How To Install a Secure BSD System

Author: soup4you2
Posted on: 9/21/2003
Discuss:
http://www.littleblackdog.com/viewtopic.php?p=194945
http://www.littlewhitedog.com/content-72.html

The information contained within was copied from: http://www.littlewhitedog.com/
This document was created as a single reference for building a secure email server using FreeBSD 5.x and information from Mail::Toaster.
This document will be used as a starting point for a Mail::Toaster email server.
I have also taken the liberty of deleting some unnecessary text denoted by {snip}. Refer to the above website links to see what's missing.

Typeset Conventions:

- *Modification I have made to this document are in this font.*
- Original text from the author "soup4you2" are in this font.
- Text from Mail::Toaster are in this font.

Introduction

I once wrote up an article on installing and securing the FreeBSD operating system. However, in that article I was unable to go in-depth into the various ways I might have configured certain parts of the operating system without compromising security to the website. So this is my gift to the community that has greatly supported me throughout the past years. And I hope that you find it somewhat educational or stimulating. You do not need to be running a FreeBSD operating system to follow this document. I recommend all users of all distributions read this article just for something to learn. If you like this document please comment on it or drop by IrC and say thanks (which is in no way required by me). A lot of hard work goes into these articles and I enjoy hearing people's feedback. I'm just trying to be a good guy. I highly suggest reading though this article a couple times before attempting this on your own servers and if any damage or data loss occurs I am not responsible. You have been warned.

Why write this document?

Our exploration into setting up this server started with a simple question from our friends. That question always tends to be "How do I set this up"? Well, in this article, we are going to cover many areas of the operating system, perhaps a little bit too much. The overall goal here is to help you setup a secure server with Mail services configured for multiple domains, and various other services that a typical user needs. Basically, I'm writing up this article to keep the flow of information going after all the months of research I had to do. I'm taking my best articles from my site and putting them into one massive document.

Why choose FreeBSD?
FreeBSD is, in my opinion, a truly great operating system, unlike its rival Microsoft (in which 2 images mocking their policies and practices can be seen below). It is a fairly secure and easy to manage operating system, though not as secure as OpenBSD. And the best part is, things are easily installed and kept up to date, unlike your Linux systems out there. Don't get me wrong, Linux is great and all, but about 75% of the packages I install are custom based and, well, RedHat sucks when it comes to that.

What's so bad about the Linux updating system?
Well, you need to keep in mind that the BSD distros are mostly source-based, from the packages you install to updating the operating system. And when you're dealing with source-based you can completely configure the application to do what you want and not what the person who made the package intended. So when you're using services such as up2date, you're using pre-packaged binaries that just don't suit my needs.

{snip}

FreeBSD is an industry standard
FreeBSD currently powers a large portion of the Internet. Over 30% of ISPs and web hosting companies have chosen FreeBSD because of its excellent performance and reliability. In addition, some of the busiest and most successful Internet sites in the world use FreeBSD.

The success of these sites proves that FreeBSD is one of the most powerful and scalable operating systems available. You can take advantage of that same power and scalability.

Requirements:
- Good background knowledge of the FreeBSD operating system
- A lot of free time and lots of patience
- And a good amount of some beer never hurts
- Compatible system for the FreeBSD Operating system
- ISO Images of the latest version of FreeBSD
- Internet Access

What's covered in this document?
- Base OS Installation
- First time configurations (Basic System security Part 1)
- Updating your server
- Basic System Security Part 2
- Unix File Permissions
- Installing and configuring Mail services
- Firewall configurations
- Closing Remarks

Base OS Installation:

Please note that our first goal here is to secure your box. Then we'll add service by service, securing them as we go along. Normally speaking, from experience, if you skip steps as you go along or rush installation of your services, you often forget and leave things unattended that should be set up. And often in that case you tend to forget about them.

Installing FreeBSD from CD:

- Ensure that the network cable is not plugged in; network access is not needed at this stage. Power on with the FreeBSD CD in the CD-ROM drive.
- Select “Standard” from the main menu.
- Select the size for the partition. For the database host, all of the drive space on the drive array will be used (press Z to toggle to size, it makes it easier to see the actual size). Press A to use the whole disk and then press S to make this a bootable partition. Then Q when finished.
- Choose your boot manager. Since FreeBSD is the only OS on the machine, we will be using standard. Failure to do so may render the computer unbootable after the installation.
- Use auto assign for the disk space. The program will automatically assign most of the space to /usr. Since the database will physically reside at /var/db, most of the disk space will be assigned to that file system. Move cursor to the /usr and delete the partition. Create /usr again with 4G, which is more than sufficient for most purposes, and assign all other space to /var/db, which is the database. /usr will be used to install most of the binaries and programs that do not come with FreeBSD and should have a sufficient amount of storage. The /var/db file system will host the MySQL database, so most drive spaces are assigned to it. When this is done, press Q. You might want to think real hard on how you want your partitions set up though.. it's also recommended to make a separate /usr/home partition as well as a separate /chroot partition. You can also press 1 or 2 to toggle between UFS1 and UFS2 types.
- When prompted for distribution sets, select Minimal config (move cursor to minimal and press space) and then select “Custom”. Most of the other components are not useful for database host and installing them will only increase the size of the OS and increase the burden to maintain them. At the “Custom” menu, select “man”. This is manual for all binaries, and a very important component for all administrators.
- When asked for configuration for Ethernet adapter, answer YES and select the “xl0” network card (which is the 3Com network card) or whatever your particular network card is.
- When prompted for IPv6 configuration, answer NO. There's no reason to use IPv6 yet.
- When prompted for DHCP configuration, answer NO.
- Then the IPv4 configuration screen will come up, this database host will have a temporary internal IP address for installation (it will be connected to a specially firewalled network segment for installation).
- When prompted for whether to bring up the network interface now, answer NO. (The network cable is not even in place, there is no point of bringing up the interface).
- When prompted for this host being a gateway, answer NO (There is no need for packet forwarding on this machine).
- Then the prompt for configuring inetd and simple Internet services will come up, answer YES.

- When asked whether to enable inetd, answer NO. There is no service on the database server host that will run with inetd.
- When asked whether this host is an anonymous FTP server, NFS server and NFS client, answer NO. These are all very high-risk server software and are not needed for the database server operation.
- At the prompt for choosing security profile, answer YES. This will allow a choice of security profile which can make the OS much more secure. Choose "Extreme" in selection for security profile. The extreme security profile defaults to not start with sshd and sendmail (unlike in medium) and it also sets the kernel securelevel to 2, which requires single-user mode for a lot of system modification (will slow down and/or deter hacking attempts).
- When prompted for time zone, select YES and choose your correct time zone.
- Answer NO to Linux Compatibility.
- When prompted for packages installation, answer NO. The packages in the CD may not be up to date and may require immediate update. Software installation will be done at a later stage.
- Answer YES to adding user to system, and then add a normal user to the system. When prompted for root password, enter a strong password.
- When offered to view the options again, answer YES and review all options before exiting the installation and rebooting.

First Time Configuration (Basic System Security Part 1)

Now that we have covered Installing FreeBSD, let's go a little into the initial first things to do and some basic securing of your new operating system. As usual, this is a **DO AT YOUR OWN RISK**. We take no responsibility in anything that may happen.

Once you first boot into FreeBSD, go ahead and login as the root User. Now let's find out what ports are presently open on your system and close the unneeded ones. You can find this out by issuing the socket status command.

```
($:~)=> sockstat -4
```

You should see a list of all the current daemons that are in effect and which ports they’re running from.

Let's start off by working with sendmail. You should notice both ports 25 and 587 belong to sendmail. First of all, we can completely close port 587, and I have no idea to this day why that is open by default. We can close this by going in and editing the /etc/mail/sendmail.cf file.

```
($:~)=> vi /etc/mail/sendmail.cf
```

(Depending on your experience, if you don't like vi you can use ee or something different)

now search for a line that states:

```
O DaemonPortOptions=Port=587, Name=MSA, M=E
```

Once you find that line, put a comment in front of it. Then save and close. Now execute:

```
($:~)=> killall -HUP sendmail
```

The -HUP won't stop sendmail, but will tell it to read the changes you made to /etc/mail/sendmail.cf. Repeat sockstat -4 and it should no longer show port 587.

What about port 25? You may or may not need to leave this port open, depending upon which program you use to send and read your e-mail. If you're running FreeBSD 4.6-RELEASE or higher, put this line in /etc/rc.conf:

```
sendmail_enable="NO"
```

This will tell sendmail to only listen on the localhost, which will allow any mail client to be able to send e-mail. If you know that your mail client has its own built-in SMTP agent or you're feeling adventurous, you can try this line instead:

```
sendmail_enable="NONE"
```

Which will close port 25 completely. To see if you've broken the ability to send e-mail, make sure you've closed all of your terminals and saved all of your work. Then, as the superuser:

```
($:~)=> shutdown now
```

Press enter when prompted, then type exit. Once you've logged back in, see if you can send a test message to your e-mail account. If you can't, go back to the word "NO" and repeat the above to re-open port 25 for the localhost.. This is all I'm going to go into on sendmail. I would suggest reading more on configuring sendmail at a later time.

If port 111 (portmap) shows up in your "sockstat" output, remove it by adding the following lines to /etc/rc.conf (or, if a line already exists in that file, change the YES to a NO):

```
nfs_server_enable="NO"
nfs_client_enable="NO"
portmap_enable="NO"
```

Portmap is only needed if you are running NFS, which you won't be on a stand-alone FreeBSD server. It also has a long history of security issues; so if you don't absolutely need it, disable it.

syslog (port 514) will probably also show in your output. You don't want to disable syslog completely, as you do want to receive logging messages. However, you don't need to have this port open to do so. In your /etc/rc.conf file, make sure syslog is enabled and add a second line

with some options:

```
syslogd_enable="YES"
syslogd_flags="-ss"
```

Those two ss'es (make sure you have two, not just one) in the flags will disable logging from remote hosts and close that port, but still allow your localhost to keep its logging capabilities. After doing this, do another shutdown now command, then once in singleuser mode, press control+D to return to regular mode.. Issue the sockstat command again and the syslog port should now be closed.

Next, make sure inetd_enable is not set to YES in /etc/rc.conf. If inetd is showing up in your sockstat output, something has been uncommented out in /etc/inetd.conf. If you don't need it, put a # back in front of that line, and do a killall inetd.

If you get your address from your ISP's DHCP server, keep dhclient (port 68) open, or you won't be able to renew your IP address.

Some other things to possibly add in your rc.conf are:

```
log_in_vain="YES"
```

If you didn't, it is a good option to include, as it logs all attempts to closed ports. This can get to be annoying after awhile though.

An interesting option is:

```
accounting_enable="YES"
```

This will enable system accounting. If you're new to system accounting, read man sa and man lastcomm to decide whether this option would be useful to you or not.

Finally, this is a good option to include:

```
clear_tmp_enable="YES"
```

as it will clear /tmp at startup, which is always a good thing.

Let's leave /etc/rc.conf and see what else we can do to tighten up your system. I like to change the default algorithm used when encrypting a user's password to the Blowfish algorithm, as it provides the highest security at the greatest speed.

To implement Blowfish hashes, edit /etc/login.conf and change the passwd_format line so that it looks like this:

```
:passwd_format=blf:
```

Also, add the following to set the password defaults. These changes will accomplish the following:

force the password change interval to 90 days; Warn the users to use mixed case passwords; The next change will set the minimum password length to 10 characters; And the final field will log the user out after an idle time of 30 minutes.

```
:passwordtime=90d:
:mixpasswordcase=true:
:minpasswordlen=10:
:idletime=30:
```

Save your change, and then rebuild the login database with this command:

```
($:~)=> cap_mkdb /etc/login.conf
```

You'll then have to change all of your user's passwords so they will get a new Blowfish hash. You can do this by typing:

```
($:~)=> passwd username
```

As the superuser, whatever username you use will be the user whose password will be updated. Repeat for all of your users, including the root account.

Once you're finished, double-check that it worked and you didn't forget any users:

```
($:~)=> more /etc/master.passwd
```

All of the passwords for your users should begin with **$2**.

Finally, configure the adduser utility to use Blowfish whenever you create a new user by editing /etc/auth.conf. Change the crypt_default line so that it looks like this:

```
crypt_default=blf
```

You've probably noticed when you log in to your FreeBSD system that your login prompt reminds you that you are running FreeBSD. And that after you log in, you receive the FreeBSD copyright information, which is followed by the version of FreeBSD and the name of your kernel, and finally, a useful (but rather boring) motd which again reminds you that you are running FreeBSD. You probably already know what version of FreeBSD you are running and might not want to share that information with the rest of the world. And the motd is a good place to remind the rest of the world that they shouldn't be messing with your system anyways.

You can edit /etc/motd to say whatever suits your purposes, be it anything from your favorite sci-fi excerpt to all the nasty things that will happen to someone if they continue to try to log in to your system. Below is a good example of a common motd.

```
* * * * * * * * * * * * * * W A R N I N G * * * * * * * * * * * * * *
THIS SYSTEM IS RESTRICTED TO AUTHORIZED USERS FOR AUTHORIZED USE ONLY.
UNAUTHORIZED ACCESS IS STRICTLY PROHIBITED AND MAY BE PUNISHABLE UNDER
THE COMPUTER FRAUD AND ABUSE ACT OF 1986 OR OTHER APPLICABLE LAWS.
```

```
IF NOT AUTHORIZED TO ACCESS THIS SYSTEM, DISCONNECT NOW. BY CONTINUING,
YOU CONSENT TO YOUR KEYSTROKES AND DATA CONTENT BEING MONITORED. ALL
PERSONS ARE HEREBY NOTIFIED THAT THE USE OF THIS SYSTEM CONSTITUTES
CONSENT TO MONITORING AND AUDITING. THE ADMINISTRATORS ALSO RESERVE THE
RIGHT TO CANCEL OR LOCK YOUR ACCOUNT AT ANY GIVEN TIME. ALL TERMS
DESCRIBED ABOVE ARE SUBJECT TO CHANGE WITHOUT ANY GIVEN NOTICE. IF YOU
DO NOT AGREE TO THESE TERMS LOGOUT NOW!
* * * * * * * * * * * * * W A R N I N G * * * * * * * * * * * * * *
```

If you want to remove the copyright info:

```
($:~)=> touch /etc/COPYRIGHT
```

To change the text that appears at the login prompt, edit /etc/gettytab. Find the line in the default: section that starts with

```
:cb:ce:ck:lc
```

Carefully, change the text between r : to whatever text you wish to appear. Double-check that you have the right amount of s and s and save your change. For example, my login prompt looks like this:

I'm a node in cyberspace. Who the hell are you?

login:

You can test your changes by going to another terminal and logging in.

Finally, even though you've edited your motd to remove your version and kernel information, by default FreeBSD will still re-add it to /etc/motd every time you log in. To prevent this behavior, add the following line to /etc/rc.conf:

```
update_motd="NO"
```

This change requires a reboot, so make sure you've first tested your previous changes and have saved all of your work on any other terminals.

No one (including you) should ever log in to your system using the root account. To prevent this from happening, edit /etc/ttys. Once you get past a page's worth of comments, you'll notice a section that goes from ttyv0 to ttyv8. Change the word "secure" on each of those lines to "insecure". This is a file you don't want a typo in, so double-check your changes carefully. Test your change by trying to log in as root on one of your terminals. You should receive a "Login incorrect" message.

Personally, I tend to use all nine terminals on my desktop. If you don't, you can also change the word "on" to "off" on some of the ttys in /etc/ttys. Remember to leave at least one terminal "on," or else you won't be able to log in, which will severely hamper the usefulness of your system. You'll also note that ttyv8 is "off" by default, which means you have to manually start an X Window session. If you'd like X to start automatically at bootup, change that "off" to "on."

Another good thing to do inside your ttys file the top part says:

```
console none unknown off secure
```

I like to completely change all the secure portions from this file to insecure. So what's that do? Basically, every time you launch into single user mode you have complete root access without a password. By removing secure you're going to force people to enter in your root password upon entering single user mode or be given the option to return back to multi-user mode. It should look like:

```
console none unknown off insecure
```

Changing all the tty lines will make sure that the root user cannot login to the console by himself. A normal user must login and use either **su** or **sudo** to gain root access.

Now that the basics are completed let's get into some fun stuff shall we?

The first thing to do is to elevate the normal user's group, making the normal user able to become root user later.

```
($:~)=> vi /etc/group
```

and edit the first line (after the # comments) which starts with wheel,

```
wheel:*:0:root becomes wheel:*:0:root,soupx
```

Where "soupx" is the normal user that was created at installation time. As a normal user, FreeBSD will only allow minimal privileges. No changes to the system configuration files are allowed. For the purpose of securing the system, configuration would have to be changed. Unix systems have a superuser account for the purpose of system configuration and maintenance. The name of this account is "root" - it is the God-mode account, and a root user can do anything to the system. Given this amount of privileges, it is considered to be dangerous to login using this account for day to day operation because the consequence of a simple mistake is just too great, therefore, it is much safer to operate or "use" the system with a normal user account and only become the "root" user as need arises. To allow this to happen the user must be a member of wheel group (group 0), which is where user "soupx" was assigned above.

Similarly, when an attacker performs reconnaissance on a system, the knowledge of which OS platform is running would be of great value. Sometimes, an attacker would send out of spec packets to a host and by the information returned, they would be able to determine the type of OS running (this usually teams up with port scan to produce more information). To avoid unnecessary information leaking to the attacker:

```
tcp_drop_synfin="YES"
```

Tip: This option would break RFC compliance. Do not use this on a web server.

This should be added to rc.conf, and will effectively tell the system to ignore all the TCP packets

with SYN and FIN flag set. Notice that the kernel has to be set with “TCP_DROP_SYNFIN” to activate this option (see below – kernel section). ICMP specification allows a type of packets for the router to tell a sending host the most optimized route to another host, that routing is done statically and there is only one router out to other parts of the network. This type of packets should not exist under normal conditions. If these packets are on the network, it should either be a network component error (maybe misconfiguration) or a local LAN attack. The danger of these packets lies in the redirection, if traffic were redirected to a hostile local LAN host, that host would be able to eavesdrop on all traffic to and from the server. To disable the effect of such packets and to log them, add:

In the rc.conf file.

```
icmp_drop_redirect=”YES”
icmp_log_redirect=”YES”
```

In rc.conf configuration, “log_in_vain” option was selected to log all abnormal connections. This is only a logging feature and does not eliminate the threat of information leaked. Normally, when a host sends a SYN packet indicating the intention to establish a connection, the receiving host would either send a SYN+ACK packet back to continue the connection or send an RST packet to notify that the port is not listening. By monitoring whether SYN+ACK or RST packet is in the reply, the attacker would be able to map out the opened ports on a host. On the other hand, if the target host does not send back anything, the sending host would wait till timeout before trying another port that would slow down the scanning process. FreeBSD has an option to disable sending back the RST packet for unopened ports. Edit the sysctl.conf by typing:

```
($:~)=> vi /etc/sysctl.conf
```

This file should only contain comments. After the comments, add the following lines:

```
net.inet.tcp.blackhole=2
net.inet.udp.blackhole=1
```

Tips: This is not a replacement for a firewall (packet filter). It should be used in conjunction with a packet filter and possibly be a failsafe mechanism for the firewall. With a proper firewall configured, any log produced by log_in_vain should be a warning that the firewall is failing or mis-configured.

While you’re in the sysctl.conf add in this:

```
security.bsd.see_other_uids=0
```

Please note if you're running versions earlier than 5.2 the name changed. The old name is:

```
kern.ps_shoallprocs=0
```

That will make it so users will only be able to display the processes that are owned by that user. But root will still be able to display all processes. (This option first appeared in FreeBSD 4.7)

Updating Your Server:
Before we get into too much else, let's first make sure we have the most current sources and binaries on our system. We will do this through cvsup. To build a secure server, it is essential that the computer is free from known vulnerability. After securing the base OS, the database server should be updated to ensure it is free from known vulnerability. One of the biggest strengths of FreeBSD is the ability to update the whole OS from source code; this makes updating the OS an easy task. It can save FreeBSD administrator a lot of time tracking down bug fixes and patches.

There are a couple ways to get cvsup installed. And there are also 2 cvsup's, depending which one you want. You have cvsup (this one has a GUI for xwindows) or you have cvsup-without-GUI (this is for a system w/out xwindows) but for this article, I'm going to use the no-GUI one.

One way you can install cvsup is with the packaging system. We will go into detail about the packaging system and ports tree at a later time.

```
($:~)=> pkg_add -r cvsup-without-gui
```

If the package system is not your cup of tea you can also use the ports tree. If you've never used the ports tree before, this is what makes FreeBSD such a great operating system. So if you installed the ports collection you can:

```
($:~)=> cd /usr/ports/net/cvsup-without-gui ; make install clean
```

However, if you followed my instructions earlier, you did not install the ports collection because we want to use fresh ports. So now we go onto method 3 of how to install this.

```
($:~)=> /stand/sysinstall
```

Welcome back to the installer app. Normally I try to stay away from automated things like this to do my bidding. But I'm not fully going to go into the packaging system yet.

- Choose the 4th Option labeled Configure
- Choose the 2nd Option labeled Packages
- Choose the 1st Option labeled CD-ROM

Now pop your FreeBSD cd1 into the cd-rom drive and select your cd-rom from the menu

Scroll down to net and select cvsup. Once you're done installing cvsup you can continue to update your sources. You can also install from the FreeBSD ftp server instead of using the old outdated binaries from the cd-rom.

Create configuration file for CVSup:
CVSup can update all the source code provided by FreeBSD or just download and update ones that the user specifies. All of these are controlled by a configuration file provided at run time. I'm not going to go in-depth on cvsup - if you want more information about it I suggest you read the handbook.

Create a file in your /root directory called cvs-supfile. This file will contain the information required

for updating your system.

```
($:~)=> vi /root/cvs-supfile
```

Put this in it:

```
*default host=cvsup11.FreeBSD.org
*default base=/usr
*default prefix=/usr
*default release=cvs
*default delete use-rel-suffix
*default compress

src-all tag=RELENG_5_2
ports-all tag=.
```

Please remember to find your fastest cvsup server and change the release tags to the appropriate values. Now save and exit. Congratulations ! You have a cvs-supfile. Now run this as root:

```
($:~)=> /usr/local/bin/cvsup /root/cvs-supfile
```

Building Your World:
The source tree and ports tree will be synchronized with the FreeBSD distribution server.

So once your sources finished updating, let's go ahead and compile and install them.

```
($:~)=> cd /usr/src
($:~)=> make buildworld
```

The buildworld is going to take a long time. This is where the requirement of beer comes into play because unless you want to sit there and watch a bunch of code fly across your screen, a good alternative is to invite some friends over and play some good drinking games. Once buildworld completes, let's go ahead and setup a kernel. But keep in mind over time I have come up with a rule that if you're seeing double you probably should not be operating your server.

```
($:~)=> cd /usr/src/sys/i386/conf
($:~)=> cp GENERIC KERNNAME (kernname is whatever you wish to
call your kernel)
($:~)=> vi KERNNAME
```

Before editing your kernel I would suggest you read up about it, in the FreeBSD Handbook there are lots of options you should probably do inside your kernel, but I'm only going to suggest a couple to begin with.. You might also want to comment out the devices you don't have or need.

```
options TCP_DROP_SYNFIN
options SC_DISABLE_DDBKEY
options SC_DISABLE_REBOOT
options CPU_ENABLE_SSE
options CPU_ATHLON_SSE_HACK
```

```
options IPSEC
options IPSTEALTH
options TCP_DROP_SYNFIN

#Misc Console Settings
options VGA_WIDTH90
options SC_PIXEL_MODE
options VESA
```

If you add in the console settings above you should add the following into your /etc/rc.conf

```
font8x8="swiss-8x8"
font8x14="NO"
font8x16="swiss-8x16"
allscreens_flags="VGA_90x60 red black"
```

Exchange "swiss-8x8" with whichever font you prefer. You will find all available fonts under the /usr/share/syscons/fonts/ directory.

Back inside the kernel, we should also choose the type of firewall we would like to make our server use.

```
options IPFILTER
options IPFILTER_LOG
options IPFILTER_DEFAULT_BLOCK
```

These options will enable the IPF Firewall.

```
options IPFIREWALL
options IPFIREWALL_VERBOSE
options IPFIREWALL_DEFAULT_TO_ACCEPT
```

Where as these options will enable the IPFW firewall. If you plan on using OpenBSD's PF firewall, go ahead and add these in.

```
device bpf #DEFAULT
options PFIL_HOOKS
options RANDOM_IP_ID
```

And if using PF add these into your rc.conf file

```
pf_enable="Yes"
pf_logd="Yes"
pf_conf="/usr/local/etc/pf.conf"
```

After doing your buildworld later on, be sure to install /usr/ports/security/pf. PF is the way to go, in my opinion. For more information about PF and ALTQ, Solarflux has setup some nice information on this over at: http://solarflux.org/pf. I recommend visiting it and take the time to learn how to use

it. Another fun and interesting kernel option to add in is: (NOTE: the option below is for the 4x branch. It's standard in 5x)

```
pseudo-device snp 4
```

The 5x branch does not need this. But it must be specified in the 4x Branches. Also, if you're running the 4x branch, you need to create some devices by hand. And snp would be one of them.

```
($:~)=> cd /dev
($:~)=> ./MAKEDEV snp0
($:~)=> ./MAKEDEV snp1
($:~)=> ./MAKEDEV snp2
($:~)=> ./MAKEDEV snp3
```

So what does this allow me to do?

That's a good question. The watch command works in conjunction with the pseudo-device snp. Basically, it will allow you the root user to snoop tty's on your server, and even interact with users or take over their shells. The 4 means that you can snoop on 4 different tty's at any given time. You can change this to whatever value you would like, just remember to make the devices for them. Read man watch for more information about this.

You can check your LINT file for other options you wish to have. If there is no LINT file present, just issue the command make LINT, and it will create one for you. However, on the 5x branch you might get more of an understanding if you read the NOTES instead of LINT, since when you create a LINT it basically copies the NOTES file and strips the comments.

After you edit your kernel head on over to /usr/src.

```
($:~)=> cd /usr/src
($:~)=> make buildkernel KERNCONF=KERNELNAME
```

Hopefully everything will go well there. The kernel is ready to be installed at this point. You can ultimately issue the command:

```
($:~)=> echo "KERNELNAME" >> /etc/make.conf
```

This will make it so that each time you build a new kernel, you no longer have to specify the name for it as well as installing the kernel.

```
($:~)=> make installkernel KERNCONF=MYKERNEL
```

This will install the kernel.

To activate the kernel, a reboot is necessary. As root user, type the command

```
($:~)=> reboot
```

The system will reboot at this stage. When the reboot is finished, the system is running on the new kernel. You can issue a uname –a command to verify that you're operating on a new kernel.

Now to install compiled binaries. The source tree was compiled earlier. Now that the kernel is compiled and installed, it is ready for the binaries to be installed. For this to happen, the system has to be in single user mode,

```
($:~)=> shutdown now
```

After the system gets into single user mode,

```
($:~)=> cd /usr/src
($:~)=> make installworld
```

This will install all the updated binaries compiled from source downloaded by CVSup, and the system is then ready to be rebooted.

```
($:~)=> reboot
```

You might want to look around at various file permissions once you're finished. One thing I've experienced is that /tmp likes to change permissions on you. So remember to look at that. Another example is the master.passwd file. You should chmod it to 600 - nobody but root needs to see that file. We will go into these files and permissions later though.

Now that you know how cvsup works it's handy to set up a cron job to do this nightly. Anyhow, this should get you on your feet.

```
($:~)=> vi /etc/crontab
```

And add the following in there:

```
0 3 * * * root /usr/local/bin/cvsup /usr/local/etc/ports-supfile
1>/dev/null 2>&1
```

So, what's all this mean? Crontabs are pretty simple but commonly overlooked by system administrators. An easy breakdown of it is:

```
MIN HOUR DAY MONTH WDAY WHO CMD
```

So we're saying at 3am every day we are going to issue the following command as our root user. When updating your operating system it's also strongly recommended to read the /usr/src/UPDATING file for any changes that might have occurred.

Change permission of root directory. There should not be any normal user other than system administrators logging onto the system, but just in case this happens, the root directory should be properly protected. To avoid unnecessary monitoring by other users on the system.

```
($:~)=> chmod 700 /root
```

You might also want to look at the permissions in the users home directories. I would recommend changing them to 700 also.

Change permission of suid and sgid binaries. Suid binaries allows a user to execute the program as a different user (usually root). Sgid works in similar fashion and allows the user to become another group. Some of the suid binaries are badly implemented and easily exploited; they could easily lead to a local user compromising the machine through these suid and sgid binaries. The best practice is to use suid and sgid binaries only if necessary and disallow the use of the unnecessary ones. To find all suid binaries on a machine:

```
($:~)=> find / -perm –4000
```

To find all sgid binaries on a machine:

```
($:~)=> find / -perm –2000
```

Some binaries are almost for sure never to be touched. For those binaries, permission 000 should be given, this will disallow any read, write and execute from any user. For the binaries that are only useful to root, permission 500 should be given and should be owned by root, it would only allow root to execute and read them. Later in this document we will go over how permissions are defined.

Now I'm going to help you out a little bit here and show you an excellent application for managing suid and sgid binaries.

```
($:~)=> cd /usr/local/src ; fetch http://www.watson.org/fbsd-
hardening/suidcontrol-0.1.tgz
($:~)=> tar –xzvf suidcontrol-0.1.tgz
($:~)=> cd suidcontrol ; make
```

Please take the time to read the README file and learn how to use this great application. A good example of a hardened system policy file can be viewed over here: http://www.watson.org/fbsd-hardening/lockdown.pol

```
# cut-n-paste fromhttp://www.watson.org/fbsd-hardening/lockdown.pol
#
suid-binaries
{
        entry cu suid-disable;
        /* DEFAULT: entry man suid-enable; */
        entry uucp suid-disable;
        entry suidperl suid-disable;
```

```
        entry at suid-disable;
        /* DEFAULT: entry passwd suid-enable; */
        entry skey suid-disable;
        entry login suid-disable;
        entry quota suid-disable;
        entry rsh suid-disable;
        entry crontab suid-disable;
        /* DEFAULT: entry su suid-enable; */
        entry lpr suid-disable;
        /* DEFAULT: entry mail suid-enable; */
        entry kerberos suid-disable;
        entry multicast suid-disable;
        entry pppslip suid-disable;
        entry timedc suid-disable;
        /* DEFAULT: entry ping suid-enable */
        entry route suid-disable;
        /* DEFAULT: entry shutdown suid-operator */
}

sgid-binaries
{
}

inetd-daemons
{
        entry telnet inetddaemon-disable;
}
#
# end cut-n-paste
```

Mounted drives:
Some local directories should never encounter certain types of files, such as suid. Such types of files running on directory such as /tmp is certainly abnormal and should be stopped. Fortunately, FreeBSD allows mount options that will limit the type of operation on a mount point. Even though no user should be allowed to log into database host system to execute anything or run any files, a precaution measure is always helpful. To set the allowed operation on a mount point:

```
($:~)=> vi /etc/fstab
For /tmp's option, change to rw,nosuid,nodev
For /usr's option, change to nodev,rw
For /var's option, change to nodev,nosuid,noexec,rw
For /home option, change to nodev,nosuid,noexec,rw
```

If you're using prior to FreeBSD 5, you might also want to consider disabling automount of the Procfs. I would suggest you read up on it to see if it's right for you.

The "nodev" option disallows files to be a device, avoid unnecessary access to hardware devices. The "nosuid" option disallows files on the specified file system to run as suid binaries. The "noexec" option disallows execution of any files on the specified file system. By limiting the

capabilities of files in different file system, normal user's ability on the system is reduced. Even if an attacker gained access as a normal user on the system, it would be harder to exploit local vulnerabilities. Please keep in mind with the above configurations that when you update your system in the future you're going to have to limit these restrictions to be able to perform a buildworld with out errors. That is the key reason to drop down into single user mode while doing such tasks.

Now, if you didn't notice yet, your system has 2 temporary placement directories. You have /var/tmp and /tmp. So ideally we don't really need both of these, but certain services use /var/tmp for writing stuff to. So we're going to delete the /var/tmp directory and create a symbolic link to /tmp

```
($:~)=> cd /var && mv ./tmp/* /tmp/ && rm -rf tmp && ln -s /tmp
tmp
```

Another good security-related aspect of FreeBSD permissions is the chflags command. We will talk about this in the next section when we go over permissions. But there are certain files across your system that should never be touched. So we're going to add a little bit of flavor to these files.

```
($:~)=> chflags schg /bin/*
($:~)=> chflags schg /sbin/*
($:~)=> chflags schg /usr/sbin/*
```

On some of these files you will get an error message saying that the operation could not be permitted. That's fine, just ignore them.

Moving on, we need to change some permissions in /var/log. The permissions set by default tend to be a bit too liberal for my taste.

Remove any logs you don't use (refer to /etc/syslog.conf to see what you actually use) and chmod the files 600 (except the file wtmp which may be sane to chmod 640 so at least the wheel user may be able to use the 'last' command).

```
($:~)=> cd /var/log
($:~)=> chmod 600 *
($:~)=> chmod 640 wtmp
```

Finally, create an empty file called /etc/ipf.rules and an empty file called /etc/ipnat.rules so IP Filter and IPnat sees the configuration files you specified in /etc/rc.conf.

```
($:~)=> touch /etc/ipf.rules
($:~)=> touch /etc/ipnat.rules
($:~)=> chmod 600 /etc/ip*.rules
```

Across our system there are various files that no normal user should be able to view or access. This is where the default permissions, in my opinion, are wrong. So let's quickly change some of these.

```
($:~)=> chmod 600 /etc/crontab
($:~)=> chmod 700 /root
($:~)=> chmod 700 /home/*
($:~)=> chmod 650 /etc/rc.*
($:~)=> chmod 600 /etc/master.passwd
```

These are just a couple prime examples. There are many more files that you should change permissions to. Something to keep in mind though is that each time you perform an install world some of the permissions change back to their original default values.

Now onto SSH. OpenSSH is a remote login program that will enable a person to log into your box from anywhere. But like everything it needs some tweaking. So let's go ahead and edit it’s configuration file.

```
($:~)=> vi /etc/ssh/sshd_config
```

The key lines to change or add are the following:

```
Port 22
Protocol 2
PermitRootLogin no
X11Forwarding no
PrintMotd yes
PrintLastLog yes
KeepAlive yes
PermitEmptyPasswords no
PasswordAuthentication yes
ReverseMappingCheck no
GatewayPorts no
AllowTcpForwarding yes
AllowGroups ssh
Banner /etc/issue
MaxStartups 10
```

Notice that we're only going to allow users login that are in the ssh group. let's go ahead and create one.

```
($:~)=> echo “ssh:*:666:soupx” >> /etc/group
```

Please remember to change "soupx" to your NORMAL user name. If you decide not to, can you at least e-mail me my password?

A good measure to take now is to backup various system directories. We're going to take /etc for an example. It’s a good idea to keep an active backup on your system for quick measures of restoring a file that might accidentally be borked.

```
($:~)=> tar –cvvpzf /root/etc.tar.gz /etc
($:~)=> chmod 600 /root/etc.tar.gz
```

For an extra layer of protection let's encrypt that file:

```
($:~)=> openssl enc -blowfish -in /root/etc.tar.gz -out
/root/etc.tgz.bf
```

To unencrypt the file you would issue:

```
($:~)=> openssl enc -d -blowfish < /root/etc.tgz.bf | tar -xzf -
```

Once encrypted, remove the etc.tar.gz file.

Onto further tweaking. Let's go back and edit the rc.conf file again.

Do not turn on RFC1323 extensions:

```
tcp_extensions="NO"
```

Disable probing of idle TCP connections to verify the peer is up and reachable:

```
tcp_keepalive="YES"
```

Do not respond to broadcast ping packets:

```
icmp_bmcastecho="NO"
```

ICMP error response bandwith limiting. Helps protect against DoS attacks:

```
icmp_bandlim="YES"
```

And now jump back into the /etc/sysctl.conf file.

To verify that an incoming packet arrives on an interface that has an address matching the packet's destination address:

```
net.inet.ip.check_interface=1
```

Increase TCP Window size for increase in network performance:

```
net.inet.tcp.recvspace=65535
net.inet.tcp.sendspace=65535
```

Change the ELF Branding fallback:

```
kern.fallback_elf_brand=3
```

Block SYN Cookies and other ICMP Stuff:

```
net.inet.tcp.syncookies=0
```

```
net.inet.icmp.bmcastecho=0
net.inet.icmp.maskrepl=0
net.inet.icmp.icmplim=200
```

How To Install a Secure BSD System - Part 2

Unix File Permissions:
One aspect of a Unix system security is putting control on users. There are several different types of controls that can be used, including file permissions, file attributes, file system quotas and system resource limits.

Unix file permissions are a way to allow a user to restrict access to a file or directory on the file system. For files, a user can specify who can read the file, who can write to the file and who can execute the file. For directories, a user can specify who can read the directory, who can write to the directory and who can execute programs located in the directory.

Files:
Let's look at a simple example of a file.

```
($:~)=> ls -a example.txt
-rw-rw-r-- 1 soup4you2 soup4you2 852B Feb 22 00:26 example.txt
```

Here we execute ls -l. The ls command lists the contents of the directory, or in this case, only the file example.txt the -l option lists the file in long form, which displays quite a bit of information about the file. The output lists the following info:

```
-rw-rw-r-- 1 soup4you2 soupx 852B Feb 22 00:26 note.txt
```

Here's a breakdown of what's being displayed here:

-rw-rw-r-- Permissions
1 Number of links
soup4you2 File Owner
soupx Group
852B File size
Feb 22 00:26 Date last modified
example.txt Filename

Notice that this file has one owner (soup4you2) and belongs to one group (users). The owner and group are important when we discuss file permissions.

The file permissions are as follows:

-rw-rw-r--

The information is divided into four parts

```
- File Type
rw- Owner Permissions
rw- Group Permissions
r-- World Permissions
```

The first part of the output is the file type. Common file types are as follows:

```
- A Normal File
d A Directory
l A Symbolic link
s A Socket
```

Following the file type are three groups of these characters representing the permissions for the owner group and world. Three characters indicate whether or not permission is granted to read the file (r), write to the file (w), or execute the file (x). If permission is granted, the letter is present. If permission is denied, the letter is, well, you guessed it sparky, not there and a (-) in it's place. Here's another example

rwxr-x--x

The first three characters are the permissions for the owner. The permissions rwx indicate that the owner can read the file, write to the file and execute the file. The next three characters are the permissions for the group associated with the file. The next three characters are the permissions for the group associated with the file. The permissions r-x indicates that members of the group can read the file and execute the file but cannot write to the file. The last three characters are the permissions for the rest of the world. Cannot read the file and cannot write to the file but can execute the file. Getting this yet? It only gets more fun from here.

Take notice that the three permissions are either granted or denied, either on or off (I've got to get specific.. you might be a slobbering drunk who needs all the help they can get). Since the permissions can be considered either on or off the permissions can be thought of as collection of 0s or 1s. For instance "rwx" has read, write and execute permissions on. Therefore we can write these permissions as "111", and in octal format the value is 7. Similarly "r-x" has read permissions and execute permissions on and write off. Therefore we can write these permissions as 101 and in octal format the value is 5.

If we put this idea in practice for owner/group/world permissions, then the permissions rwxr-x--x in binary format are 111101001, and if we treat this as a series of three groups of octal numbers, the value is 750

Confused yet? Let me try to clarify just a little bit here.

Read Permissions - you should give a value of **4** Write Permissions - you should give a value of **2** Execute Permissions - you should give a value of **1**

So if you want read+execute permissions, add 4 + 1 and get 5. You do this for each of the 3 parts

user/group/other

U = User G = Group O = Other (not U or G)

Triplet for u: rwx => 4 + 2 + 1 = 7
Triplet for g: r-x => 4 + 0 + 1 = 5
Tripler for o: r-x => 4 + 0 + 1 = 5
Which makes : 755

-U- --G-- --O--
-rwxr-xr-x 1 nick users 382 Jan 19 11:49
-drwxr-xr-x 3 nick users 1024 Jan 19 11:19 lib/
-rwxr-xr-x 1 nick users 1874 Jan 19 10:23 socktest.pl

Everything goes by 4 2 1

Changing File Permissions:
The chmod command changes file permissions. You would use it like so:

chmod mode file [file ...]

To see how to use chmod, let's look at a file lying around.

```
($:~)=> ls -l a.txt
-rw-rw-r-- 1 soupx users 10 Sep 03 06:50 a.txt
```

To change the permissions to an explicit mode, use the octal method:

```
($:~)=> chmod 751 a.txt
($:~)=> ls -l a.txt
-rwx--x--x 1 soupx users 10 Sep 03 06:50 a.txt
```

Do you see how the permissions 751 translated into rwxr-x-x. And look at this if you're feeling saucy enough.

```
($:~)=> chmod 640 a.txt
($:~)=> ls -l a.txt
-rw-r----- 1 soupx users 10 Sep 03 06:50 a.txt
```

Here, 640 translates to rw-r----- You can also use the chmod command in symbolic mode as follows (even though I don’t care too much for this way):

```
($:~)=> chmod +x a.txt
($:~)=> ls -l a.txt
-rwxr-x--x 1 soupx users 10 Sep 03 06:50 a.txt
```

This time, chmod is used with +x which means to add executable permissions where the + character is used. It means to add the permissions whereas - character means to remove. Here,

+x means to add executable permissions for the owner, group and world. The chmod command can also be used to change permissions for a specific group:

```
($:~)=> chmod g-r a.txt
($:~)=> ls -l a.txt
-rwx--x--x 1 soupx users 10 Sep 03 06:50 a.txt
```

This shows chmod being executed with g-r which means "remove group executable permissions." Are you getting tired of my bad spelling yet, or have you drunk enough to where you just don't care anymore? (proofreader thinks, "Yes", even though spelling/grammar corrections were made, and opens another brewski)

Sticky Bits:
If a user has write permissions to a directory that user can delete files and directories within, even if those files are not owned by the user and permissions are set so that the user cannot read or write the file:

```
($:~)=> ls -ld temp
($:~)=> ls -l a.txt
drwxrwxrwx 1 soupx users 10 Sep 03 06:50 temp
```

We see here that the temp directory is owned by "soupx", yet writeable by the world. This is bad because somebody else not in the group or owner can delete the file even though they cannot see it. Since I'm a few beers into this, I'm going to skip right along to our next topic.

Default Permissions and Umask:
When a user creates a file or directory, that file or directory is given default permissions:

```
($:~)=> touch a.txt
($:~)=> mkdir testdir
($:~)=> ls -l
total 1
-rw-rw-r-- 1 soupx users 10 Sep 03 06:50 a.txt
drwxrwxr-x 1 soupx users 10 Sep 03 06:50 testdir
```

Notice that the default permissions for the user soupx are 644 for files and 775 for directories. Default file and directory permissions are set according to the value of the users umask value. The umask value is due to mask off bits from the most permissive default values, 666 for files and 777 for directories. To display your umask, just type in umask

```
($:~)=> umask
002
```

The user soupx has a umask value of 002. An easy way to determine the value of soupx's default permissions when soupx creates a file or directory, is to simply subtract the value of umask from the system default permission values.

```
Files: Directories
```

```
666 777
002 002
664 775
```

To change your default permission change your umask value to create the most restrictive permission, use a value of 777 which would give out the equivalent of 000 permissions. Of course, this is too restrictive since soupx does not have read and write permissions for new files (pretty lame, ehh?). To create files and directories with the most practical restrictive permissions, use a umask value of 077 which will grant read/write/execute for the owner but nothing for the group or world. So it would retain file permissions values of 700. If you wish to change your umask value permanently, add it in to your users login profile.

The easiest way to figure out what umask to use is to take the octal number and subtract it from 777. So, if you wanted all new files to be created in your directory as 754 , subtract that from 777, 777-754=023.

A default mode of 755 would be 777-755 = 022. This is the default umask on most systems as far as I know.

File and directory Flags:
The chflags utility comes in very handy when dealing with system security. This utility will place various restrictions on a file on top of their permission values.

Flags are a comma-separated list of keywords that are defined as:

sappnd - set the system append-only flag (superuser only)
This is the system append-only flag. And can only be set and removed by the root user. Files with this flag can be added but cannot be removed or otherwise edited (a particularly useful feature for log files). Setting sappnd on a .history file can be interesting in the event of a system compromise. A popular script kiddie trick is to remove .history or link it to /dev/urandom so that the sysadmin cannot see what was done, but sappnd prevents this tactic from working without changing any file system permissions on the .history file itself. This flag cannot be removed when the system is at securelevel 1 or higher.

schg - set the system immutable flag (superuser only)
The system-level immutable flag can only be set or removed by root. Files with this flag set cannot be changed in any way: not edited, not moved and not replaced. Basically, the kernel will prevent all attempts to touch this file. This flag cannot be removed when the system is running at securelevel 1 or greater.

uappnd - set the user append-only flag (owner or superuser only)
Only the file owner or root can set the user append-only flag. Like the system append-only flag, sappnd, a file with this flag set can be added to but not otherwise edited or removed. This is most useful for logs from personal programs and the like and is primarily a means users can employ to prevent vital files from accidental removal. The user or root may perform this flag at any time.

uchg - set the user immutable flag (owner or superuser only)
Only the owner or root can set the user's immutable flag. Like the schg flag, the user immutable flag prevents a user from changing the file. Again, root can override this, and the user at any securelevel can remove it. An immutable file may not be changed, moved or deleted. An append-only file is immutable except that data may be appended to it. Putting the letters "no" before a flag name causes the flag to be turned off.

For example:

nouchg the immutable bit should be cleared.

The superuser-settable "sappnd" and "schg" flags can be set at any time, but may only be cleared when the system is running at security level 0 or -1 (insecure or permanently insecure mode, respectively). The securelevel is normally set to 0, for example, when running in single-user mode.

Symbolic links do not have flags, so unless the -H or -L option is set, chflags on a symbolic link always succeeds and has no effect. The -H, -L, and -P options are ignored unless the -R option is specified. In addition, these options override each other and the command's actions are determined by the last one specified.

Only the superuser can change the user flags on block and character de-vices. You can use ls -lo to see the flags of existing files. These are just a couple of the flags available. I recommend reading man chflags more for a list of possible flags and their meanings.

The information about chflags was taken from the chflags manpage. And other absolute OpenBSD

Mail::Toaster stuff to be edited below..... ???

Installing and Configuring Mail Services:
If you are after one badass secure mail server, then this section is for you. Sendmail is installed by default on FreeBSD systems and unless configured properly, it is extremely insecure. So what I'm going to attempt here is setting up:

- Postfix With Qmail Maildir support
- Courrier IMAP / POP3 with IMAP and POP though SSL
- Procmail
- SPAMassasin
- Amavis Virus scanning
- Virtual domain support
- Cyrus SASL/TLS authentication, Along With Postfix SSL

I will go into configuring most of these. But for some of these, you're going to have to do a bit of tweaking and playing around.

Installing and Configuring Postfix:
Let's start off by installing our new mail server postfix. Use the ports luke.

```
($:~)=> cd /usr/ports/mail/postfix
($:~)=> make install clean
```

To avoid any confusion with other applications that use postfix we're going to make a quick symbolic link.

```
($:~)=> ln -s /usr/local/etc/postfix /etc/postfix
```

Now let's configure this puppy. Thanks to elmore at screamingelectron.org for providing the configuration we will be using here. We'll start off by going into your postfix directory and editing main.cf with your favorite editor.

```
($:~)=> vi main.cf
```

The first thing you're going to modify is the following line:

```
"myhostname = your.servername.here"
```

This line will set the mail server host name of your box. This cannot be any arbitrary name; it MUST be a fully qualified domain name!!

The next line to modify is:

```
"mydomain = some.domain.name"
```

Again this sets the default domain of the box and must be a Fully Qualified Domain Name Here!

The next line to look at is the following:

```
"myorigin = $mydomain"
```

Make sure this variable is set correctly! "$mydomain" is a valid global variable for the file so you should be good with that!

The next line to modify is:

```
"mydestination = $myhostname, localhost.$mydomain $mydomain
virtualdomain1 virtual domain2"
```

Obviously, this example assumes you'll be setting up virtual domains. It also assumes you'll be setting up sendmail style virtual domains. I know the file says to use the virtual file instead, but I don't do that. I specify here, sendmail style, and if do not want to do that that's fine with me. Just

understand in this how-to that we'll be setting up sendmail style virtual domains.

Moving On, the next line to modify is the following:

```
"home_mailbox = Maildir/"
```

DO NOT uncomment this line if you do not intend to use imap. If you are using pop or traditional /var/mail spool to deliver mail this is not needed and will undoubtedly mess up your mail delivery. If you are using courier-imap, this line needs to be uncommented!

Moving right along, the next line to modify is the:

```
"relay_domains = $mydestination, 127.0.0.1"
```

This will only allow messages to be accepted where the final destination domain is on this box. It also allows delivery over the loopback interface!

Now, modify the following:

```
"mynetworks = xxx.xxx.xxx.xxx/zz, 127.0.0.1"
```

This only allows mail to be sent out to the Internet from the specified ips! This can be an entire subnet or just one box - you decide.

This brings us to the next lines to modify:

```
"local_destination_concurrency_limit = 2"
"default_destination_concurrency_limit = 10"
```

These lines limit the amount of concurrent connections to a domain. Good to have especially if you have a user that is forwarding mail out of his home via the use of a **.forward** file.

The next lines I actually insert into the file. They are for the canonical maps. You'll need these if the following is true:

You have virtual domains
You have virtual usernames like full.name@somehost.com mapped back to username@somehost.com

If you're using one of the above, add it in. If not, don't worry about it!

```
"sender_canonical_maps = hash:/etc/postfix/canonical"
"recipient_canonical_maps = hash:/etc/postfix/canonical-receive"
```

We'll be going over canonical tables in more depth in a little while. For now what you need to know is that on table modifies, the incoming mail and one will modify outgoing mail.

The following lines I also add - you may or may not want to add these, depending on how true you want to be to the rfc and how strict you want to be on other hosts trying to send mail to you! These lines will lay down the framework for cutting down on your SPAM!

```
smtpd_client_restrictions = reject_rbl_client,
check_client_access hash:/etc/postfix/client_access,
reject_unauth_pipelining

smtpd_recipient_restrictions =
regexp:/etc/postfix/regexp_access, check_recipient_access
hash:/etc/postfix/access,
permit_mynetworks,reject_unknown_recipient_domain,
reject_unknown_hostname, reject_rbl_client,
reject_unauth_pipelining, reject

smtpd_sender_restrictions =
regexp:/etc/postfix/sender_checks.regexp, check_sender_access
hash:/etc/postfix/sender_access, reject_unknown_sender_domain,
reject_non_fqdn_sender, reject_rbl_client,
reject_unauth_pipelining
```

(That should be 3 lines. Thought I would mention that to avoid future frustrations)

Ok, the first section specifies rejection if a client is not in an access list. An access list is a list I use which details the exact usernames on the box. This list is a necessity if you are running virtual domains. If you don't use it userid1@virtualdomain1 can receive an e-mail to userid1@virtualdomain2 and so on and so forth. This is the most efficient way I know to block this! It is also a good practice to do even if you aren't running virtual domains (opinion). It also specifies a lookup to the rbl list (a real time black hole list for open-relay mail servers). It also does not allow unauthorized pipelining, not exactly sure why that's needed, but it is. If you know, let me know!

The second section does much of the same but is for outside connections (people trying to send mail in). It specifies a regular expression file which sorts through the headers and looks for junk, the access list, and rejects mail from computers that don’t have a fqdn. It also rejects if it can't get the computer hostname through nslookup, and rejects via the rbl list and the pipelining again!

Ok, the next section is a lot more of the same. Nothing really new to explain here!

The next line I add is my rbl line, defining what list to use. I personally use the ordb list which can be found at http://ordb.org (I like it a lot and it's free!)

```
"maps_rbl_domains = relays.ordb.org"
```

The next things added are the following body and header checks. They specify lookup files to run against all incoming and outgoing messages!

```
"header_checks = regexp:/etc/postfix/header_checks"
```

```
"body_checks = regexp:/etc/postfix/body_checks"
```

The final things I add are a couple of reject codes and a message size limit (self-explanatory). Here they are!

```
"unknown_hostname_reject_code = 554"
"unknown_client_reject_code = 450"
"message_size_limit = 5000000"
```

After the SMTP helo is sent, the client needs to tell Postfix who the e-mail is from (MAIL FROM) and where to send to (RCPT TO). This communication is supposed to follow RFC-821. Some SPAM software is not strict about its conformance, so we can block SPAM based on this fact.

```
strict_rfc821_envelopes = yes
```

With this done, we are now through with the main.cf file. Save it and let's move on to our next section, configuring the rest of postfix! A keynote here - once you finish editing files you need to cap it. Don't forget to do this whenever you edit your postfix configuration files.

```
($:~)=> postmap main.cf
```

Cyrus SASL/TLS And Postfix SSL:

Let's get SASL2 Installed now.

```
($:~)=> cd /usr/ports/security/cyrus-sasl2 ; make install clean
```

Now go ahead and edit postfix's main.cf so we can tell it to start utilizing the TLS features. Add in the following somewhere near the bottom:

```
#TLS
smtp_use_tls = yes
smtpd_use_tls = yes
smtpd_tls_auth_only = yes
smtp_tls_note_starttls_offer = yes
smtpd_tls_key_file = /etc/postfix/ssl/post.pem
smtpd_tls_cert_file = /etc/postfix/ssl/post.pem
smtpd_tls_CAfile = /etc/postfix/ssl/post.pem
smtpd_tls_loglevel = 3
smtpd_tls_received_header = yes
smtpd_tls_session_cache_timeout = 3600s
tls_random_source = dev:/dev/urandom

enable_sasl_authentication = yes

smtpd_sasl_auth_enable = yes
smtpd_sasl_security_options = noanonymous
smtpd_sasl_local_domain =
```

```
broken_sasl_auth_clients = yes
```

Also add this in the beginning of your recipient restrictions

```
($:~)=> permit_sasl_authenticated,
```

Here we create our postfix SSL Stuff

```
($:~)=> mkdir /usr/local/etc/postfix/ssl
($:~)=> chmod 700 /usr/local/etc/postfix/ssl
```

Next we create our SSL certificates for postfix

```
($:~)=> cd /usr/local/etc/postfix/ssl
($:~)=> vi pst.cnf
```

The contents of pst.cnf are:

```
RANDFILE = /etc/postfix/ssl/post.rand

[ req ]
default_bits = 1024
encrypt_key = yes
distinguished_name = req_dn
x509_extensions = cert_type
prompt = no

[ req_dn ]
C=countryName Two letters!
ST=stateOrProvinceName
L=localityName
O=organizationName
OU=OrganizationalUnitName
CN=commonName
emailAddress=emailAddress

[ cert_type ]
nsCertType = server
```

Be sure to enter the correct options. Next we generate our SSL certificates.

```
($:~)=> dd if=/dev/urandom of=/etc/postfix/ssl/post.rand count=1
2>/dev/null

($:~)=> /usr/sbin/openssl req -new -x509 -days 365 -nodes
-config /etc/postfix/ssl/pst.cnf -out /etc/postfix/ssl/post.pem
-keyout /etc/postfix/ssl/post.pem

($:~)=> /usr/sbin/openssl gendh -rand /etc/postfix/ssl/post.rand
```

```
512
>>/etc/postfix/ssl/post.pem

($:~)=> /usr/sbin/openssl x509 -subject -dates -fingerprint
-noout -in
/etc/postfix/ssl/post.pem
```

Next we need to create a symbolic link before SASL2 gets confused.

```
($:~)=> ln –s /usr/local/lib/sasl2 /usr/lib/sasl2
```

Now the next file to edit is /usr/lib/sasl2/smtpd.conf

```
($:~)=> vi /usr/lib/sasl2/smtpd.conf
```

The contents of this file look like this:

```
pwcheck_method: saslauthd
mech_list: plain login
```

Next we need to tell SASL2 what method we wish to authenticate our mail users. In this document we're going to be using system users and passwords. You might want to read the man page for other methods of authenticating users.

```
($:~)=> vi /usr/local/etc/rc.d/saslauthd.sh
```

Now change the sasl_saslauthd_flags variable from -a pam to –a getpwent. The line should like something like this:

```
sasl_saslauthd_flags=”-a getpwent”
```

Be sure to send yourself a test e-mail after starting up all the services to ensure they are working properly.

The Access file:
This file is the definitive list as that decides who to accept incoming mail from the internet. If you defined it earlier in the main.cf file it must be defined here. Basic syntax is one user per line followed by an OK, so edit /etc/postfix/access now:

```
"userid1@domain1 OK"
"userid2@domain1 OK"
"userid3@domain2 OK"
```

etc. etc. Now for a little postfix sorcery, don't specify an account if you don't want them to have the ability to receive internet mail for instance, if you only want the ability for someone to mail local accounts (accounts only contained on your box, leave them out of here) Also, if you are running lots of Virtual domains, you may want to specify system account for each domain, like:

```
"webmaster@domain1 OK"
"webmaster@domain2 OK"
"postmaster@domain1 OK"
"postmaster@domain2 OK"
```

Remember, if the specific e-mail you want to receive mail from is not listed in this file, it will bounce. Once you've entered all your info, save that file. From here you must make that file into a hashed db. Also, after any updates you make to this file in the future, you must recompile that hashed db to include your new accounts. Do this by typing the following:

```
($:~)=> postmap access
```

And to avoid any delay you should always reload postfix, but don't do that now because we haven't turned postfix on yet! It can be done like this:

```
($:~)=> postfix reload
```

CANONICAL FILES:
The next files we'll look at are, /etc/postfix/canonical and /etc/postfix/canonical-receive. Again, if you're not using canonical tables specified in your main.cf file, you don't need to worry about it here.

canonical
The canonical file as defined in this how-to will remap a user's e-mail from the default domain to the appropriate virtual domain. If this is the case you need to specify all users except those in the default domain of the box here! Also if you are mapping a local account to use another name eg. userid1 -> full.name you need to specify that here. Syntax is: userid@domain userid@virtualdomain, or, userid@domain full.name@domain this file handles mail outgoing only! edit /etc/postfix/canonical now

```
"userid@defaultdomain userid@virtualdomain"
"userid2@defaultdomain userid2@virtualdomain"
```

Or

```
"userid@defaultdomain full.name@defaultdomain"
```

canonical-receive
This file is used to clean up incoming mail so that the virtual addresses don't get remapped to the default domain. All users should have an entry here including system accounts unless they are on the default domain of the box alone, and not using virtual usernames. With that, let's edit /etc/postfix/canonical-receive now!

```
"userid1@defaultdomain userid1@defaultdomain"
"userid1@virtualdomain userid1@virtualdomain"
"userid2@virtualdomain userid2@virtualdomain"
"postmast@virtualdomain userid2@virtualdomain"
"webmaster@defaultdomain webmaster@defaultdomain"
```

```
"webmastr@virtualdomain userid2@virtualdomain"
```

Or for virtual usernames

```
"full.name@domain userid@domain"
```

Got it? Good. It's not that difficult now is it? Again once you're done with these files you'll need to make a hashed db out of them using the postmap command and you'll need to reload postfix. You remember how to do that right? Good! You're on your way.

CLIENT_ACCESS:
The next file we'll be looking at is the /etc/postfix client_access file. This file will specify a list of exceptions and specific denials of mail servers. For instance, your friend, god bless him, has a mail server but is pretty clueless when it comes to dns, he hasn't configured his dns to reverse lookup properly. Well you could bypass that here. Also, you have some evil spammer that keeps sending you mail and the rbl list isn't blocking him, you could add a specific block here. Syntax of this file is xxx.xxx.xxx.xxx function where x is an ip address and function is either OK or REJECT!

```
"xxx.xxx.xxx.xxx OK"
"xxx.xxx.xxx.xxx REJECT"
```

after making this file do you know what you need to do? That's right you must make a hashed db out of this file as well! Ok that's done let's move on!

SENDER_ACCESS:
This file /etc/postfix/sender_access is where you can specify specific e-mail addresses or domains to block. Usually bogus SPAM addresses. Syntax is fakeemail@bogusdomain.com function where the function is a reject code.

```
"fakeemail@fakedomain.com 550 No Spam Accepted"
"fakedomain.com 550 What kind of bogus address is that?"
```

etc. etc. This file further solidifies you box and your place as an anti-SPAM ninja! -For you McDonald! You'll need to make this file a hashed db as well once you're done with it!

BODY_CHECKS:
The next file we'll be looking at is /etc/postfix/body_checks. This file is either a regex file or a pcre file (if you compiled postfix with pcre support) I mainly use this file to block troublesome attachments I have no use for anyways. The following blocks certain types of attachments. Self-explanatory.

```
"/^(.*)name="(.*). (com|vbs|vbe|js|jse|exe|bat|cmd|vxd|scr|hlp|
swf|mpeg|mpg|mov|mp3|avi|pif|mpe|shs|ini)"$/ REJECT"
```

This will help out with viruses among other things, you won't have to worry about vbs scripts and that sort of stupid thing from now on. This file just needs to be in place. There is NO need to make it a hashed db. Although after making changes DO reload postfix!

HEADER_CHECKS:
The next file we'll take a look at is /etc/postfix/header_checks. This file does exactly as it says. It checks mail headers. Again either regex or pcre. I will give a couple of examples here that I use:

```
# This will block 8 non ascii characters in a row, which
shouldn't be in the
# header anyway according to the RFC... Japan and Chinese
spammers...
"/[^[:print:]]{8}/ REJECT"

# Pegasus uses "Comments: ..." not "Comment: ...". spammers got
it wrong.
"/^Comment: Authenticated sender is/ REJECT"
```

These are just a couple examples. My files are actually really large for this. You can spend a lot of time doing this! Try to avoid going crazy, as it can be fairly obsessive! Again no hashed db needed, just reload postfix when you finish altering.

REGEXP_ACCESS:
The next file to look at is regexp_access. This file pretty much does some more of the same, kicking spammers where it hurts! Here is an example I have in mine:

```
"/[%!@].*[%!@]/ 550 Sender specified routing is not supported
here."
```

There you see, not too bad huh? Ok, again reload postfix when you're done. No db needed, so let's move on.

ALIASES:
Let's edit /etc/aliases

The aliases file is very limited with the configuration we have specified here. It does need some things filled in. Standard system aliases should be placed here: aliases for root, postmaster, abuse, etc. etc. Also, if you are running majordomo, you'll specify your outgoing secrets here. If you're forwarding mail to another domain and not using a .forward file in your home, specify that here as well. Other than that, you should be good to go. After editing the aliases file, you should run the command **newaliases** to tell the system there's new content in that file.

STARTING UP POSTFIX:
Ok now, our configuration is complete. Let's start up postfix! Run the following:

```
($:~)=> postfix start
```

It should start-up error free! Be sure to send a couple e-mails to yourself through yahoo or something and watch your maillog to ensure everything goes smoothly. I would actually recommend keeping this setup for a week or 2 before continuing with the rest of this article.

Installing Courier POP and IMAP:

```
($:~)=> cd /usr/ports/mail/courier-imap ; make install clean
```

Next we need to generate our imap ssl certs

```
($:~)=> /usr/local/share/courier-imap/mkimapdcert
($:~)=> cd /usr/local/etc/courier-imap/
```

If you notice every configuration file in here is named .dist, so let's copy those so our services actually work.

```
($:~)=> for foo in *.dist; do cp $foo `basename $foo .dist`;
done
($:~)=> mv quotawarnmsg.example quotawarning
```

Pretty simple on this one.

```
($:~)=> vi /usr/local/etc/courier-imap/imapd
($:~)=> vi /usr/local/etc/courier-imap/imapd-ssl
```

Make any changes that you feel are necessary.

Now here's a brief introduction to imap and what it can do for you:

Introduction to imap and maildirmake:

Imap is a great protocol for handling mail. For the reason that all mail that a user gets is still stored on the mailserver, so in the event that windows crashes (and trust me it will) all your valuable e-mail is still nice and safe. You can also setup user quotas and public shared imap folders. A good example of the need for public shared folders is, say that you subscribe to bugtraq mailing lists and the freebsd mailing lists. And you're not the only geek in the neighborhood who enjoys reading these. Instead of having both accounts get mail from them you can ultimately setup some procmail rules to automatically send those e-mails to the shared imap folder for anybody to read and enjoy.

For each user we need to create a mailbox. We will make one for the user soupx in this demonstration.

```
($:~)=> cd /home/soupx
($:~)=> su soupx
($:~)=> maildirmake ./Maildir
```

To make things simple I changed users to soupx so that the permissions are correct. If you do this as root, the folders will be marked with the ownership of root and the local user "soupx" will not be able to write or read mail from there. So unless you want to change the permissions on the Maildir folders, it's just easier this way.

Now, say we wanted to put a quota on soupx's mail account. As the root user we would run

```
($:~)=> maildirmake -q 10000000S ./Maildir
```

On certain occasions, I've run into issues where the quota values do not change. To get around this, create a cron job to perform this task for you every 30 minutes or so on the local mailboxes.

Now let's talk about shared imap folders. First, we need to create a collection of shared folders on a separate Maildir. Since we're sending our SPAM over to /var/mail/spam, it only seems logical to make a maildir in /var/mail/shared.

```
($:~)=> maildirmake -S /var/mail/shared
```

Now we create the individual folders we want shared.

```
($:~)=> maildirmake -s write -f bugtraq /var/mail/shared
```

Here we are stating that the "bugtraq" folder is created with read/write permissions to everybody. You can set your permissions however you like. Now you can also use maildirmake to join the folders to peoples' mailboxes, but we're going to use a different approach. we're going to be defining things in a simple configuration file:

```
($:~)=> vi /usr/local/etc/courier-imap/maildirshared
```

An example is listed below:

```
# ==========================================
# Shared IMAP Folders
# ==========================================

Bugtraq /var/mail/Shared/Bugtraq
FreeBSD /var/mail/Shared/FreeBSD
OpenBSD /var/mail/Shared/OpenBSD
NetBSD /var/mail/Shared/NetBSD
```

The first area is the name we wish to display on our shares, followed by a single space or tab, and then the location of the folder.

Congratulations! You should now have POP, IMAP, and IMAP-SSL working on your new postfix install...Now onto bigger and better things.

Configuring Procmail:
This part is easy. And any dumbass could probably figure this one out, but just so I don't get 500 e-mails asking how to do this, I'll go over it anyways..

```
($:~)=> cd /usr/ports/mail/procmail ; make install clean
```

Now let's tell postfix that we now have procmail installed on our system, so let's edit the main.cf file again.

```
($:~)=> vi /etc/postfix/main.cf
```

Scroll down to the end of the file and add in:

```
mailbox_command = /usr/local/bin/procmail -m
/usr/local/etc/procmailrc
```

Save, exit, rehash and reload. Congratulations! Procmail is now installed and ready for the next part of our howto.

Configuring SPAMassasin and vipul's razor to postfix:
Thanks to strog over at screamingelectron.org for the information on this one.

```
($:~)=> cd /ports/mail/p5-Mail-SpamAssassin/ ; make install
clean
```

Now we need to create a procmailrc file

```
($:~)=> vi /usr/local/etc/procmailrc

# tell procmail we use Maildir style
DEFAULT="$HOME/Maildir/"

# specify the location for identified spam
SPAM="/var/mail/spam/new"

# various debugging stuff uncomment if needed
# VERBOSE=no
# LOGFILE=/var/log/procmailrc
# LOGABSTRACT=no

# Allow previously identified spam to be delivered since it
# must have been approved to get back here with the
# X-Spam-Deferred: YES flag set

:0 w
* ^X-Spam-Deferred: YES
$DEFAULT

# SPAM time
# first send to razor

:0 Wc
|/bin/nice usr/bin/razor-check -home=/root -logfile=
/var/log/razor-agent.log

# if previous procmail recipe successfully completed then
# message is spam. prepare for quarantine. use formail to
# rip Delivered-To out (else you'll get loop errors) and tag
```

```
# with identifying headers

:0 Waf
| formail -I "Delivered-To:" -A "X-Razor2-Warning: SPAM" -A "X-
Spam-Deferred: YES"

# drop razor identified spam into $SPAM (see above)

:0:
* ^X-Razor2-Warning: SPAM
$SPAM

# got this far, time to hand it off to spamassassin

:0fw
| /usr/bin/spamc

# Redirect definitive spam add identifying tags and rip
# postfix Delivered-To headers

:0 f
* ^X-Spam-Flag: YES
| formail -I "Delivered-To:" -A "X-Spam-Deferred: YES"

# quarantine it

:0:
* ^X-Spam-Flag: YES
$SPAM
```

A quick note on the X-Spam-Deferred: YES flag. We need to add this as mail gets identified so that if we later determine that we want to actually approve that message on to deliver to the intended recipient, we can sneak it back through procmail without getting tossed back into the SPAM trap. Ok, almost done! The next thing is to add a means to easily examine the messages in the SPAM trap. For this we rely on the mighty mutt! We want to add one simple line to Muttrc (usually /usr/local/etc/mutt/Muttrc):

```
# Approve message identified as spam and deliver it macro index
A "| /usr/sbin/sendmail"
```

Now we need to create the Maildir for our SPAM.

```
($:~)=> maildirmake /var/mail/spam ; chown root:mail
/var/mail/spam
```

Then we can just use mutt to view /var/mail/spam:

```
($:~)=> mutt -f /var/mail/spam
```

You can now browse through identified and quarantined SPAM, looking for incorrectly identified SPAM, etc. If you find a message you want to approve, from the index screen just hit Shift A. This will prompt you asking if you want to pipe to /usr/sbin/sendmail. Hit enter to do it and the message will get delivered, bypassing further SPAM detection. Don't forget to delete the message!!! You can, if you so desired, tack some additional commands on the macro added to mutt to delete the message after approval.

NOTE: If you use mbox style delivery as opposed to Maildir, simply remove the top line of the procmailrc that tells procmail to deliver Maildir style.

You may want to configure SPAMAssassin by editing /usr/local/etc/mail/spamassassin/local.cf. By default, SPAMAssassin modifies messages heavily. I recommend the following options in order to minimally annoy people who don't want to be pestered with this crap:

```
rewrite_subject 0
report_header 1
defang_mime 0
```

Congrats you don’t have SPAM. Now we take our journey into defeating those pesky viruses that even the best system administrator can get in their e-mail unless precautions are made.

Now let's do some work with Razor.

```
($:~)=> cd /usr/ports/mail/razor-agents/ ; make install clean
($:~)=> vi /usr/local/etc/razor.conf
```

Here’s a good starting configuration to use

```
debuglevel = 3
identity = identity
ignorelist = 0
listfile catalogue = servers.catalogue.lst
listfile discovery = servers.discovery.lst
listfile nomination = servers.nomination.lst
logfile = razor-agent.log
logic method = 4
min cf = ac + 10
razorzone = razor2.cloudmark.com
rediscovery wait = 172800
report headers = 1
turn off discovery = 0
use engines = 1,2,3,4
whitelist = razor-whitelist
razorhome = /usr/local/etc/razor
logfile = /var/log/razor-agent.log
```

Configuring Amavis Virus Scanning:

This part is extremely easy to setup but can drive the average user nuts trying to get it to work properly. The best software to use in my opinion to do virus scanning is McAfee's Sophis. However, this is a commercial product and we are cheap bastards here. So in the ideal of this article, we're going to use a primary scanner available in the ports tree called vscan and a secondary backup scanner called clamav.

```
($:~)=> cd /usr/ports/security/vscan ; make install clean
($:~)=> cd /usr/ports/security/clamav ; make install clean
```

So let's start off by installing amavisd-new

```
($:~)=> cd /usr/ports/security/amavisd-new
($:~)=> vi Makefile
```

Now to follow our maildirectories change

```
AMAVISQUARANTINE?= /var/virusmails
```

To

```
AMAVISQUARANTINE?= /var/mail/virusmails
```

Now install it

```
($:~)=> make install clean
```

Now let's move onto configuring it.

```
($:~)=> vi /usr/local/etc/amavisd.conf
```

The main thing to change here is the domain name of our server

```
$mydomain = 'example.com'; # Change to reflect your domain
```

Now we need to set a port and host for amavis.

```
$forward_method = 'smtp:127.0.0.1:10025'; # Where to forward
checked mail
$notify_method = $forward_method; # Where to submit
notifications
$inet_socket_port = 10024; # Where to accept SMTP requests from
```

Make any other necessary changes to this configuration file. The file is well documented so I don't believe I need to go in-depth on what needs to be done here. Now we need to tell postfix that amavis is here.

```
($:~)=> vi /etc/postfix/main.cf
```

And add in:

```
content_filter = smtp-amavis:[127.0.0.1]:10024
```

Next edit the master.cf

```
($:~)=> vi /etc/postfix/master.cf
```

And add in:

```
smtp-amavis unix - - y - 2 smtp
-o smtp_data_done_timeout=1200
-o disable_dns_lookups=yes
127.0.0.1:10025 inet n - y - - smtpd
-o content_filter=
-o local_recipient_maps=
-o smtpd_helo_restrictions=
-o smtpd_client_restrictions=
-o smtpd_sender_restrictions=
-o smtpd_recipient_restrictions=permit_mynetworks,reject
-o mynetworks=127.0.0.0/8
```

Don't forget to rehash the files and restart postfix.

Next we need our virus scanner

```
($:~)=> cd /usr/ports/security/clamav ; make install clean
```

Cyrus SASL/TLS And Postfix SSL:

Let's get SASL2 Installed now.

```
($:~)=> cd /usr/ports/security/cyrus-sasl2 ; make install clean
```

Now go ahead and edit postfix's main.cf so we can tell it to start utilizing the TLS features. Add in the following somewhere near the bottom.

```
#TLS
smtp_use_tls = yes
smtpd_use_tls = yes
smtpd_tls_auth_only = yes
smtp_tls_note_starttls_offer = yes
smtpd_tls_key_file = /etc/postfix/ssl/post.pem
smtpd_tls_cert_file = /etc/postfix/ssl/post.pem
smtpd_tls_CAfile = /etc/postfix/ssl/post.pem
smtpd_tls_loglevel = 3
smtpd_tls_received_header = yes
smtpd_tls_session_cache_timeout = 3600s
```

```
tls_random_source = dev:/dev/urandom

enable_sasl_authentication = yes

smtpd_sasl_auth_enable = yes
smtpd_sasl_security_options = noanonymous
smtpd_sasl_local_domain =
broken_sasl_auth_clients = yes
```

Also add this in the beginning of your recipient restrictions

```
($:~)=> permit_sasl_authenticated,
```

Now we create our postfix SSL Stuff

```
($:~)=> mkdir /usr/local/etc/postfix/ssl
($:~)=> chmod 700 /usr/local/etc/postfix/ssl
```

Next we create our SSL certificates for postfix

```
($:~)=> cd /usr/local/etc/postfix/ssl
($:~)=> vi pst.cnf
```

The contents of pst.cnf are:

```
RANDFILE = /etc/postfix/ssl/post.rand

[ req ]
default_bits = 1024
encrypt_key = yes
distinguished_name = req_dn
x509_extensions = cert_type
prompt = no

[ req_dn ]
C=countryName Two letters!
ST=stateOrProvinceName
L=localityName
O=organizationName
OU=OrganizationalUnitName
CN=commonName
emailAddress=emailAddress

[ cert_type ]
nsCertType = server
```

Be sure to make the correct options. Next we generate our SSL certificates.

```
($:~)=> dd if=/dev/urandom of=/etc/postfix/ssl/post.rand count=1
```

```
2>/dev/null

($:~)=> /usr/sbin/openssl req -new -x509 -days 365 -nodes
-config /etc/postfix/ssl/pst.cnf -out /etc/postfix/ssl/post.pem
-keyout /etc/postfix/ssl/post.pem

($:~)=> /usr/sbin/openssl gendh -rand /etc/postfix/ssl/post.rand
512
>>/etc/postfix/ssl/post.pem

($:~)=> /usr/sbin/openssl x509 -subject -dates -fingerprint
-noout -in
/etc/postfix/ssl/post.pem
```

Next we need to create a symbolic link before SASL2 gets confused.

```
($:~)=> ln –s /usr/local/lib/sasl2 /usr/lib/sasl2
```

Now next file to edit is /usr/lib/sasl2/smtpd.conf

```
($:~)=> vi /usr/lib/sasl2/smtpd.conf
```

The contents of this file look like below:

```
pwcheck_method: saslauthd
mech_list: plain login
```

Next we need to tell SASL2 what method we wish to authenticate our mail users. In this document we're going to be using system users and passwords. You might want to read the man page for other methods of authenticating users.

```
($:~)=> vi /usr/local/etc/rc.d/saslauthd.sh
```

Now change the sasl_saslauthd_flags variable from -a pam to –a getpwent. The line should like something like this:

```
sasl_saslauthd_flags=”-a getpwent”
```

Be sure to send yourself a test e-mail after starting up all the services to ensure they are working properly.

So where do you go from here? My best suggestions would be to study up on different ways that you can authenticate mail users. Ultimately, a good way is to use OpenLDAP to keep a database of users and store mail through there. That’s beyond the scope of this article but a definite nice way to get things done. By utilizing the OpenLDAP services you cannot only have postfix authenticate though they’re but courier-imap and a vast majority of your services. Along with the ability to use a shared contacts system or whatever you can think of.

Firewall Configurations:

It would take me all month to give you a good overview of how PF or IPFW works. I pretty much don't have the time for that these days. So I'm just going to cover final stages of getting PF to work and include a decent sample rule set.

First lockdown the kernel. I'm assuming there should not be any further changed made to it.

```
($:~)=> vi /etc/rc.conf
```

Here's the changes we're going to be making:

```
kern_securelevel_enable="YES"
kern_securelevel="1"
```

You might want to read up a little bit in the handbook to see which securelevel is right for you. **NOTE:** When operating in kernel securelevel, the Xwindows system **WILL NOT WORK!** There I hope I save you some future frustrations.

Now let's enable the PF firewall with use of IPNAT

```
ipfilter_enable="YES"
ipnat_enable="YES"
ipfilter_rules="/etc/ipf.rules"
```

If you're using IPFW, you would use something like this:

```
firewall_enable="YES"
firewall_script="/etc/ipfw.rules"
firewall_quiet="NO"
firewall_logging="YES"
```

Now we will activate the Ip monitoring utility.

```
ipmon_enable="YES"
ipmon_flags="-Dvn /var/log/firewall.log"
```

So, any denied packets that come in, as long as in your rules you state to log, will be sent to /var/log/firewall.log. This is a good file to keep an eye on. You will be amazed at how many evil things try to make their way into your box.

Sample Firewall Rules:

I'm only going to cover PF rule sets here. I also recommend you spend the time to learn IPFW and ultimately OpenBSD's PF, which is just a crazy insane firewall that just dominates. For the

demonstration of these rules, we will be using sis0 as our network card.

```
($:~)=> vi /etc/ipf.rules

# ==========================================
# Here is a basic list of things we need to be able to access
# ==========================================

pass out quick on sis0 proto udp from any to any port = 67 keep
state
pass out quick on sis0 proto udp from any to any port = 68 keep
state

#Allow web browsing
pass out quick on sis0 proto tcp from any to any port = 80 flags
S keep frags keep state

#Allow email usage / pop3 / imap / smtp
pass out quick on sis0 proto tcp from any to any port = 110
flags S keep frags keep state
pass out quick on sis0 proto tcp from any to any port = 143
flags S keep frags keep state
pass out quick on sis0 proto tcp from any to any port = 25 flags
S keep frags keep state

#Allow outgoing ssh
pass out quick on sis0 proto tcp from any to any port = 22 flags
S keep frags keep state

#Allow DNS lookups
pass out quick on sis0 proto udp from any to any port = 53 keep
state keep frags

#Allow CVSUP Access
pass out quick on sis0 proto tcp from any to any port = 5999
flags S keep state

#Allow access to Time Server
pass out quick on sis0 proto tcp from any to any port = 37 keep
state
pass out quick on sis0 proto tcp from any to any port = 123 keep
state

#Allow whois lookups
pass out quick on sis0 proto tcp from any to any port = 43 keep
state

#Razor and spamassasin needs
pass out quick on sis0 proto tcp from any to any port = 2703
flags S keep state
```

```
pass out quick on sis0 proto tcp from any to 216.52.3.16 port <
3500 flags S keep state
pass out quick on sis0 proto tcp from any to 216.52.13.94 port <
3500 flags S keep state
pass out quick on sis0 proto tcp from any to 216.52.3.10 port <
3500 flags S keep state
pass out quick on sis0 proto tcp from any to 216.52.3.5 port <
3500 flags S keep state
pass out quick on sis0 proto tcp from any to 216.52.13.92 port <
3500 flags S keep state
pass out quick on sis0 proto tcp from any to 216.52.3.6 port <
3500 flags S keep state

#IRC
pass out quick on sis0 proto tcp from any to any port = 6667
keep state
pass out quick on sis0 proto tcp from any to any port = 6668
keep state
pass out quick on sis0 proto tcp from any to any port = 6669
keep state

# Other Random Things.

#IGMP
block out quick on sis0 proto igmp all

#ICMP
block out quick on sis0 proto icmp from any to any keep state

# EGRESS filtering
# Filter outbound packets directed to reserved networks
# Uncomment the first line to allow Multicast traffic.

block out quick on sis0 from !192.168.1.0/24 to any
block out quick on sis0 from any to 0.0.0.0/7
block out quick on sis0 from any to 2.0.0.0/8
block out quick on sis0 from any to 5.0.0.0/8
block out quick on sis0 from any to 10.0.0.0/8
block out quick on sis0 from any to 23.0.0.0/8
block out quick on sis0 from any to 27.0.0.0/8
block out quick on sis0 from any to 31.0.0.0/8
block out quick on sis0 from any to 69.0.0.0/8
block out quick on sis0 from any to 70.0.0.0/7
block out quick on sis0 from any to 72.0.0.0/5
block out quick on sis0 from any to 82.0.0.0/7
block out quick on sis0 from any to 84.0.0.0/6
block out quick on sis0 from any to 88.0.0.0/5
block out quick on sis0 from any to 96.0.0.0/3
block out quick on sis0 from any to 127.0.0.0/8
```

```
block out quick on sis0 from any to 128.0.0.0/16
block out quick on sis0 from any to 128.66.0.0/16
block out quick on sis0 from any to 169.254.0.0/16
block out quick on sis0 from any to 172.16.0.0/12
block out quick on sis0 from any to 191.255.0.0/16
block out quick on sis0 from any to 192.0.0.0/19
block out quick on sis0 from any to 192.0.48.0/20
block out quick on sis0 from any to 192.0.64.0/18
block out quick on sis0 from any to 192.0.128.0/17
block out quick on sis0 from any to 192.168.0.0/16
block out quick on sis0 from any to 197.0.0.0/8
block out quick on sis0 from any to 201.0.0.0/8
block out quick on sis0 from any to 204.152.64.0/23
block out quick on sis0 from any to 206.112.0.0/16
block out quick on sis0 from any to 224.0.0.0/3

#All The Rest
block out on sis0 all

# ========================================
# What ports do i want allowed to be open
# ========================================

#Allow bootp traffic in from your ISP's DHCP server only.
pass in quick on sis0 proto udp from any to any port = 68 keep
state
pass in quick on sis0 proto udp from any to any port = 67 keep
state

#Incomming SSH Access

pass in quick on sis0 proto tcp from STATION1 to SERVER port =
22 flags S keep frags keep state
pass in quick on sis0 proto tcp from 166.222.222.0/24 to SERVER
port = 22 flags S keep frags keep state
pass in quick on sis0 proto tcp from STATION2 to SERVER port =
22 flags S keep frags keep state
```

OK, let me take a quick break to explain something here. We don't want any computer on the internet to be able to try and gain SSH access to our box. So we are going to filter who can get in and who can't. So inside your /etc/hosts file, you declare a hostname to match an IP address, so we're saying that on port 22 (SSH) we will only allow hostname STATION1 to connect to SERVER (our box).

```
#SMTP/POP/IMAP
pass in quick on sis0 proto tcp from any to any port = 25 keep
state
pass in quick on sis0 proto tcp from any to any port = 110 flags
S keep frags keep state
```

```
pass in quick on sis0 proto tcp from any to any port = 143 keep
state

#AMAVISD
block in quick on sis0 proto tcp from any to any port = 10024
keep state

#ICMP
block in quick on sis0 proto icmp from any to any keep state

#IGMP
block in quick on sis0 proto igmp all

#OTHER ODDITIES
block in quick on sis0 all with ipopts
block in quick on sis0 all with frag
block in quick on sis0 all with short
block return-rst in quick on sis0 proto tcp all flags FUP
block return-rst in quick on sis0 proto tcp from any to any
block return-icmp-as-dest(port-unr) in quick on sis0 proto udp
from any to any

## Now we are blocking packets that are too short to
## contain a complete header, or with source routing
## options (most-likely setted to bypass our firewall).

block in log quick on sis0 all with opt lsrr
block in log quick on sis0 all with opt ssrr

# Prevent spoof of bogus/non-routable addresses
# May seem like overkill since we already block everything,
# but we really want to make sure these networks never reaches
us.

block in quick on sis0 from 0.0.0.0/7 to any
block in quick on sis0 from 2.0.0.0/8 to any
block in quick on sis0 from 5.0.0.0/8 to any
block in quick on sis0 from 10.0.0.0/8 to any
block in quick on sis0 from 23.0.0.0/8 to any
block in quick on sis0 from 27.0.0.0/8 to any
block in quick on sis0 from 31.0.0.0/8 to any
block in quick on sis0 from 69.0.0.0/8 to any
block in quick on sis0 from 70.0.0.0/7 to any
block in quick on sis0 from 72.0.0.0/5 to any
block in quick on sis0 from 82.0.0.0/7 to any
block in quick on sis0 from 84.0.0.0/6 to any
block in quick on sis0 from 88.0.0.0/5 to any
block in quick on sis0 from 96.0.0.0/3 to any
block in quick on sis0 from 127.0.0.0/8 to any
```

```
block in quick on sis0 from 128.0.0.0/16 to any
block in quick on sis0 from 128.66.0.0/16 to any
block in quick on sis0 from 169.254.0.0/16 to any
block in quick on sis0 from 172.16.0.0/12 to any
block in quick on sis0 from 191.255.0.0/16 to any
block in quick on sis0 from 192.0.0.0/19 to any
block in quick on sis0 from 192.0.48.0/20 to any
block in quick on sis0 from 192.0.64.0/18 to any
block in quick on sis0 from 192.0.128.0/17 to any
block in quick on sis0 from 192.168.0.0/16 to any
block in quick on sis0 from 197.0.0.0/8 to any
block in quick on sis0 from 201.0.0.0/8 to any
block in quick on sis0 from 204.152.64.0/23 to any
block in quick on sis0 from 224.0.0.0/3 to any

#ALL THE REST
block in log quick on sis0 all

# Loopback Interface
pass in quick on lo0 all
pass out quick on lo0 all
```

Save and exit. With a little bit of modifying you should have a decent set of rules there.

Closing Statements

Originally, I planned on making this document a lot more in-depth specifically towards IDS, setting up various other services. Then I got burned out and I just don't feel like it anymore. This is my final article that I had written for bsdhound, and I hope you enjoy it. It's been a great few years operating that site, and as I mentioned above, this is my gift back to you all for all the support and help you've given me throughout the years. I hope you enjoyed reading this and possibly learned a thing or two.

References Used: Daemonnews.org, Onlamp.com, Screamingelectron.org, BSDVault.net

Editors Note: This article was written by a long time-forum member of LWD. It was originally written and posted on his site, BSDHound, which has since been removed from service. Since that time, the article has been thoroughly updated, and reprinted here on LWD. Please let us know what you think of this article, by posting your comments
http://www.littleblackdog.com/viewtopic.php?p=194945

My Notes

- *Original edited by Don Munyak, 07/27/2006*
- *Need to look into JAILS.*
- *How Mail:Toaster fits into this as well.*
- *Updates for current FreeBSD release.*

- *http://solarflux.org/pf*
- *http://www.watson.org/fbsd-hardening/lockdown.pol*
-