# Improving OpenSSH security

OpenSSH is the swiss-army knife of remote-access programs: it provides you with a shell on your distant machine, and transmits data in a secure and encrypted way - including commands, file transfer, X11 and VNC sessions, rsync data, etc.

OpenSSH is so advanced that it can be considered as a simplified VPN (Virtual Private Network).

Slackware contains OpenSSH of course, and the default configuration is already pretty secure. This page has been created to show you a few simple configuration tweaks that can be applied to the default configuration to improve its security.

You will need to know how to use a text editor to be able to follow this HOWTO. If you are a complete beginner, try `nano`. Once you are more advanced, and more at ease with Linux, feel free to use something more powerful...

## The SSH configuration files and finding more information

The OpenSSH configuration files reside in the directory `/etc/ssh/`. The most important one is the `/etc/ssh/sshd_config` that we are going to modify here.

The OpenSSH documentation is very good, so feel free to enter the command: `apropos openssh` or `man -k openssh` and read the different man pages, which are much more detailed than this wiki page.

## Modifying the sshd_config file

The first rule, before you modify an important configuration file is to make a backup copy of it. For instance:

```
# cp -v /etc/ssh/sshd_config /etc/ssh/sshd_config.ORIG.20120826
```

The command shown above will make a copy of the `sshd_config` file, and append the extension `.ORIG` (for original, of course) and include a modification date. While not perfect, this system insures that you can always go back to the previous version of an important file by entering the following:

```
# cp -v /etc/ssh/sshd_config.ORIG.20120826 /etc/ssh/sshd_config
```

Please note that this is a suggestion, of course, and system administrators that have to manage large installation, with hundreds of servers, are strongly advised to investigate better solutions, such as Puppet or other distributed system configuration managers.

Now, edit the `/etc/ssh/sshd_config` file with your favourite editor and tweak the following lines:

# Change the SSH default port

By default, OpenSSH listens on port 22. It is sometimes advised to change this default port to something different, such as 2222 or 4242. This is not a bad idea, but you have to remember that scanning your machine with a program such as `nmap` will obtain this new port very quickly. So, while it may slow down some attacks on your machine, it will not prevent them entirely.

If you'd like to change the port, search for the `Port` option in `sshd_config`, which is usually at the top of file, and change the value.

For instance:

```
Port 22
```

Can be changed to:

```
Port 4242
```

# Alternate method of Changing the SSH default port without changing

This suggestion is completely lifted from another site http://null.redcodenetwork.ro/changing-the-ssh-port-without-changing-it/ The idea here is add a call to this script in rc.local so it will run at start up.

To use Download sample at the bottom change as you like and save it in /etc/rc.d/ (make it executable to use) Add this sample below to /etc/rc.d/rc.local

#add ssh_hide to port 8889

```
if [ -x /etc/rc.d/rc.ssh_hide ]; then
  /etc/rc.d/rc.ssh_hide
fi
```

What it is doing is making it look like you changed the port ssh is using and provide some additional security. What happens is scanners will continue to see port 22 open and try to go there while your server drops those packets, because the header is not mangled. Real packets come in on port 8889 and are redirected by iptables to port 22 with mangle in the header so they don't get dropped.

rc.ssh_hide

```
#!/bin/bash

######First, set SSHD back to the default port 22.
######Next, figure out what port or ports you want to do SSH over.
######Were going to use 99, 88, and 8889 here.
######Now we take care of the Hypothetical Evil Unprivileged User
######by not accepting anything over those ports in the first place.
```

```
######This is only effective for port 8889 but well do all three ports
for the sake of completeness.

/usr/sbin/iptables -t filter -A INPUT -p tcp -m multiport --dports
99,88,8889 -j REJECT --reject-with tcp-reset

######Then, pick a number between 1 and 4294967295 Ill use 0x13F ()
######Were going to tell iptables to reject anything without this mark
coming into port 22.

/usr/sbin/iptables -t filter -A INPUT -p tcp -m tcp --dport 22 -m
connmark ! --mark 0x13F -j REJECT --reject-with tcp-reset

######Now well tell iptables what ports we will accept for ssh.

/usr/sbin/iptables -t filter -A FORWARD -p tcp -m multiport --dports
99,88,8889 -j ACCEPT

######In the mangle table we slap our mark on these packets.

/usr/sbin/iptables -t mangle -A PREROUTING -p tcp -m multiport --dports
99,88,8889 -j CONNMARK --set-mark 0x13F

######Finally in the nat table we tell iptables to send the marked
packets back to port 22

/usr/sbin/iptables -t nat -A PREROUTING -p tcp -m multiport --dport
99,88,8889 -j REDIRECT --to-ports 22

exit 0
```

Note not ipv6 compatible, due to nat (ip6tables not supporting nat)

## Forbid root access to your machine

This is probably the single most important change you can do to improve the security of your machine: forbid the `root` user to access your machine. To do this, look for the following line in `sshd_config`:

```
PermitRootLogin yes
```

And change it to:

```
PermitRootLogin no
```

If you apply the changes above, make sure you have at least one user on your machine that can `su` (switch user) to `root` or use `sudo` for more fine-grained system administration permissions. The best way to administrate a server through SSH is to have a user in the `wheel` group, who can use both `sudo` and `su` to become `root` when need be.

Since most people who try to get into your machine uninvited use brute-force scripts that target the `root` login, you can already rest a lot easier, knowing this access is locked.

## Improve login security and timing

Above and below the `PermitRootLogin` option, you will find other permissions, that we are going to detail quickly here:

- `LoginGraceTime` is used to increase, or decrease, the time left to a user to log into the machine. This can be safely be restricted to 5 minutes, with the following:

```
LoginGraceTime 5m
```

- `MaxAuthTries` is used to increse, or decrease, the number of tries allowed to a user to authentify correctly to the machine. This can be restricted to 3 attempts by using:

```
MaxAuthTries 3
```

## Deny X11 forwarding

Unless you need to use X11 over SSH, you can safely disable it by using the following option:

```
X11Forwarding no
```

Please note that disabling X11 does not disable VNC over SSH, for instance. In most cases, it is always a good idea to disable un-needed services.

## Restrict SSH login to approved users

If you are sure only specific users need to connect through SSH to your machine, you can declare them using the `AllowUsers` option. All the users - and only the users - mentioned after the option will be able to connect through SSH to the machine.

```
AllowUsers jack backup betty
```

In the example shown above, only user `jack`, `betty` and `backup` will be able to use SSH to connect to the machine. All other users will be denied access. Of course, you should use this option with caution and make sure the (or “a”) default user is included…

## Restart the SSH server

The last thing to do, to make sure the SSH server takes the new configuration into account, is to restart the SSH server with the command:

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

Your new, and slightly more secure OpenSSH configuration is now in effect. Congratulations!

## See Also

- OpenSSH manual pages (on-line)

# Sources

- Originally written by User Noryungi

howtos, security, ssh, author Noryungi