OpenSSH Lab

OpenSSH is a free, open source implementation of the SSH (*S*ecure *SH*ell) protocols. It replaces telnet, ftp, rlogin, rsh, and rcp with secure, encrypted network connectivity tools. OpenSSH supports versions 1.3, 1.5, and 2 of the SSH protocol. Since OpenSSH version 2.9, the default protocol is version 2, which uses RSA keys as the default.

OpenSSH server configuration:

1. Install the openssh-server and openssh rpm package included in the Red Hat Linux 7.3. Please note that the openssh-server package depends on the openssh package. OpenSSH packages also require the OpenSSL package (openssl) which installs several important cryptographic libraries that help OpenSSH provide encrypted communications

2. OpenSSH daemon, sshd, uses the configuration file /etc/ssh/sshd_config

The default config file is sufficient. For customization, refer to sshd man page for config options.

3. Edit/view the sshd_config file

```
# vi /etc/ssh/sshd_config
```

#Port 22 - Specifies the port number that **sshd** listens on
#Protocol 2,1 - Specifies the protocol versions **sshd** supports
#ListenAddress 0.0.0.0 - Specifies the local addresses **sshd** should listen on

#HostKey /etc/ssh/ssh_host_rsa_key - Specifies a file containing a private host key used by SSH

#SyslogFacility AUTH - Gives the facility code that is used when sshd logs messages
#LogLevel INFO - Gives the verbosity level that is used when sshd logs messages

#LoginGraceTime 600 – time limit for a user to log in
#PermitRootLogin yes - Specifies whether root can login using ssh
#StrictModes yes - Specifies whether **sshd** should check file modes and ownership of the user's files and home directory before accepting login
#PubkeyAuthentication yes - Specifies whether public key authentication is allowed
#PasswordAuthentication yes - Specifies whether password authentication is allowed
#PermitEmptyPasswords no - When password authentication is allowed, it specifies whether the server allows login to accounts with empty password strings
#MaxStartups 10 - Specifies the maximum number of concurrent unauthenticated connections to the **sshd** daemon
#KeepAlive yes - Specifies whether the system should send TCP keepalive messages to the other side
#VerifyReverseMapping no - Specifies whether **sshd** should try to verify the remote host name
Subsystem sftp /usr/libexec/openssh/sftp-server

Managing the sshd service

To start the service: `#/ /sbin/service sshd start`
To stop the service: `# /sbin/service sshd stop`
To restart the service: `# /sbin/service sshd restart`

To automatically run the service: `# chkconfig sshd on`

OpenSSH Client configuration:

To connect to an OpenSSH server from a client machine, you must have the openssh-clients and openssh packages installed on the client machine.

System wide ssh client configuration: defaults for all system wide users

`/etc/ssh_config`

The configuration values can be changed in per-user configuration files or on the command line:

`~/.ssh`

Using the ssh command with password authentication:

1. Make sure you have the following enabled in /etc/sshd_config file on the ssh server

`PasswordAuthentication yes` - Specifies whether password authentication is allowed

2. To log in to a host named server.com, type the following:

```
$ ssh ritesh@server.com
```

The first time you ssh to a remote machine, you will see a message similar to the following:

The authenticity of host 'server.com (202.52.255.22)' can't be established.
RSA key fingerprint is 0f:41:6c:52:31:59:43:0f:dd:49:5f:3d:47:9d:b5:9e.
Are you sure you want to continue connecting (yes/no)? yes

*Type **yes** to continue.*
This will add the server to your list of known hosts as seen in the following message:

Warning: Permanently added 'server.com,202.52.255.22' (RSA) to the list of knownhosts.
ritesh@server.com's password:
Last login: Thu Jan 16 19:34:13 2003 from mc-gw.mos.com.np
[ritesh@server.com ritesh]$

3. known_hosts file in .ssh folder contains the remote servers' host keys:

```
$ more known_hosts
server.com,202.52.255.22 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEA5IwKqHiqSeXjrX3dCpS1gXZo9GqJ0hkk+clf+WmABYbEH
6IGMyy2CeARtaR6QLpqB1SaGEFsn84dqA6kWLYfn4FuDVDc8KTyABLVEMOm6NnLZkHPKr3
Cb0RgivDYSYHlwgWuDi7XBvmoC44WA2EbM7eBy5h1kHrXZ5yPXq3rxI0=
```

Using the ssh command with public key authentication:

1. Make sure you have the following enabled in /etc/sshd_config file on the ssh server.

`PubkeyAuthentication yes` - specifies whether public key authentication is allowed

2. Key pair generation:

ssh-keygen - authentication key generation, management and conversion

To generate a RSA key pair to work with version 2 of the protocol:

```
$ ssh-keygen -t rsa
```

Generating public/private rsa key pair.
Enter file in which to save the key (/home/ritesh/.ssh/id_rsa):

Enter a passphrase different from your account password and confirm it by entering it again

Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ritesh/.ssh/id_rsa.
Your public key has been saved in /home/ritesh/.ssh/id_rsa.pub.
The key fingerprint is:
e5:a1:ac:ce:8d:16:3a:b9:3a:e4:e9:06:9c:90:b6:da ritesh@myhost.com

The public key is written to ~/.ssh/id_rsa.pub.
The private key is written to ~/.ssh/id_rsa.
Never distribute your private key to anyone.
Change the permissions of your .ssh directory using the command chmod 755 ~/.ssh

```
$ ll .ssh/
total 8
-rw-------  1 ritesh  ritesh   951 Jan 16 19:37 id_rsa
-rw-r--r--  1 ritesh  ritesh   236 Jan 16 19:37 id_rsa.pub
```

Copy the contents of ~/.ssh/id_rsa.pub to ~/.ssh/authorized_keys on the machine to which you want to connect. If the file ~/.ssh/authorized_keys does not exist, you can copy the file ~/.ssh/id_rsa.pub to the file ~/.ssh/authorized_keys on the remote SSH server.

Securely copy your public key file to the remote ssh server:

```
$ scp ~/.ssh/id_rsa.pub ritesh@server.com:
```

Logon to the remote ssh server using password authentication, one more time:

```
$ ssh ritesh@server.com
```

Copy the content of the public key file to authorized_keys file on the remote host:

```
$ cat id_rsa.pub >> ~/.ssh/authorized_keys
```

Now, logon to the remote ssh server via public key authentication:

```
$ ssh ritesh@server.com
```

Enter passphrase for key '/home/ritesh/.ssh/id_rsa':
Last login: Thu Jan 16 19:40:11 2003 from myhost.com
[ritesh@server.com ritesh]$

To run a command on the remote host with ssh and exit:

```
$ ssh penguin.example.net ls /usr/share/doc
```

Using the scp command
The scp command is used to transfer files between machines over a secure, encrypted connection

To transfer the local file shadowman to /home/*username*/shadowman on penguin.example.net:

```
$ scp shadowman username@penguin.example.net:/home/username
```

To transfer a remote file to the local system:

```
scp username@tohostname:/remotefile /newlocalfile
```

To transfer the contents of the directory /downloads to an existing directory called uploads on the remote machine penguin.example.net:

```
scp /downloads/* username@penguin.example.net:/uploads/
```

Using the sftp command
The sftp command is used to open a secure, interactive FTP session via an encrypted channel.

```
sftp username@hostname.com
```

Once authenticated, you can use a set of commands similar to using FTP

OpenSSH commands:

- ssh - The basic rlogin/rsh-like client program.
- sshd - The daemon that permits you to login.
- ssh-agent - An authentication agent that can store private keys.
- ssh-add - Tool which adds keys to in the above agent.
- sftp - FTP-like program that works over SSH1 and SSH2 protocol.
- scp - File copy program that acts like rcp(1).
- ssh-keygen - Key generation tool.
- sftp-server - SFTP server subsystem (started automatically by sshd).
- ssh-keyscan - Utility for gathering public host keys from a number of hosts.
- ssh-keysign - Helper program for hostbased authentication.

Iptables Lab

Installation:

Install iptables rpm package from the Redhat distribution CD.
It may also be installed by default during Redhat installation.

```
# rpm –ivh iptables-1.2.5-3.rpm
```

Using Iptables:

1. To view all iptables command line options:

```
# iptables –h
```

2. To list all current default rules/chains:

```
# iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

3. To set the default policies for all chains:

```
# iptables -P INPUT DROP
# iptables -P OUTPUT DROP
# iptables -P FORWARD DROP

# iptables -L

Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

Now, all packets will be dropped – no network traffic to/from the host
The default policy should be to DROP all packets not matched by any rules/chains.

4. To allow ping to work to/from your host to anywhere

```
# iptables -A INPUT -p ICMP -j ACCEPT
# iptables -A OUTPUT -p ICMP -j ACCEPT
```

List the iptables rules now:

```
# iptables -L

Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  anywhere             anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  anywhere             anywhere
```

5. To allow ping to work only across the firewall but not to/from it:

Flush the previous FORWARD chain rules:

```
# iptables -F FORWARD
```

Apply the new rule:

```
# iptables -A FORWARD -p ICMP -j ACCEPT
```

List the rules now:

```
# iptables -L --line-numbers

Chain INPUT (policy DROP)
num  target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     icmp --  anywhere             anywhere

Chain OUTPUT (policy DROP)
num  target     prot opt source               destination
```

6. To allow all internal users to access websites on the Internet:

```
# iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 80 -j ACCEPT
# iptables -A FORWARD -i eth0 -o eth1 -p tcp -m state --state
ESTABLISHED,RELATED -j ACCEPT
```

Note: eth0 = outside interface, eth1 = inside interface

7. To allow some external users access to SSH, SMTP, POP, HTTP, DNS servers in your internal network:

Inbound FORWARD rules:

```
# iptables -A FORWARD –s 202.52.250.0/24 –d 202.52.255.1 -i eth0 -o
eth1 -p tcp --dport 22 -j ACCEPT

# iptables -A FORWARD –s 202.52.250.0/24 –d 202.52.255.3 -i eth0 -o
eth1 -p tcp --dport 25 -j ACCEPT

# iptables -A FORWARD –s 202.52.250.0/24 –d 202.52.255.6 -i eth0 -o
eth1 -p tcp --dport 110 -j ACCEPT

# iptables -A FORWARD –s 202.52.250.0/24 –d 202.52.255.35 -i eth0 -o
eth1 -p tcp --dport 80 -j ACCEPT

# iptables –A FORWARD –s 202.52.250.0/24 –d 202.52.255.47 –I eth0 –o
eth1 –p udp –dport 53 –j ACCEPT
```

Outbound FORWARD rules:

```
# iptables -A FORWARD –s 202.52.255.0/24 -i eth1 -o eth0 -p tcp -m
state --state ESTABLISHED,RELATED -j ACCEPT

# iptables -A FORWARD –s 202.52.255.0/24 -i eth1 -o eth0 -p udp -m
state --state ESTABLISHED -j ACCEPT
```

8. To view all current rules with numeric addresses:ports and rule line numbers:

```
# iptables –L –n --line-numbers
```

9. To save all rules/chains to /etc/sysconfig/iptables to make permanent:

```
# service iptables save
```

10. To stop iptables and flush all rules:

```
# service iptables stop
```

11. To start and all saved rules:

```
# service iptables start
```

12. To load iptables at every system startup:

```
# chkconfig iptables on
```

NAT exercices:

1. Fixed IP address mapping (inbound) - maps the public IP of a server to the private IP of the internal server

```
# iptables -t -nat -A PREROUTING -i eth1 -d 202.52.255.5 -j DNAT –to-
destination 192.168.0.1
```

2. Port mapping (inbound) - maps port 80 of host with IP 202.52.255.5 to port 8080 of the internal host having IP 192.168.14.2

```
# iptables -t -nat -A PREROUTING -i eth0 -d 202.52.255.5 -p tcp -m tcp
–dport 80 -j DNAT –to-destination 192.168.0.1:8080
```

3. IP Masquerading (outbound) - translates the source IP of all outbound packets to 202.52.255.5, the IP of eth0

```
# iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

4. SNAT (outbound) - The source IP of all outbound packets will be converted to 202.52.255.5

```
# iptables -t nat -A POSTROUTING -o eth1 -s 192.168.0.0/24 -j SNAT
–to-source 202.52.255.5
```

5. Fixed IP mapping (outbound) - The source IP of 192.168.10.11 will be converted to 202.52.255.5 while exiting from eth0

```
# iptables -t nat -A POSTROUTING -o eth1 -s 192.168.0.11 -j SNAT
–to-source 202.52.255.5
```

Nmap Lab

Installation:

Install the nmap rpm package that comes with the RedHat CD distro or download it from www.insecure.org/nmap or www.rpmfind.net

```
# rpm -ivh nmap-3.00-1.i386.rpm
```

Usage:

1. This option scans all reserved TCP ports on the machine target.example.com . The -v means turn on verbose mode.

```
nmap -v target.example.com
```

2. Launches a stealth SYN scan against each machine that is up out of the 255 machines on class ´C´ where target.example.com resides. It also tries to determine what operating system is running on each host that is up and running. This requires root privileges because of the SYN scan and the OS detection.

```
nmap -sS -O target.example.com/24
```

3. Sends an Xmas tree scan to the first half of each of the 255 possible 8 bit subnets in the 198.116 class ´B´ address space. We are testing whether the systems run sshd, DNS, pop3d, mapd, or port 4564. Note that Xmas scan doesn't work on Microsoft boxes due to their deficient TCP stack. Same goes with CISCO, IRIX, HP/UX, and BSDI boxes.

```
nmap -sX -p 22,53,110,143,4564 198.116.*.1-127
```

4. Rather than focus on a specific IP range, it is sometimes interesting to slice up the entire Internet and scan a small sample from each slice. This command finds all web servers on machines with IP addresses ending in .2.3, .2.4, or .2.5

```
nmap -v --randomize_hosts -p 80 ´*.*.2.3-5´
```

5. Launch a stealth scan with OS detection on all privileged ports against 255 hosts in the network, output the results into the file /root/nmap.scan

```
nmap -sS -O 192.168.10.0/24 -oN /root/nmap.scan
```

6. Launch a stealth scan with OS detection on specified ports against 255 hosts in the network, in verbose mode.

```
nmap -sS -O -v 192.168.10.0/24 -p '1-1024,1080,3128'
```

Snort Lab

Installation:

1. Download snort from http://www.snort.org/dl/

2. `# rpm –ivh snort-1.9.0-1snort.i386.rpm`

Usage:

There are three main modes in which Snort can be configured:

Sniffer, packet logger, and network intrusion detection system.

Sniffer mode simply reads the packets off of the network and displays them for you in a continuous stream on the console.

Packet logger mode logs the packets to the disk.

Network intrusion detection mode is the most complex and configurable configurations, allowing Snort to analyze network traffic for matches against a user defined rule set and perform several actions based upon what it sees.

Sniffer Mode

First, let's start with the basics. If you just want to print out the TCP/IP packet headers to the screen (i.e. sniffer mode), try this:

```
./snort -v
```

This command will run Snort and just show the IP and TCP/UDP/ICMP headers, nothing else. If you want to see the application data in transit, try the following:

```
./snort -vd
```

This instructs Snort to display the packet data as well as the headers. If you want an even more descriptive display, showing the data link layer headers do this:

```
./snort -vde
```

(As an aside, these switches may be divided up or smashed together in any combination. The last command could also be typed out as:

```
./snort -d -v -e
```

and it would do the same thing.)

Packet Logger Mode

OK, all of these commands are pretty cool, but if you want to record the packets to the disk, you need to specify a logging directory and Snort will automatically know to go into packet logger mode:

```
./snort -dev -l ./log
```

Of course, this assumes you have a directory named "log" in the current directory. If you don't, Snort will exit with an error message. When Snort runs in this mode, it collects every packet it sees and places it in a directory hierarchy based upon the IP address of one of the hosts in the datagram.

If you just specify a plain "-l" switch, you may notice that Snort sometimes uses the address of the remote computer as the directory in which it places packets, and sometimes it uses the local host address. In order to log relative to the home network, you need to tell Snort which network is the home network:

```
./snort -dev -l ./log -h 192.168.1.0/24
```

This rule tells Snort that you want to print out the data link and TCP/IP headers as well as application data into the directory ./log, and you want to log the packets relative to the 192.168.1.0 class C network. All incoming packets will be recorded into subdirectories of the log directory, with the directory names being based on the address of the remote (non-192.168.1) host. Note that if both hosts are on the home network, then they are recorded based upon the higher of the two's port numbers, or in the case of a tie, the source address.
If you're on a high speed network or you want to log the packets into a more compact form for later analysis you should consider logging in "binary mode". Binary mode logs the packets in "tcpdump format" to a single binary file in the logging directory:

```
./snort -l ./log -b
```

Note the command line changes here. We don't need to specify a home network any longer because binary mode logs everything into a single file, which eliminates the need to tell it how to format the output directory structure. Additionally, you don't need to run in verbose mode or specify the -d or -e switches because in binary mode the entire packet is logged, not just sections of it. All that is really required to place Snort into logger mode is the specification of a logging directory at the command line with the -l switch, the -b binary logging switch merely provides a modifier to tell it to log the packets in something other than the default output format of plain ASCII text.

Once the packets have been logged to the binary file, you can read the packets back out of the file with any sniffer that supports the tcpdump binary format such as tcpdump or Ethereal. Snort can also read the packets back by using the -r switch, which puts it into playback mode. Packets from any tcpdump-formatted file can be processed through Snort in any of its run modes. For example, if you wanted to run a binary log file through Snort in sniffer mode to dump the packets to the screen, you can try something like this:

```
./snort -dv -r packet.log
```

Network Intrusion Detection Mode

To enable network intrusion detection (NIDS) mode (so that you don't record every single packet sent down the wire), try this:

```
./snort -dev -l ./log -h 192.168.1.0/24 -c snort.conf
```

Where snort.conf is the name of your rules file. This will apply the rules set in the snort.conf file to each packet to decide if an action based upon the rule type in the file should be taken. If you don't specify an output directory for the program, it will default to /var/log/snort.
One thing to note about the last command line is that if Snort is going to be used in a long-term way as IDS, the "-v" switch should be left off the command line for the sake of speed. The screen is a slow place to write data to, and packets can be dropped while writing to the display.
It's also not necessary to record the data link headers for most applications, so it's not necessary to specify the -e switch either.

```
./snort -d -h 192.168.1.0/24 -l ./log -c snort.conf
```

This will configure Snort to run in it's most basic NIDS form, logging packets that the rules tell it to in plain ASCII to a hierarchical directory structure (just like packet logger mode).

NIDS Mode Output Options

There are a number of ways to configure the output of Snort in NIDS mode. The default logging and alerting mechanisms are to log in decoded ASCII format and use "full" alerts. The full alert mechanism prints out the alert message in addition to the full packet headers. There are several other alert output modes available at the command line, as well as two logging facilities.
Alert modes are somewhat more complex. There are six alert modes available at the command line, full, fast, socket, syslog, smb (WinPopup), and none. Four of these modes are accessed with the -A command line switch. The four options are:

-A fast
fast alert mode, write the alert in a simple format with a timestamp, alert message, source and destination IPs/ports

-A full
this is also the default alert mode, so if you specify nothing this will automatically be used

-A unsock
send alerts to a UNIX socket that another program can listen on

-A none
turn off alerting

Packets can be logged to their default decoded ASCII format or to a binary log file via the -b command line switch. If you wish to disable packet logging all together, use the -N command line switch.

For output modes available through the configuration file, note that command line logging options override any output options specified in the configuration file. This allows debugging of configuration issues quickly via the command line.

To send alerts to syslog, use the "-s " switch. The default facilities for the syslog alerting mechanism are LOG_AUTHPRIV and LOG_ALERT. If you want to configure other facilities for syslog output, use the output plugin directives in the rules files.

Finally, there is the SMB alerting mechanism. This allows Snort to make calls to the smbclient that comes with Samba and send WinPopup alert messages to Windows machines. To use this alerting mode, you must configure Snort to use it at configure time with the -enable-smbalerts switch.

Here are some output configuration examples:

- Log to default (decoded ASCII) facility and send alerts to syslog

```
./snort -c snort.conf -l ./log -h 192.168.1.0/24
```

- Log to the default facility in /var/log/snort and send alerts to a fast alert file:

```
./snort -c snort.conf -s -h 192.168.1.0/24
```

- Log to a binary file and send alerts to Windows workstation:

```
./snort -c snort.conf -b -M WORKSTATIONS
```

High Performance Configuration

If you want Snort to go *fast* (like keep up with a 100 Mbps net fast) use the "-b" and "-A fast" or "-s" (syslog) options. This will log packets in tcpdump format and produce minimal alerts.
For example:

```
./snort -b -A fast -c snort.conf
```

In this configuration, Snort has been able to log multiple simultaneous probes and attacks on a 100 Mbps LAN running at saturation level of approximately 80 Mbps. In this configuration, the logs are written in binary format to the snort.log tcpdump-formatted file. To read this file back and break out the data in the familiar Snort format, just rerun Snort on the data file with the "-r" option and the other options you would normally use.
For example:

```
./snort -d -c snort.conf -l ./log -h 192.168.1.0/24 -r snort.log
```

Once this is done running, all of the data will be sitting in the log directory in its normal decoded format.

Changing Alert Order

Some people don't like the default way in which Snort applies it's rules to packets. The Alert rules applied first, then the Pass rules, and finally the Log rules. This sequence is somewhat counterintuitive, but it's a more foolproof method than allowing the user to write a hundred alert rules and then disable them all with an errant pass rule.

For people who know what they're doing, the "-o" switch has been provided to change the default rule application behavior to Pass rules, then Alert, then Log:

```
./snort -d -h 192.168.1.0/24 -l ./log -c snort.conf -o
```

Miscellaneous

If you are willing to run snort in "daemon" mode, you can add -D switch to any combination above. Please NOTICE that if you want to be able to restart snort by sending SIGHUP signal to the daemon, you will need to use full path to snort binary, when you start it, i.g.:

```
/usr/local/bin/snort -d -h 192.168.1.0/24 -l  \
          /var/log/snortlogs -c /usr/local/etc/snort.conf -s -D
```

Relative paths are not supported due to security concerns.
If you're going to be posting packet logs to public mailing lists you might want to try out the -O switch. This switch "obfuscates" your the IP addresses in the packet printouts. This is handy if you don't want the people on the mailing list to know the IP addresses involved. You can also combine the -O switch with the -h switch to only obfuscate the IP addresses of hosts on the home network. This is useful if you don't care who sees the address of the attacking host. For example:

```
./snort -d -v -r snort.log -O -h 192.168.1.0/24
```

This will read the packets from a log file and dump the packets to the screen, obfuscating only the addresses from the 192.168.1.0/24 class C network.