last several patches, it has taken me about 30 minutes to patch 25 servers for two companies.

B. Kernel

1. Rebuild the kernel(s) on the Master Server and install. Reboot to make sure it works.
2. Upload the kernels with this little while loop:
cat /root/patch/standardbuild.txt | while read line ; do scp /path/to/compiled/kernel root@"$line":/ ; done
3. Use the *remote-command.pl* script to reboot the machines, or do them manually if that is better for your setup.
4. Copy the new kernel to your Master Server ftp site so new machines get the most recent kernel.

C. Packages

1. As there are patches to your favorite packages, use the /root/patch/cvs-ports.sh to update your ports tree, then recompile by using *cd /usr/ports/path/to/port ; make clean, make package* .
2.Use the remote-command.pl scripts to install the package and then restart any services.

VI. New server rollout after initial deployment.

This can be used to slowly replace the current server farm as the conversion of any proprietary or in house code takes place or in bulk to do several new servers at once.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

1. Use the standard CD or floppy install disks. Do an ftp install from your Master Server and fetch base30.tgz and
bsd. This installes a server that has your basic packages and has already been run through the hardening checklist.

2. Run the patch.pl on the Master Server and update all patches on the box (just to be safe).

3. Use ftp or NFS to install the packages you need.

Summary:

Aggressive patching means building in a patching system to your infrastructure that makes patching easy to
do. Additionally,someone must be responsible for monitoring patches by getting on the correct mailing lists and watching the
right web sites regularly (daily). Following the above procedure, or something similar, will greatly enhance your ablility
to respond to security threats and focus on other issues. The configuration above is inexpensive and low maintenance,
perfect for small/medium businesses on a budget. As with any approach to an aspect of security, Aggressive Patching is only
one part of an over all security strategy that includes firewalls, intrusion detection, host based defenses and backups.

All of the tools mentioned are of my own invention (hence I deserve all the blame!) and are in the Appendicies in order to
make the document more comprehensible.

----------Appendix A: OpenBSD Hardening Checklist ----------

OpenBSD Internet Server setup checklist:
Michael Sullenszino
version 1.0.2 - January 6, 2002
This document assumes a basic knowledge of Unix, a unix shell, a text editor
and basic unix services.

File system layout
Minimum recommendations, more usually can't hurt
/root 300MB
/usr 1000MB (2000MB or more if you have the source and ports tree installed)
/var As much as you can spare
/var/log 500MB

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

/home Depends on the function of the box

I separate /var/log so log overload does not crash mail, database or other /var/ functions. Even if you are using a syslog server, keep some space for local logging.

Once you have installed the base system.
All commands should be run as root unless otherwise stated.

Global:
Edit /etc/sysctl.conf to allow kernel services you need (vpn, packet forwarding, swap encrypt etc)
Edit /etc/rc.conf to allow and shutdown services you (don't) need, shutdown portmap and inetd if you don't need them.
Edit /etc/inetd.conf and shut down/enable what is minimally necessary
Edit /etc/rc.local to start 3rd party services (MySQL, etc).
Edit /etc/pf.conf to set up basic host based perimeter, see example in Appendix B
Edit /etc/hosts to add any IPs you do not want to trust DNS with, localhost, your firewall and Master Server are recommended.
Edit /etc/resolv.conf and add the line "lookup file bind" if not there, this makes sure your hosts file is used before DNS
Edit /etc/newsyslog.conf to add any other logs you want rotated (exim,apache, etc)
Edit /etc/motd to put a restriction notice in it, see below Appendix H
Edit root's dot.files
    Add /root/patch to any shell profiles
    Add any public keys to /root/.ssh/authorized_keys
        (root@MasterServer should be there)
Make the patch directory
    mkdir /root/patch
    chmod 700 /root/patch
Check file permissions on /home and in /etc/adduser.conf make sure they are least permissive for the function of the box.
Run adduser to set default new user settings to the least permission necessary
Run crontab -e
    Comment out the sendmail line

Service Specific: The basic three services, SMTP, HTTP and DNS all will run without root priviledge!

We have used exim for years as it is robust, has an excellent security track record, comes with easy virus filtering and is a drop in replacement for sendmail.

Exim:
Install the exim package from Master Server
Edit /etc/mailer.conf to change sendmail to /usr/local/sbin/exim
Edit /etc/exim/configure (make sure to run exim as user "exim" and group "mail"
Add user exim and group mail so it does not run with root priviledges
    useradd exim
    Edit /etc/group and add mail as group "12" (or whatever works for your installation).
Edit /etc/rc.conf and set the sendmail flags to just "-bd -q30m" if you need to run the daemon.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

Run, as root, crontab -e and comment out the sendmail line.
Set ownership of relevant directories

```
chown root.mail /var/mail
mkdir -p /var/spool/exim/log
chown -R exim.mail /var/spool/exim
```

Set permissions on directories

```
chmod 775 /var/mail
chmod -R 750 /var/spool/exim
```

Edit /etc/mail/aliases and point root to a real user email address at your domain.
Edit /etc/daily (daily cron jobs to send the exim stats and rotate the logs, and change the email address)

Add these lines

```
# Send out eximstats and rotate logs
if [ -x /usr/local/sbin/eximstats ]; then
    /usr/local/sbin/eximstats -nr /var/log/exim/mainlog | mail -s "   ostname Exim Stats"
your_address@yourdomain.com
fi;
if [ -x /usr/local/sbin/exicyclog ]; then
    /usr/local/sbin/exicyclog
fi

if [ -f /etc/daily.local ]; then
    echo ""
    echo "Running daily.local:"
    . /etc/daily.local
fi
```

Change permissions on the original sendmail binary

```
chmod 000 /usr/libexec/sendmail/sendmail
```

DNS
runs fairly well and fairly secure out of the box. Don't run it if you don't need it!

Apache
Check your CGI options and directory permissions. For a basic cgi site, the defaults are mostly good!

----------Appendix B: Sample pf.conf----------

Sample pf.conf for a single nic internet server (radius, web, dns, etc)
You will normally only need to edit:
1) The nic (change fxp0 to your nic driver)
2) The firewall and dnsservers ip addresses
3) The services you want to allow in/out, examples given

This is a fairly aggressive host based perimeter, block all except the services we offer and anything to the firewall. Even
the services we offer must be a SYN packet if TCP, this limits OS finger printing, etc. ICMP is optional. We log quite a
bit. I took some of the spoofing ideas out of Building Linux and OpenBSD Firewalls (Sonnereich, 2000).

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

```
---------------------------------------------------
# Begin pf.conf
# Michael Sullenszino - Single nic internet server
# version 1.2 February 10, 2002

###########
#Variables#
###########
external=fxp0
loopback=lo0
spoofed="{ 0.0.0.0/32, 10.0.0.0/8, 127.0.0.0/8, 172.16.0.0/16, 192.168.0.0/16,
255.255.255.255/32 }"
localnet=fxp0/32
externalip=fxp0/32
firewallip = "222.222.222.222/32"
dnsservers = "{ 222.222.222.220/32, 222.222.222.221/32 }"


###################
#Default Rule Set#
###################
block in log all
block out log all

#
# Block always evil traffic
#
block in log quick proto tcp from any to any flags SF/SF
block in log quick proto tcp from any to any flags SR/SR



#######################
# External Interface #
#######################
# Block traffic that should never be on the external interface
block in quick on $external from $spoofed to any
block in log quick on $external from $externalip to any

# Reject identd requests, this can speed up some connections
block return-rst in quick on $external proto tcp from any to any port = 113

#
# Services we allow outside connections to, add or remove as needed for the server
# In this example we allow outside connectins to DNS, http and https.
# Allow HTTPS
pass in on $external proto tcp from any to $externalip port = 443 flags S keep state
# Allow HTTP
pass in on $external proto tcp from any to $externalip port = 80 flags S keep state
# Allow in DNS requests
pass in on $external proto udp from any to $externalip port = 53 keep state


#
# Services we want to allow connections out
```

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

```
#

# Allow outgoing tcp connections (optional)
pass out on $external proto tcp from $externalip to any port = 80 flags S keep state

# Allow connections to our firewall for internal services from the Master Server
pass out on $external proto tcp from $externalip to $firewallip flags S keep state
pass out on $external proto udp from $externalip to $firewallip keep state

# Allow outgoing dns to our nameservers
pass out on $external proto udp from $externalip to $dnsservers port = 53 keep state

# Comment the following out if it does not meet your security policy
# Let in the crucial ICMP packets
pass in on $external inet proto icmp all icmp-type 0
pass in on $external inet proto icmp all icmp-type 3
pass in on $external inet proto icmp all icmp-type 8
pass in on $external inet proto icmp all icmp-type 11
pass out on $external inet proto icmp all icmp-type 0
pass out on $external inet proto icmp all icmp-type 3
pass out on $external inet proto icmp all icmp-type 8
pass out on $external inet proto icmp all icmp-type 11


#End of pf.conf
```

----------Appendix C: NFS portforwarding----------

If you use OpenBSD for your firewall, these entries in your nat.conf and pf.conf files will allow NFS to be redirected to the
Master Server.

for pf.conf use:

```
# Allow incoming nfs request for Master Server
# external is the "$external" nic name and "$masterserver" is the ip of the Master Server
# Comment these out when not needed.
external="fxp0"
masterserver="192.168.0.x/32"
pass in on $external proto udp from any to $masterserver port = 111 keep state
pass in on $external proto udp from any to $masterserver port > 500 keep state
pass in on $external proto tcp from any to $masterserver port > 500 flags S keep state
```

For your nat.conf use:

```
# Forward NFS request to Master Server, used to do updates to OpenBSD boxen
# To figure out which ports to forward on the first two lines, go to
# Master Server and run 'rpcinfo -p' . Set the tcp and udp ports correctly
# Change firewallip and masterserver to appropriate values.
# Comment these out when not needed.
firewallip="222.222.222.222/32"
masterserver="192.168.0.xx"
rdr on fxp0 proto udp from any to $firewallip port 697 -> $masterserver port 697
```

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

```
rdr on fxp0 proto tcp from any to $firewallip port 826 -> $masterserver port 826
rdr on fxp0 proto tcp from any to $firewallip port 2049 -> $masterserver port 2049
rdr on fxp0 proto udp from any to $firewallip port 111 -> $masterserver port 111
```

When (un)commenting these items, reinitialize the firewall with:
pfctl -R /etc/pf.conf -N /etc/nat.conf

----------Appendix D: Scripts Descriptions----------

Here is a brief description of the scripts mentioned in the paper.

1)cvs-src.sh and cvs-ports.sh
These are just simple little scripts to update the source code and ports tree of an OpenBSD 3.0 system.

```
-------
#!/bin/sh
# Michael Sullenszino
# version 1.0 December 3, 2001

# cvs-ports.sh - Small script to automate update of the ports tree

cd /usr/
export CVSROOT=anoncvs@anoncvs6.usa.openbsd.org:/cvs
export CVS_RSH=/usr/bin/ssh
/usr/bin/cvs up -PAd ports

exit 0

-------
```

```
#!/bin/sh

# cvs-src.sh - Small script to update source tree
# Michael Sullenszino
# version 1.1 December 3, 2001
cd /usr/
export CVSROOT=anoncvs@anoncvs6.usa.openbsd.org:/cvs
export CVS_RSH=/usr/bin/ssh
/usr/bin/cvs up -rOPENBSD_3_0 -PAd src

exit 0

-------
```

2) remote-command.pl
This executes one or more commands on one or more servers. If you do not list any servers it will execute the command on all
servers listed in /root/patch/serverlist.txt. This can be used to install packages or to restart services after an update.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

The full text of this script is in Appendix F

To customize it for your site, set the variable at the top of the script, and manually set your internal ip range from
192.168 to whatever you use.

3) patch.pl

The full text of this script is in Appendix G

This is the main update engine for OpenBSD core source patches. It prompts you for:
information about the patch
the status of the Master Server (is Master Server patched?)
the servers you want to patch (or if "all", it will take all servers in /root/patch/serverlist.txt)
the special Makefile.bsd-wrapper that pops up in some of the software included with OpenBSD that needs special attention.

From there it will do a cvs update on the necessary directories, build and install on the Master Server (if necessary) and
then NFS mount the /usr/src and /usr/obj directories on remote machines and install them there too.

You will need to look at the variables at the beginning of the script for it to function properly and edit the internal
network if it is not 192.168.x.x.

----------Appendix E: Policy----------

As excerpted from a generic security policy I use. It should be noted
that, if applicaple, at least one member of the development
team is required to be on the Security Team and speak for the developers...

4. Patching - Patching is defined as updating software to close a
known security vulnerability.

3.a. As patching known security holes is crucial to the stability and viability of the server farm, the Security Team is responsible for monitoring vendor advisories for all software. If resources are available, a secured list of vendor software should be posted and responsibility given to individuals for each vendor as appropriate. Additionally, one or more people should monitor a few of the white, grey and black hat sites for early announcements of exploits as resources allow. (For example, Securityfocus.com, packetstorm.com, theregister.uk.co, etc)

3.b. Patch timeline - When a patch is released, the Security Team will make a determination of the level of the patch's potential to harm any functions. Depending on the impact the patch may have the following timelines will be adhered to:

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

Insignificant - the patch is strongly believed not to break any server function. Patch by the next business day. (Example, small five line source code patch to a little used library to clean up a typo or buffer overrun.

Moderate - Patch is believed not to harm any functions, but a short test period on a non-production server is prudent. Patch within 2 business days. (patch to a web server that will probably not break anything, but could harm web services if something unexpected happens).

Significant - the patch has the potential to harm mission critical applications or functions. Patch within 5 business days. Example: Significate patches to RDBMS systems.

Unknown - The damage of the patch could be significant, but there is no precedent to decide if the patch will harm current functions. The security team, development team (if any) and vendors will be consulted and tests run over the next 48 hours, at which point the Security Team will review the results and rate the patch with the preceding scale. Example: forced version upgrade to a scripting engine used on the main web site where custom web applications are running.

3.c Tracking - The Security Team will delegate responsibility for tracking an open patching issue to an appropriate individual, that individual will post information to a secure location accessible to the Security Team, the Development Team and Management regarding the progress and completion of a patching operation. That individual also is responsible for following up with other team members to make sure the patch is completed on time and will report any difficulties or need to extend the above deadlines.

----------Appendix F: patch.pl----------

```
#!/usr/bin/perl

# Script to automate the patching of any or all of our openbsd
# servers.  It asks several questions about the source directories to
# update, whether we need to update and build those directories on the
# Master, and then uses nfs to mount and install the new binaries on
# our server farm.

# Thanks go to O'Reilly's the Perl Cookbook (Christiansen, 1998) for little
# snippets of code and ideas. Forgive the clumsyness of this script it is a year
# old and one of my earlier forays into perl!

##########################################################
# Set up variables/arrays and prompt for user input
#

# Fixed variables
#
$firewall = "your.firewall.com";
```

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

```
$ssh = "/usr/bin/ssh";
$nfsserver = "your.masterserver.com";

# User input variables
#
# Do we need to update the source tree?
print ("\n\nDo we need to do a cvs update of the source tree? [y/n] ");
$cvsupdate = ;
chomp $cvsupdate;
# Figure out if the new binaries need to be built on Master Server (we
# will assume that if we have to update the source tree, we have to
# build
if ($cvsupdate eq "y") {
  $buildhere = "y";
  chomp $buildhere;
} else {
  print ("\n\nDo the binaries need to be built on Master Server? [y/n] ");
  $buildhere = ;
  chomp $buildhere;
}


# Ask for directories to run make install on or give the all option and pull all from file.
print ("\n\nEnter the names of the directories under /usr/src/ needing updating,\n");
print (" separated by whitespace.\n");
print ("For example: lib/libz  usr.bin/ssh libexec/rshd \n");
print ("If you want to run all patches, just hit enter.\n");
print ("Directories to run 'make install' on: ");
$dirs = ;
chomp $dirs;

# If dirs is empty then take all directories from the patchlist.
if (! $dirs) {
  open (DATA, "< /root/patch/serverlist.txt")            or die "Couldn't read from datafile: $!\n";
  while () {
    chomp;
    push(@servers, $_);
  }
  close (DATA);
# If not a all, then split whitespace list into an array.
else {
  @servers = split (/\s+/,$servers);
}

#
# Run through the server list and execute the command
#
foreach $server (@servers) {
  print ("****** $server *****\n");
  system ("$ssh root\@$server $command\n");
  print ("\n");
}

# End remote-command.pl
```

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

----------Appendix H: Banner Warning Message----------

This machine and all of its services are for authorized use only. All unauthorized use is strictly prohibited. All authorization must come from your.admin.address@here.com in writing. Log off now and obtain that authorization.

All users, authorized or not, have no explicit or implicit expectation of privacy. By using this system, the user consents to such interception, monitoring, recording, copying, auditing, inspection, and disclosure at the discretion of authorized site.

Unauthorized or improper use of this system may result in administrative disciplinary action and civil and criminal penalties. By continuing to use this system you indicate your awareness of and consent to these terms and conditions of use. LOG OFF IMMEDIATELY if you do not agree to the conditions stated in this warning.

References:

Anonymous. Maximum Security Linux. 2000. SAMS Publishing, Indianapolis, IN. 168-190
ISBN: 0-672-31670-6

Atkins, Derek, et.al. . Second Edition. 1998. New Riders Publishing, Indianapolis, IN.
ISBN: 156205760X

CAIDA. "CAIDA Analysis of Code-Red". CAIDA. (March 30, 2002)
URL:http://www.caida.org/analysis/security/code-red/

Christiansen, Tom and Torkington, Nathan. Perl Cookbook. 1998. O'Reilly Publishers, Paris.
ISBN: 1565922433

Microsoft. "Microsoft Technet" Microsoft Corporation. (April 5, 2002)
URL:http://www.microsoft.com/technet/security/current.asp

OpenBSD. "OpenBSD FAQ". OpenBSD. (April 5, 2002)
URL:http://www.openbsd.org/faq/

OpenBSD. "OpenBSD Errata" 1998-2002 OpenBSD. (April 5, 2002)
URL:http://www.openbsd.org/errata.html

RedHat. "Red Hat Linux 7.2 General Advisories". Red Hat, Inc. (April 4, 2002)
URL:http://www.redhat.com/apps/support/errata/index.html

RedHat. "RedHat Security Resource Center" Red Hat, Inc. (April 4, 2002)
URL:http://www.redhat.com/security

Schneier, Bruce. "Closing the Window of Exposure: Reflections on the Future of Security". (Sep 18, 2000)
URL:http://online.securityfocus.com/guest/3384

Sonnenreich, Wes and Yates, Tom. Building Linux and OpenBSD Firewalls. 2000.
John Wiley and Sons; New York, NY.
ISBN: 0471353663

© SANS Institute 2002, Author retains full rights.

© SANS Institute 2002, Author retains full rights.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

Last Updated: November 28th, 2015

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Cape Town 2015	**Cape Town, ZA**	**Nov 30, 2015 - Dec 05, 2015**	**Live Event**
SANS San Francisco 2015	**San Francisco, CAUS**	**Nov 30, 2015 - Dec 05, 2015**	**Live Event**
HIMSS	**Boston, MAUS**	**Dec 01, 2015 - Dec 02, 2015**	**Live Event**
Security Leadership Summit & Training	**Dallas, TXUS**	**Dec 03, 2015 - Dec 10, 2015**	**Live Event**
SANS Cyber Defense Initiative 2015	**Washington, DCUS**	**Dec 12, 2015 - Dec 19, 2015**	**Live Event**
SANS Dubai 2016	**Dubai, AE**	**Jan 09, 2016 - Jan 14, 2016**	**Live Event**
SANS Las Vegas 2016	**Las Vegas, NVUS**	**Jan 09, 2016 - Jan 14, 2016**	**Live Event**
SANS Cyber Defence Delhi 2016	**Delhi, IN**	**Jan 11, 2016 - Jan 22, 2016**	**Live Event**
SANS Brussels Winter 2016	**Brussels, BE**	**Jan 18, 2016 - Jan 23, 2016**	**Live Event**
SANS Security East 2016	**New Orleans, LAUS**	**Jan 25, 2016 - Jan 30, 2016**	**Live Event**
Cyber Threat Intelligence Summit & Training	**Alexandria, VAUS**	**Feb 03, 2016 - Feb 10, 2016**	**Live Event**
SANS Scottsdale 2016	**Scottsdale, AZUS**	**Feb 08, 2016 - Feb 13, 2016**	**Live Event**
SANS Secure Japan 2016	**Tokyo, JP**	**Feb 15, 2016 - Feb 20, 2016**	**Live Event**
SANS Munich Winter 2016	**Munich, DE**	**Feb 15, 2016 - Feb 20, 2016**	**Live Event**
SANS McLean 2016	**McLean, VAUS**	**Feb 15, 2016 - Feb 20, 2016**	**Live Event**
ICS Security Summit & Training	**Orlando, FLUS**	**Feb 16, 2016 - Feb 23, 2016**	**Live Event**
SANS Secure India 2016	**Bangalore, IN**	**Feb 22, 2016 - Mar 05, 2016**	**Live Event**
SANS Anaheim 2016	**Anaheim, CAUS**	**Feb 22, 2016 - Feb 27, 2016**	**Live Event**
ICS410 London 2015	**OnlineGB**	**Nov 30, 2015 - Dec 04, 2015**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**