# Exercises: FreeBSD Security: pre SANOG VI Workshop

January 11, 2005

**Exercises**

**1.) Enable ssh and start the service** [Top]

For this exercise you need to be root user.

Remember that we discussed that you must first specify in /etc/rc.conf that a service is enabled before you can even start it by using, for instance, a startup script located in /etc/rc.d/. The ssh service is an excellent example of this. So, to first enable the ssh daemon (server) you must edit the file /etc/rc.conf:

```
$ vi /etc/rc.conf
```

Where you place the line to enable ssh is not that important, but why don't you scroll down to the line that reads:

```
moused_enable="YES"
```

and press the "o" key to add a new line just under this line. Then add a line to the file /etc/rc.conf that reads:

```
sshd_enable="YES"
```

Now save the file by pressing ESCape and then ":wq".

Notice, if you type:

```
$ grep ssh /etc/defaults/rc.conf
```

you'll see:

```
sshd_enable="NO"                # Enable sshd
sshd_program="/usr/sbin/sshd"   # path to sshd, if you want a different one.
sshd_flags=""                   # Additional flags for sshd.
```

So, by placing 'sshd_enable="YES"' in /etc/rc.conf you have overridden the default setting for this service.

Now we still need to start the ssh server. To do this type:

```
$ /etc/rc.d/sshd start
```

And, this should start ssh on your machine. Remember, ssh will now start each time you reboot your machine as well.

To verify ssh is really running try these commands:

```
/$ etc/rc.d/sshd status

$ ps -auxw | grep ssh
```

Did you notice in the second command where the actual ssh program file is located?

Finally, try connecting to your own machine, but connect using the *username* you created for yourself on day 1.

```
$ ssh username@localhost
```

"localhost" is a convention for 127.0.0.1 or your machine.

Here is what your session should have looked like, more or less:

```
localhost# ssh hervey@localhost
The authenticity of host 'localhost (::1)' can't be established.
DSA key fingerprint is d8:d1:eb:2a:89:a8:e0:2c:c7:e4:66:0e:2e:21:12:ed.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (DSA) to the list of known hosts.
Password:
```

We'll go over in detail what the "fingerprint" prompt means in our next section dealing with ssh and security, but for now just say yes to the initial prompt that you receive.

After you are connected to your machine as your *username* be sure to logout to continue with the exercises:

```
exit
```

**2.) Allow root connections with ssh** [Top]

For this exercise you need to be root.

This is generally a bad idea - to allow root access for ssh. It is much better to connect to your server as a user that is allowed to become root using the "su" command, or that has "sudo" privileges.

If/when a security hole is found in ssh you don't want someone to gain access to your machine as root simply because you allow root to connect directly with ssh.

For now, however, for purposes of this class we'll ask that you allow root access to your machine so that the instructors can gain access quickly if necessary (you didn't change the root password, right? :-)). The other way to do this is to create an account like "inst" (a convention we've used in the past), place it in the wheel group, and use the same password for this account to avoid allowing root access.

To tell ssh to allow root to connect you will need to edit the ssh server config file and change one line in the file. Otherwise when you try to login as root you'll just get the password prompt multiple times.

The file we are going to edit is /etc/ssh/sshd_config. So, to do this:

```
$ vi /etc/ssh/sshd_config
```

Now, let's use a vi search trick to find the line we want to change. In vi press the "/" (forward slash) key and type:

```
PermitRootLogin
```

This should take you directly to the line which reads:

```
#PermitRootLogin no
```

Press the "i" key in vi (for input mode) and change this line to read:

```
PermitRootLogin yes
```

And, now save and exit the file by pressing the ESCape key and then ":wq".

Note that you had to remove the comment character from the left-hand column. The configuration directives are commented out, but they indicate the default value. Thus, to override a default value you must uncomment the directive and change the value.

Now for this change to take affect you need to restart the ssh server. There are several ways to do this, but let's just use the ssh script:

```
$ /etc/rc.d/sshd restart
```

and you'll see the sshd daemon (service) stopping and starting. Now to confirm that your change worked try:

```
$ ssh root@localhost
```

and you should be allowed to login. Note that you don't get the signature prompt this time. We'll explain this later.

Once you login as root be sure to logout to continue with the exercises:

```
$ exit
```

**3.) Turn on password checking** [Top]

Cracklib is a library and a set of routines for checking a user's password when they try to create or change their password. It will force users on your system to use better passwords and won't allow them to do things like change a password to the same password, etc...

As setup now, you could change your pasword to be a single character, your name, a common word, etc. and FreeBSD would allow this. This is a security hole.

By default FreeBSD does not do password checking when you first create a password for a user account, or when you change your user's password. We are going to turn on the PAM (Pluggable Authentication Module) Password Quality Control module. For more in-depth reading on these two items see:

```
$ man pam

$ man pam_passwdqc
```

If you use "su" to become your *username* by doing:

```
$ su - username
```

then go ahead and try changing your password to something really insecure:

```
pcn# passwd

Changing local password for username Old Password: New Password:
one Retype New Password: one
```

and note that it lets you do this! Now you have an insecure password.

Logout of your user session by typing "exit" and then we'll go and edit the file /etc/pam.d/passwd:

```
$ vi /etc/pam.d/passwd
```

And change the line that reads:

```
#password       requisite       pam_passwdqc.so         enforce=users
```

to read:

```
password       requisite       pam_passwdqc.so         enforce=users
```

Now save the file by pressing the ESCape key and typing ":wq".

Go back to your *username* session by typing:

```
$ su - username
```

And, now run through the password changing sequence again. This time you are going to see lots of verbose output. Change your password to something secure enough to be accepted by this new password change procedure:

```
pcn# passwd

Changing local password for hervey Old Password: You can now
choose the new password or passphrase. A valid password should be
a mix of upper and lower case letters, digits and other
characters. You can use an 8 character long password with
characters from at least 3 of these 4 classes, or a 7 character
long password containing characters from all the classes.
Characters that form a common pattern are discarded by the check.
A passphrase should be of at least 3 words, 12 to 40 characters
long and contain enough different characters. Alternatively, if
noone else can see your terminal now, you can pick this as your
password: "heyday,book!Virgo". Enter new password: Re-type new
password:
```

You'll find this routine is actually quite strict about what you can use and this may take you some time to be able to come up with a valid password. If you think it's too hard exit from your user shell and go back to root. Then read the man pages for pam_passwdqc again and note the various options you can set to make this easier.

**Note:** If you are typing a complex password and you make a mistake don't press the backspace or delete key, they probably won't work. Instead press CTRL-u to clear the password line (you won't see anything to indicate this), and then start typing the password over.

You can force set a password for any user to anything you want as root by simply typing:

```
$ passwd username
```

An alternative password checking library that is very popular is cracklib. You can find cracklib in /usr/ports/security/cracklib/. You can install it if you wish and then configure PAM to use cracklib instead.

## 4.) Scan your neighbor's machine with nmap [Top]

For this exercise you need to be root user.

First we need to install the "network exploration tool and security scanner" (nmap) from the ports collection. You can find nmap in /usr/ports/security/nmap/. So, to install nmap do:

```
$ cd /usr/ports/security/nmap/

$ make

$ make install
```

Remember from on day 1 that we set the default machine for finding source when you make ports to be our noc. We set this in /etc/make.conf. We've placed the nmap source on noc, so this procedure should be quite quick.

Now, to get a feel for what nmap can do:

```
$ man nmap
```

nmap is an incredibly powerful tool, and it's a tool that can get you in to a lot of trouble if you use it without first asking for permission. That is, if you plan on scanning a series of machines with nmap, then ask their owner's or system administrators first. Generally speaking nmap will use a brute force scan of all ports trying to figure out what is running on a machine, what OS it is running, etc. This can set of all sorts of network monitoring warning bells when you do this, so be warned.

In addition, nmap can be tricked. People can, for instance, set the version being reported by a service to something else - for instance with ssh.

With that said, go ahead and scan the machine to your left with the following command:

```
$ nmap -sV -O 202.144.139.N
```

where N is your neighbor's IP address. If you don't have a neighbor to your left then you can the machine by the entrance to the lab, or 202.14.139.15.

If you read "man nmap" you'll see that the two options given do the following:

```
-sV           Version detection: Afer TCP  and/or  UDP  ports  are  discovered
              using  one of the other scan methods, version detection communi-
              cates with those ports to try and determine more about  what  is
              actually  running.  A file called nmap-service-probes is used to
              determine the best probes for detecting various services and the
              match  strings  to  expect.  Nmap tries to determine the service
              protocol (e.g. ftp, ssh, telnet,  http),  the  application  name
              (e.g. ISC Bind, Apache httpd, Solaris telnetd), the version num-
              ber, and sometimes  miscellaneous  details  like  whether  an  X
              server  is open to connections or the SSH protocol version).  If
              Nmap was compiled with OpenSSL support, it will connect  to  SSL
              servers  to  deduce the service listening behind the encryption.
              When RPC services are discovered, the Nmap RPC grinder  is  used
              to  determine  the  RPC  program  and version numbers.  Some UDP
              ports are left in the "open|filtered" state after a UDP scan  is
              unable  to determine whether the port is open or filtered.  Ver-
              sion detection will try to elicit a response  from  these  ports
              (just  as it does with open ports), and change the state to open
              if it succeeds. Note that the Nmap -A option also  enables  this
              feature.   For  a much more detailed description of Nmap service
              detection, read our paper  at  http://www.insecure.org/nmap/ver-
              sionscan.html .  There is a related --version_trace option which
              causes Nmap to print out extensive  debugging  info  about  what
              version  scanning  is  doing (this is a subset of what you would
              get with --packet_trace).

-O            This option activates remote host identification via TCP/IP fin-
              gerprinting.   In  other words, it uses a bunch of techniques to
              detect subtleties in the  underlying  operating  system  network
              stack  of the computers you are scanning.  It uses this informa-
              tion to create a "fingerprint" which it compares with its  data-
              base of known OS fingerprints (the nmap-os-fingerprints file) to
              decide what type of system you are scanning.

              If Nmap is unable to guess the OS of a machine,  and  conditions
              are  good (e.g. at least one open port), Nmap will provide a URL
              you can use to submit the fingerprint if you know (for sure) the
              OS  running on the machine.  By doing this you contribute to the
              pool of operating systems known to nmap and thus it will be more
              accurate  for everyone.  Note that if you leave an IP address on
              the form, the machine may be scanned when we add the fingerprint
              (to validate that it works).
```

Now what did you get back? It probably took a few seconds as nmap is scanning all ports (1 to 65535).

Here's a sample from scanning a machine running FreeBSD 5.3 and ssh:

```
Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2005-01-10 01:22 GMT
Insufficient responses for TCP sequencing (1), OS detection may be less accurate
Interesting ports on localhost (127.0.0.1):
(The 1659 ports scanned but not shown below are in state: closed)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 3.8.1p1 (protocol 2.0)
Device type: general purpose
Running: FreeBSD 5.X
OS details: FreeBSD 5.2-CURRENT (Jan 2004) on x86

Nmap run completed -- 1 IP address (1 host up) scanned in 17.557 seconds
```

So, why is this important? Just from this scan someone looking to break in to your system would now know two things:

1. You are running ssh version 3.8.1p1
2. You are running FreeBSD version 5.x, at least 5.2

Remember, it's possible that an incorrect version is being reported, or in this case that the nmap OS database is outdated so it did not identify the exact FreeBSD version correctly. But, still, chances are this information is correct. Now a cracker can search for ssh version 3.8.1p1 vulnerabilities and attempt to use them against this system. The same goes for FreeBSD 5.3 in general, but with so few services running it would be harder to attack.

This is why keeping running services up-to-date is *very critical*. If you run old software with security holes crackers are likely to notice and they are very likely going to try and exploit those holes.

**5.) Figure out what is running on your machine** [Top]

For this exercise you need to be root user.

There are quite a few ways to approach this, but very quickly here are some useful commands:

```
$ ps
$ netstat
$ lsof
$ sockstat
$ grep YES /etc/defaults/rc.conf
$ grep YES /etc/rc.conf
```

You should, as usual, read the man pages for "ps", "netstat", "lsof", and "sockstat". Now, to use these four commands more effectively here are some good options:

```
$ lsof -i
$ netstat -an -f inet
$ ps -auxw | more
$ sockstat -4
```

You should try all of these commands, you should read the man pages for them, and then you should understand what is being shown. For example, the commands:

```
$ lsof -i
```

```
      $ sockstat -4
```

might produce output that looks something like this:

```
localhost# lsof -i
\COMMAND   PID USER   FD   TYPE     DEVICE SIZE/OFF NODE NAME
syslogd   272 root    4u  IPv6 0xc19f3924      0t0  UDP *:syslog
syslogd   272 root    5u  IPv4 0xc19f3870      0t0  UDP *:syslog
sshd      383 root    3u  IPv6 0xc1a21000      0t0  TCP *:ssh (LISTEN)
sshd      383 root    4u  IPv4 0xc1a20e00      0t0  TCP *:ssh (LISTEN)
soffice.b 722 root    3u  IPv4 0xc1a20380      0t0  TCP *:* (CLOSED)
soffice.b 722 root    3u  IPv4 0xc1a20380      0t0  TCP *:* (CLOSED)
soffice.b 722 root    3u  IPv4 0xc1a20380      0t0  TCP *:* (CLOSED)
soffice.b 722 root    3u  IPv4 0xc1a20380      0t0  TCP *:* (CLOSED)
soffice.b 722 root    3u  IPv4 0xc1a20380      0t0  TCP *:* (CLOSED)
soffice.b 722 root    3u  IPv4 0xc1a20380      0t0  TCP *:* (CLOSED)
localhost# sockstat -4
USER     COMMAND    PID   FD PROTO  LOCAL ADDRESS         FOREIGN ADDRESS
root     soffice.bi 722   3  tcp4   *:*                   *:*
root     sshd       383   4  tcp4   *:22                  *:*
root     syslogd    272   5  udp4   *:514                 *:*
```

Basically what we are seeing is that the syslog daemon is using UDP and port 514. ssh is using TCP and listening for traffic on port 22. And, a copy of something called soffice.bin (Open Office in this case) is running using TCP port 722, but not listening for connections.

Now let's see what's starting at system boot:

```
      $ grep YES /etc/defaults/rc.conf; grep YES /etc/rc.conf
```

Yes, we've thrown one more command line convention at you. You can string commands together on the command line by using the ";" character to separate them.

Go through the output from this to get a feel for what is starting on your system.

Finally, in a perfect world you should be able to type:

```
      $ ps -auxw
```

and know what each and every item in the process list means and does. If you can get to this point, then you can quickly spot unusual items using the ps command. So, give "ps -auxw | more" a try and then use available resources like "man" to see if you can figure out what all the items in the list mean.

**6.) Turn off any unnecessary services** [Top]

This exercise is very simple and short. You just finished looking at your system in much more detail. Did you see anything that should not be running, starting, etc.? This is a *very* subjective question. If you did find something, then the likely place to turn the item off will be in the file:

/etc/rc.conf

You usually do this by adding an entry that reads:

```
      service_enable="NO"
```

and by immediately shutting it down with:

```
      $ /etc/rc.d/service stop
```

As you are using a fresh install of FreeBSD and it is a fairly minimal install there is probably nothing that needs to be turned off right now, but if you think there is call an instructor over to discuss the item.

**7.) Understand your logs** [Top]

On a daily basis your server will receive two emails that are generated from a script that is run via the crontab facility. This script reads your log files, checks your system state, etc. and then generates the summary reports for you. The reports can be very useful, particularly if you do not have additional logging facilities in place to warn you immediately of potential attacks.

First, have a look at the file:

```
$ less /etc/syslog.conf
```

and read the man page:

```
$ man syslog.conf
```

to understand how logging is taking place on your machine. Now go to the directory /var/log/

```
$ cd /var/log
```

and look at all the informational log files available to you. These files can be invaluable when troubleshooting problems and checking for potential security issues.

The log you are probably going to look at the most is "messages". As a matter of fact, some system administrators simply open a terminal window on their desktop machine that has an ssh connection to a server, login as root (i.e. "su to root"), and then type the command:

```
$ tail -f /var/log/messages
```

This leaves a permanently updating /var/log/messages logfile window open on their desktop. This is in no way a foolproof method for knowing what is happening on a machine, but it's great for detecting patterns, catching DoS attacks, or maybe even noting someone trying to guess passwords - that is as long as you are physically in front of the screen and paying attention...

Feel free to look at some/all of the log files. Some of the files are not text files, so you won't want to cat, more orless them to the screen. And, some are empty. A quick way to check what files are text, binary, data, or empty is to do the following (in the /var/log directory):

```
$ file *
```

The file command tells you what type of file a file is. This can be invaluable, particularly if the name does not give you a good indication. The output from this command looks something like this:

```
localhost# file *
Xorg.0.log:       ASCII English text
Xorg.0.log.old:   ASCII English text
Xorg.8.log:       ASCII English text
Xorg.8.log.old:   ASCII English text
auth.log:         ASCII text
cron:             ASCII text
debug.log:        empty
dmesg.today:      ASCII English text
dmesg.yesterday:  ASCII English text
kdm.log:          ASCII English text
lastlog:          data
lpd-errs:         empty
maillog:          ASCII text
```

```
maillog.0.bz2:    bzip2 compressed data, block size = 900k
maillog.1.bz2:    bzip2 compressed data, block size = 900k
messages:         ASCII English text
messages.0.bz2:   bzip2 compressed data, block size = 900k
messages.1.bz2:   bzip2 compressed data, block size = 900k
mount.today:      ASCII text
ppp.log:          empty
security:         empty
sendmail.st:      data
sendmail.st.0:    data
sendmail.st.1:    empty
sendmail.st.2:    empty
setuid.today:     ASCII text
setuid.yesterday: ASCII text
slip.log:         empty
userlog:          ASCII text
wtmp:             data
xferlog:          empty
```

So, files that are data, empty, or bzip2 encoded you don't want to look at using cat, more or less. If you do look at a binary file by mistake and your terminal window becomes confused you can type the command:

```
      $ reset
```

in the terminal to set your terminal back to it's original state.

If you want to generate the daily summary of your server right now you can issue the following command:

```
      $ periodic daily
```

It might take a minute,or so, to complete.

Now type:

```
      $ mail
```

Later in the week you will set an alias for root to an administrative user so that mails such as this don't go to the root account.

In mail you'll see something like this:

```
Mail version 8.1 6/6/93.  Type ? for help.
"/var/mail/root": 2 messages 2 new
>N  1 root@localhost.local  Sun Jan  9 11:02  37/1039  "localhost.localdomain"
 N  2 root@localhost.local  Sun Jan  9 11:02  72/2440  "localhost.localdomain"
```

Press "1" to see the first message. Press spacebar to scroll. Press" 2" to see the next message. Type "quit" to exit from mail.

Interesting information, no? You can look at your crontab file to see how and at what time the daily cron items are done. The crontab file is a configuration file for cron which is a service that executes a set of scripts at times and on days, weeks, months as specified in the file /etc/crontab. For more information see:

```
      $ man cron
      $ man crontab
```

If you want to see how the periodic script works you can look at the file /etc/defaults/periodic.conf. So, to view the crontab and periodic.conf type:

```
      $ less /etc/crontab
      $ less /etc/defaults/periodic.conf
```

Then, to see the various scripts that are executed look in the directory /etc/periodic:

```
      $ ls /etc/periodic
```

and you'll see the following:

```
localhost# ls /etc/periodic/
daily           monthly         security        weekly
```

Finally, look in one of the directories, like daily:

```
      $ ls /etc/periodic/daily
```

and, you'll see something like:

```
localhost# ls /etc/periodic/daily/
100.clean-disks         210.backup-aliases      430.status-rwho
110.clean-tmps          300.calendar            440.status-mailq
120.clean-preserve      310.accounting          450.status-security
130.clean-msgs          330.news                460.status-mail-rejects
140.clean-rwho          400.status-disks        470.status-named
150.clean-hoststat      405.status-ata-raid     500.queuerun
200.backup-passwd       420.status-network      999.local
```

These are all the scripts that can be run (if enabled in /etc/defaults/periodic.conf) each time the crontab calls the periodic script to run the daily scripts. Scripts are executed in alphabetical order, or in this case, by their indicated number at the start of the script name. You can start to look around to get a feel for what executes at what time and how to generate the mail reports you get, and much more. So, if you do:

```
      $ grep log /etc/periodic/daily/*
```

You'll see a bunch of information about what scripts do items related to "log"ging.

This is a fairly advanced and complex topic, but it's important to begin to understand what is going on in the background on your machine that keeps track of information, informs of updates, security issues, filesystem status, network traffic, etc.

If you want to install an alternate logging facility that some people seem to like have a look at /usr/ports/security/swatch. here is it's description from the /usr/ports/security/swatch/pkg-descr file:

```
localhost# cat pkg-descr
SWATCH - The Simple WATCHer and filter

Swatch is designed to  monitor  system  activity.   Swatch
requires a configuration file which contains pattern(s) to
look for and action(s) to do when each pattern is found.

WWW: http://swatch.sourceforge.net/
```

**8.) Use rsync to backup data** [Top]

An important aspect of security is backing up your data. If you are compromised and the security break-in causes you to lose data, then without a backup you could be in serious trouble.

Sometimes you simply need to have an effective way to copy (securely) a group of files from one machine to another on a regular basis, but without having to copy the entire set of files each time. This is what rsync can do. The first time you use rsync to copy a group of files all files will be copied. The next time you use rsync only files that have changed will be copied.

For this exercise you'll need to be root. You'll, also, need root access to your neighbor's machine. If you are on the first machine 202.144.139.1, please use the last machine at 202.144.139.15 for this exercise. So, if your neighbor has not enabled root login for ssh (exercise 2), then you may need to do some negotiation with them :-).

So, login on your neighbor's machine first:

```
$ ssh root@202.144.139.N
```

where "N" is your neighbor's IP address. On their machine create a temporary directory for this exercise:

```
$ mkdir /tmp/data
```

Now logout of their machine by typing "exit".

Now we are going to copy the entire contents of the /usr/ports/mail directory to your neighbor's machine in the /tmp/data directory using rsync. To do this type:

```
$ rsync -av /usr/ports/mail/ root@202.144.139.N:/tmp/data/
```

A few things to note:

- Technically you can skip the "root@" piece as you are root and rsync (with ssh) would assume this. But, it's often better to be explicit.
- Slashes count! Don't leave off the trailing slash on "mail/" or on "data/".
- For purposes of this exercise we are using the "v" option for verbose output. If you ran this as a cron job you would not use the "v" option.
- If you were to use this as a cron job at some point you would need to setup ssh keys with no passphrase for this to work without requiring a password. After these exercises we'll discuss this in our ssh security presentation and what this implies.
- The "a" option stands for archive and it's the equivalent of adding the options "rlptgoD" to the rsync command.
- *Definitely* read the rsync man page to understand how archive works.

Once the files have been copied make sure you are in the /usr/ports/mail directory:

```
$ cd /usr/ports/mail
```

And, let's create a new file in that directory:

```
$ echo "a new file" > test
```

The ">" item directs output to the file *test*. If you wanted to add another item you would use ">>" to concatenate data to the file instead.

Now, if we run the rsync command again you should see that it quickly realizes that there is only one file that is new or changed and it only copies the file *test* over. Run the command:

```
$ rsync -av /usr/ports/mail/ root@202.144.139.N:/tmp/data/
```

And see how quickly this goes. Now remove the file *test* and run the command again:

```
$ rm /usr/ports/mail/test

rsync -av /usr/ports/mail/ root@202.144.139.N:/tmp/data/
```

You'll still be prompted for the password on your neighbor's machine, but nothign will be copied.

One item to consider is that you deleted *test* on your machine, but it was not removed on your neighbor's machine. If you want data that is remove to be deleted then you need to add an option to rsync. If you don't want data to be removed ever, then just leave the command as it is.

To delete the file you would do:

```
$ rsync -av --delete /usr/ports/mail/
root@202.144.139.N:/tmp/data/
```

The rsync command is very powerful. You can setup all sorts of options for copyin over entire directory structures, but exclude certain files or directories. In addition you can tell rysnc to compress files as they are copied, to run as a daemon, etc., etc. Again, read the rsync man page for more information:

```
$ man rsync
```

Many simple backup plans include doing things like the following:

- Execute a nightly rsync job that copies all user data (/usr/home) from one machine to another (like a designated backup server that only runs ssh, is on a private network, and has a RAID array for disk).
- Execute a nightly rsync job to backup system files.
- Execute a weekly rsync job to backup user data to a separate area on the backup server.
- Execute a monthly rsync job..., and so forth.

Hervey Allen
January 2005