

Ayub Roti Technical Solutions Engineer, Serianu Limited.

5 SSH Hardening Tips | Linux.com | The source for Linux information

Content Courtesy of

<https://www.linux.com/LEARN/5-SSH-HARDENING-TIPS>

When you look at your SSH server logs, chances are they are full of attempted logins from entities of ill intent. Here are 5 general ways (along with several specific tactics) to make your OpenSSH sessions more secure.

1. Make Password Auth Stronger

Password logins are convenient, because you can log in from any machine anywhere. But they are vulnerable to brute-force attacks. Try these tactics for strengthening your password logins.

- Use a password generator, such as **pwgen**. **pwgen** takes several options; the most useful is password length (e.g., **pwgen 12** generates a 12-character password).
- Never reuse a password. Ignore all the bad advice about not writing down your passwords, and keep a notebook with your logins written in it. If you don't believe me that this is a good idea, then believe security guru Bruce Schneier (https://www.schneier.com/blog/archives/2005/06/write_down_your.html). If you're reasonably careful, nobody will ever find your notebook, and it is immune from online attacks.
- You can add extra protection to your login notebook by obscuring the logins recorded in your notebook with character substitution or padding. Use a simple, easily-memorable convention such as padding your passwords with two extra random characters, or use a single simple character substitution such as # for *.
- Use a non-standard listening port on your SSH server. Yes, this is old advice, and it's still good. Examine your logs; chances are that port 22 is the standard attack point, with few attacks on other ports.
- Use Fail2ban (http://www.fail2ban.org/wiki/index.php/Main_Page) to dynamically protect your server from brute force attacks.
- Create non-standard usernames. Never ever enable a remote root login, and avoid "admin".

2. Fix Too Many Authentication Failures

When my ssh logins fail with "Too many authentication failures for carla" error messages, it makes me feel bad. I know I shouldn't take it personally, but it still stings. But, as my wise granny used to say, hurt feelings don't fix the problem. The cure for this is to force a password-based login in your `~/.ssh/config` file. If this file does not exist, first create the `~/.ssh/` directory:

```
$ mkdir ~/.ssh
$ chmod 700 ~/.ssh
```

Then create the `~/.ssh/config` file in a text editor and enter these lines, using your own remote HostName address:

```
HostName remote.site.com
PubkeyAuthentication=no
```

3. Use Public Key Authentication

Public Key authentication is much stronger than password authentication, because it is immune to brute-force password attacks, but it's less convenient because it relies on RSA key pairs. To begin, you create a public/private key pair. Next, the private key goes on your client computer, and you copy the public key to the remote server that you want to log into. You can log in to the remote server only from computers that have your private key. Your private key is just as sensitive as your house key; anyone who has possession of it can access your accounts. You can add a strong layer of protection by putting a passphrase on your private key.

Using RSA key pairs is a great tool for managing multiple users. When a user leaves, disable their login by deleting their public key from the server.

This example creates a new key pair of 3072 bits strength, which is stronger than the default 2048 bits, and gives it a unique name so you know what server it belongs to:

```
$ ssh-keygen -t rsa -b 3072 -f id_mailserver
```

This creates two new keys, `id_mailserver` and `id_mailserver.pub`. `id_mailserver` is your private key -- do not share this! Now securely copy your public key to your remote server with the **ssh-copy-id** command. You must already have a working SSH login on the remote server:

```
$ ssh-copy-id -i id_rsa.pub user@remoteserver
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
user@remoteserver's password:

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'user@remoteserver'"
and check to make sure that only the key(s) you wanted were added.
```

ssh-copy-id ensures that you will not accidentally copy your private key. Test your new key login by copying the example from your command output, with single quotes:

```
$ ssh 'user@remoteserver'
```

It should log you in using your new key, and if you set a password on your private key, it will prompt you for it.

4. Disable Password Logins

Once you have tested and verified your public key login, disable password logins so that your remote server is not vulnerable to brute force password attacks. Do this in the `/etc/ssh/sshd_config` file on your remote server with this line:

```
PasswordAuthentication no
```

Then restart your SSH daemon.

5. Set Up Aliases -- They're Fast and Cool

You can set up aliases for remote logins that you use a lot, so instead of logging in with something like "ssh -u username -p 2222 remote.site.with.long-name", you can use "ssh remote1". Set it up like this in your `~/.ssh/config` file:

```
Host remotel
HostName remote.site.with.long-name
Port 2222
User username
PubkeyAuthentication no
```

If you are using public key authentication, it looks like this:

```
Host remotel
HostName remote.site.with.long-name
Port 2222
User username
IdentityFile ~/.ssh/id_remoteserver
```

The OpenSSH documentation (<http://www.openssh.com/>) is long and detailed, but after you have mastered basic SSH use, you'll find it's very useful and contains a trove of cool things you can do with OpenSSH.

Ayub Roti Technical Solutions Engineer, Serianu Limited.

5 quick SSH hardening tips - TechRepublic

Content Courtesy of

<https://www.techrepublic.com/article/5-quick-ssh-hardening-tips/>

If you make use of Secure Shell, you'll want to run down this checklist of five quick tips to make that Linux server a bit more secure.



Image: Jack Wallen

If you're a Linux administrator, more than likely you depend upon Secure Shell (SSH) for remote access to your data center or other business machines. SSH is, by design, a fairly secure protocol. However, there are ways you can make it even more secure. In fact, I have five very quick tips you can use to better lock down that SSH server. These can be done in just a few minutes.

Are you ready?

SEE: Windows 10 security: A guide for business leaders (<http://www.techproresearch.com/downloads/windows-10-security-a-guide-for-business-leaders/>) (Tech Pro Research)

1. Set idle timeout interval

The idle timeout interval is the amount of time in which an ssh session is allowed to sit idle. When that timeout passes, the connection is broken. Out of the box, this option is disabled. We'll enable it and set a time out of five minutes (300 seconds). To do this, issue the command:

```
sudo nano /etc/ssh/sshd_config
```

In this file, look for:

```
#ClientAliveInterval 0
```

Change that to:

```
ClientAliveInterval 300
```

Save and close the file. Restart the SSH server with the command:

```
sudo systemctl restart sshd
```

2. Disable empty passwords

There are some system user accounts that are created without passwords. The administrator of a Linux machine can also create standard users without passwords. Out of the box, SSH is configured such that it doesn't prevent empty passwords from being allowed. Let's fix that.

Open the SSH daemon configuration file again with the command:

```
sudo nano /etc/ssh/sshd_config
```

Locate the line:

```
#PermitEmptyPasswords no
```

Change that to:

```
PermitEmptyPasswords no
```

Save and close the file. Restart the SSH server with the command:

```
sudo systemctl restart sshd
```

3. Disable X11 forwarding

If you have servers with GUI interfaces, or you have desktop machines that require the usage of SSH, you should probably disable X11 forwarding. What is X11 forwarding? This allows anyone to tunnel GUI applications via SSH. The last thing you want is for a malicious user to easily view sensitive information via GUI, or exploit this already insecure feature.

To disable this feature, open the SSH daemon configuration file again with the command:

```
sudo nano /etc/ssh/sshd_config
```

Look for the line:

```
X11Forwarding yes
```

Change the above to:

```
X11Forwarding no
```

Save and close the file. Restart the SSH server with the command:

```
sudo systemctl restart sshd
```

4. Limit max authentication attempts

By setting a low threshold for login attempts, you can help prevent against brute force attacks. Open the SSH daemon configuration file again with the command:

```
sudo nano /etc/ssh/sshd_config
```

Look for the line:

```
#MaxAuthTries 6
```

Change that line to:

```
MaxAuthTries 3
```

Save and close the file. Restart the SSH server with the command:

```
sudo systemctl restart sshd
```

5. Disable SSH on desktops

If you have desktop Linux machines, you might want to consider disabling SSH. Why? What if a malicious user were to log into one of those desktop machines (because they probably have lower security than your servers) and then use that machine as a relay to gain access to your servers? You don't want that. Period.

In fact, instead of disabling SSH, you might consider completely removing the SSH server. To do that, issue one of the following commands:

- *sudo dnf remove openssh-server* - on Fedora-based systems.
- *sudo apt-get remove openssh-server* - on Debian/Ubuntu-based systems.

If you don't want to completely remove the SSH server, disable it with the commands:

```
sudo systemctl stop sshd
sudo systemctl disable sshd
```

Simple SSH security

And there you have it: Five simple (and quick) ways of gaining a bit of added SSH security on your servers and desktops. Once you've taken care of these steps, do not think your data center and other business servers are perfectly safe. Always be diligent and on the lookout for suspicious events via log files and other means.

Cybersecurity Insider Newsletter

Strengthen your organization's IT security defenses by keeping abreast of the latest cybersecurity news, solutions, and best practices. Delivered Tuesdays and Thursdays

Sign up today

Sign up today

Also see

Ayub Roti Technical Solutions Engineer, Serianu Limited.

OpenSSH security and hardening - Linux Audit

Content Courtesy of

<https://linux-audit.com/audit-and-harden-your-ssh-configuration/>

SSH or Secure Shell (https://en.wikipedia.org/wiki/Secure_Shell) is the popular protocol for doing system administration on Linux systems. It runs on most systems, often with its default configuration. As this service opens up a potential gateway into the system, it is one of the steps to hardening a Linux system (<https://linux-audit.com/how-to-secure-linux-systems-auditing-hardening-and-security/>). This article covers the SSH security tips to secure the OpenSSH service and increase the defenses of the system.

OpenSSH security

OpenSSH is under development by the security fanatics from the OpenBSD project (<https://www.openbsd.org/>). Every new piece of functionality is created with care, especially when it comes to security (<https://www.openssh.com/security.html>). Although there were some vulnerabilities, OpenSSH is fairly secure by default. There are still some steps left that can be improved. During research for the security auditing tool Lynis, we looked also at the available OpenSSH settings. Besides the tests that are now in Lynis, this article is one of the other results of that research.

What will be covered?

We will be covering both the server and client configuration. The configuration syntax and settings are based on OpenSSH 7.x. The examples should be working for most Linux distributions like CentOS, Debian, Ubuntu, and RHEL. You can expect this to be also the case for FreeBSD, OpenBSD, and other systems that use OpenSSH. When in doubt, consult your man page. If you discovered an error or exception, let it know via the comments. Your feedback is welcome.

After reading this article, you will know:

- Where the client settings and server settings are stored
- How to see the active and default settings
- How to test your configuration settings
- Make an informed decision on how to secure SSH
- Which tools can help audit SSH and apply best practices

SSH basics

SSH has two parts: the **server daemon** (sshd) that runs on a system and the **client** (ssh) used to connect to the server. Typically administration is done by using an SSH client from a workstation. If you are on Windows, then often you will be using something like Putty (<https://www.chiark.greenend.org.uk/~sgtatham/putty/>).

When it comes to the security of the SSH configuration, it is the server part that is the most interesting. For example, is that the server can decide if normal password based logins are allowed or denied. Even if the client has a preference, it is the server to make the final call. The server configuration file is located at **/etc/ssh/sshd_config**.

The client configuration settings can be found in **/etc/ssh/ssh_config** (system wide) or **~/.ssh/config** (per user). Settings can also be specified during the connection by providing a command-line option.

Before we start making changes, let's start with some tips to do it right.

Deployment tips

Do (not) use best practices

The web is full of blogs and guides that state they are using so-called *best practices*. A best practice is an effective and good approach and typically agreed on by the experts and by consensus. Unfortunately, many of the blogs and articles are simple copies from other blogs and without the extensive research. So I strongly suggest to look up some background of the blog and author first.

If you see just configuration settings without a good explanation, be careful with applying such changes. Some are outdated or simply not relevant. What is the purpose of setting some value when it is already the default or even removed? So whatever you do, apply critical thinking and don't make assumptions. So use best practices, but always test your changes.

Check the status of SSH

Is this the first time you will change your SSH configuration? Check the status of the SSH daemon and see if the related service is started on boot. When using a distribution with systemd, make sure the daemon is running and enabled.

```
systemctl status ssh.service
```

Note: on some Linux distributions the service is named `sshd.service`.

The output should contain the enabled value.

```
Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
```

To see if SSH is running, look at the next line.

```
Active: active (running) since Mon 2018-06-04 17:18:33 CEST; 1 months 2 days ago
```

Use the SSH configuration test

If you make changes to your SSH configuration, it makes sense to restart the service. I strongly recommend to always check your configuration (`sshd_config`) first. This can be done by using the *test mode* flag. This additional step ensures the syntax and options are correct before you end up with a nonfunctioning service.

```
sshd -t
```

This command should not return any text or errors. Here is an example when something does not look good:

Screenshot of `sshd -t` command to test SSH configuration

Make sure to test your settings first.

Making changes to a remote system

Are you connected to the system with SSH and making changes to its configuration? Instead of restarting, consider sending a *reload* command to the running SSH daemon. This decreases the chance that you lose your connection and can't reconnect.

For systems using systemd, use `systemctl` to reload the SSH service.

```
systemctl reload ssh.service
```

The alternative is to manually send a *SIGHUP* to the SSH daemon. Do not to send this to any of the child processes, or you will be disconnected.

```
kill -HUP 1234
```

Another option is to temporarily run another SSH process on another port, without becoming a daemon process. Specify the full path and use `-D` together with the `-p` for the port number. Then ensure that you can access the temporary connection, especially if you are using a firewall with traffic filtering.

```
/usr/sbin/sshd -D -p 2222
```

Use `CTRL + C` to stop the process after you are done.

Deploy in small steps

While it makes sense to do a full deployment of your new SSH configuration to all systems, you might want to be careful. One example is that some older SSH clients can't use the newer key types. So have a look at the oldest Linux distributions that are used to get an idea on compatibility issues.

Show active SSH connections

Before applying changes or restarting the daemon, check for any active SSH connections. This can be done with the `ss` tool (<https://linux-audit.com/alternative-netstat-ss-tool/>).

```
ss -n -o state established '( dport = :22 or sport = :22 )'
```

Any established TCP connection will be displayed. By using both *dport* and *sport*, we can confirm what connections are active in both directions.

Securing the SSH server configuration

Preparations

Before we start making changes to our configuration, let's make a backup.

```
cp /etc/ssh/sshd_config /root/sshd_config
```

After that is done, it is good to know that each OpenSSH version has its own defaults. New features may have been added, older settings may have disappeared. To know if a specific setting is set, don't rely on the configuration file. Instead, call the SSH daemon with the *extended test mode* flag `-T` to show all details.

```
sshd -T
```

The output may look something like this:

Screenshot of output from `sshd -T` command to show active SSH settings
Show active and default settings of the OpenSSH daemon

Note: configuration settings and values are displayed with lowercase characters.

SSH security settings

Use of X11Forwarding

The display server on the client might have a higher exposure to be attacked with X11 traffic forwarded. If forwarding of X11 traffic is not needed, disable it:

```
X11Forwarding no
```

Why disabling X11Forwarding matters: the X11 protocol was never built with security in mind. As it opens up channel back to the client, the server could send malicious commands back to the client. To protect clients, disable X11Forwarding when it is not needed.

Disable rhosts

While not common anymore, *rhosts* was a weak method to authenticate systems. It defines a way to trust another system simply by its IP address. By default, the use of *rhosts* is already disabled. Make sure to check if it really is.

```
IgnoreRhosts yes
```

DNS hostname checking

By default, the SSH server can check if the client connecting maps back to the same combination of hostname and IP address. Use the option *UseDNS* to perform this basic check as an additional safeguard.

```
UseDNS yes
```

Note: this option may not work properly in all situations. It could result in an additional delay, as the daemon is waiting for a timeout during the initial connection. Only use this when you are sure your internal DNS is properly configured.

Disable empty passwords

Accounts should be protected and users should be accountable. For this reason, the usage of empty passwords should not be allowed. This can be disabled with the *PermitEmptyPasswords* option, which is the default.

```
PermitEmptyPasswords no
```

If you see this option enabled, then check which user accounts have no password set.

Maximum authentication attempts

To protect against brute-force attacks on the password of a user, limit the number of attempts. This can be done with the *MaxAuthTries* setting.

```
MaxAuthTries 3
```

Also enable monitoring for authentication failures, which starts at the half the number of maximum attempts. Use these authentication failures together with your SIEM solution, or forward them to your security administrator.

The SSH server can be configured to be used together with PAM or pluggable authentication modules. By using a set of rules, part of the authentication stack, the number of failed logins (https://linux-audit.com/locking-users-after-failed-login-attempts-with-pam_tally2/) can be used to block a particular user. Another option is to define a period to lock the account when this number of attempts has been reached. This way the server can defend better against brute-force attempts to crack a user account and its password.

When limiting the maximum authentication attempts, be aware that public key authentication (see below) can also eat up your number of attempts. If you want to enforce the SSH client (or SCP) to use password-based authentication, use the related options on the command line.

```
ssh -o PreferredAuthentications=password -o PubkeyAuthentication=no username@system
```

Public key authentication

Instead of using a normal password-based login, a better way is using public key authentication. Keys are considered much safer and less prone to brute-force attacks. Disable **PasswordAuthentication** to force users using keys.

```
PubkeyAuthentication yes
PasswordAuthentication no
```

Refer to the article Using SSH keys instead of passwords (<https://linux-audit.com/using-ssh-keys-instead-of-passwords/>), to set up key-based authentication.

Disable root login

It is best practice not to log in as the root user. Use a normal user account to initiate your connection instead, together with sudo. Direct root logins may result in bad accountability of the actions performed by this user account.

```
PermitRootLogin no
```

Newer versions of OpenSSH also support the value **without-password**. This value refers to methods like public key authentication. If your installation comes with this value, there is no reason to change it.

Set SSH protocol

If you are running an older system, version 1 of the SSH protocol might still be available. This version has weaknesses and should no longer be used. Since version 7.0 of OpenSSH, protocol 1 is automatically disabled during compile time. If your version is older than that, enforce the protocol version:

```
Protocol 2
```

Usage of AllowUsers and DenyUsers

When not all users should have access to the system, limit the number of people who can actually log in. One way is to create a group (e.g. sshusers) and add people to this group. Next set the **AllowGroups** option to define that only these users can log in.

Other possibilities include to only allow a few users with the **AllowUsers**, or specifically deny users and groups with the **DenyUsers**, or **DenyGroups**. Whitelisting access, using the 'default deny' principle, is usually better. So when possible, use the *AllowUsers* or *AllowGroups* option.

Good to know: SSH applies the following order to determine if one can log in: DenyUsers, AllowUsers, DenyGroups, finally AllowGroups.

Use HashKnownHosts

Each time the SSH client connects to a server, it will store a related signature (a key) of the server. This information is stored in a file with the name **known_hosts**. The known_hosts file itself is available in the **.ssh** subdirectory of the related user (on the client). In the case the signature of the server changes, SSH will protect the user by notifying about this chance. This option is useful but also has a risk. Previously it was common to store the hostname related to the specific host key. This made it easy for worms and other malicious scripts to use this information and spread to other systems, once they had a single system compromised. To counter this, the HashKnownHosts will hash each host, so it's not readable anymore. While being unreadable for the human eye, it still allows SSH to check for the next time you connect to the same system, as the results in the same hash.

Example output:

```
|1|XV5CFMH8LLIQPq7PxdBhGX7I9PA=|VKNLdODsQIJ/j4cvTZncqs9vgh0= ecdsa-sha2-
nistp256 AAAAE2VjZHNhLX....dJ/RzzZLH8Hs0UgroC0=
```

Restrict allowable commands

OpenSSH allows restricting the commands that a user can run via the **command** option. This is placed in the authorized_keys file, together with the allowable command and other options.

```
command="ps",no-agent-forwarding,no-port-forwarding,no-x11-forwarding, TYPE_OF_KEY KEY COMMENT
```

In the example above, replace the TYPE_OF_KEY, KEY, and COMMENT fields. The values that are to be used are similar to when using public key authentication.

Additional restrictions

Configure your firewall

Besides adjusting the SSH configuration, consider also limiting access by using traffic filtering. A local firewall like iptables or nftables can be used to restrict access to only allowed systems. Restrict access by only allowing those IP addresses that are trusted.

Use a jump server

Bigger environments typically restrict access by using a jump server or jump host. You may be familiar with them with other names like *stepping stone server* or *bastion host*. They are then the only systems within the network that are configured to allow access to other systems. That means if you want to do system administration, you always connect first to the jump server. From

there you will be connecting with the target system. A great combination with the previous tip to limit access with firewalling.

OpenSSH client security settings

As there are many SSH clients available, it would be impossible to cover them all in this article. Instead, we will have a look at the OpenSSH client tool.

Client configuration

The OpenSSH client has three ways to be configured. They are processed in order and checked for every available configuration setting. The first match wins.

1. Options provided via the command-line
2. Configuration file in the home directory (`~/.ssh/config`)
3. Configuration file for all users (`/etc/ssh/ssh_config`)

Let's say there is a setting named `A`. `A` is configured system-wide (option 3) with the value of `'True'`. User *michael* has it configured (option 2) being `'False'`. In that case, the latter would win. The reason is that it is considered before the system-wide configuration.

See the default and active client settings

Remember the trick to see the settings for the server (`ssh -T`)? The client has a similar one, although with a different character.

```
ssh -G abc
```

The `'abc'` in this example is just a random hostname. Ok, it is not really *that* random. You can use anything you want, including a real hostname. The client can use `Host` and `Match` blocks to customize the configuration to a group of systems or an individual system. As the host `'abc'` does not exist, that means the default settings will be parsed.

SSH settings for a single system

Let's say we have a system with the name **secureserver**. Instead of running on port 22, it accepts SSH connections on port 2222. Instead of using `-p` on the command line each time, we can add a **Host** block to our configuration file. So if you want to do this for your user, create the **config** file in your home directory, below the **.ssh** directory (so `/home/username/.ssh/config`).

Next step is creating a block and define the related settings that you want to use.

```
Host secureserver
  Hostname hostname.example.org
  User mynickname
  Port 2222
  MACs hmac-sha2-512
  KexAlgorithms curve25519-sha256@libssh.org
```

Indenting with spaces is not required. I would still advice to do it, so you see which settings belong to what host definition.

One question remains: What settings should you use in your client configuration file?

I suggest applying changes that make your daily work easier. So if you prefer security, set strong defaults. If a particular host is using a different SSH port, creating a `Host` block and overrule it that way. Regarding **KexAlgorithms**, use the newer algorithms that are available. This strongly depends on OpenSSH version on the other systems. If you have fairly new OpenSSH versions on the server, then the `curve25519` (<https://en.wikipedia.org/wiki/Curve25519>) is a good option. It is a high-speed elliptic-curve that is considered secure (at this moment).

Tools for SSH security

While it is good to manually harden a system, software and the related configurations can change over time. For that reason, it is helpful to perform a regular security scan.

Lynis

This open source security tool (<https://cisofy.com/lynis/>) is an allrounder when it comes to testing the security of your Linux system. From the bootloader to your web server, it will check as much as it can. It is free to use and written in shell script. Lynis runs on the system itself, so it can look both in the configuration files and the actually loaded configuration. It includes several tests focused on OpenSSH and its configuration, including security-related settings. Findings or possible improvements are displayed on the screen, so you can directly get into action and start hardening your system.

Download the tool via GitHub (<https://github.com/CISOfy/lynis>) or from the website (<https://cisofy.com/downloads/lynis/>). Never used the tool before, then use the Get Started (<https://cisofy.com/documentation/lynis/get-started/>) guide.

ssh-audit

Although slightly outdated, the `ssh-audit` (<https://github.com/arthepsy/ssh-audit>) tool is a great one to have in your toolbox. Instead of testing on the host itself, it can connect to an SSH server via the network. It performs its testing on the selected target and looks at the responses it receives. Based on these responses it can learn about the system and the SSH server. It even knows about particular vulnerabilities and can warn you about them. Download the tool via GitHub and give it a spin.

See the Linux Security Expert category SSH configuration scanners (<https://linuxsecurity.expert/security-tools/ssh-configuration-scanners>) for other alternatives.

Resources

Read the man page

A good resource for SSH configuration settings is the man page. While this sounds like an easy tip, it is actually useful to know the man page is strong and well-maintained. With all the minor differences between releases, you should never assume what a setting does. Instead, read about the setting and see if it has recent additions. Your configuration of two years ago might already be outdated. Combine this knowledge with the output of `ssh -T` and you should be able to select the right option for your situation.

References

The following references were used to create this article:

Did this article help you enhancing your SSH configuration? Great! Become part of the community and share it on your favorite website or on social media. Got questions or suggestions? Let me know in the comments.

Ayub Roti Technical Solutions Engineer, Serianu Limited.

Hardening SSH - Jason Rigden - Medium

Content Courtesy of
<https://medium.com/@jasonrigden/hardening-ssh-1bcb99cd4cef>

Check out my podcast, "Talking Cryptocurrency" where I interview with the people making the cryptocurrency and blockchain revolution happen. Guests have ranged from solo devs to CEOs. These are quick 15–20 minutes episodes. Read my Podcast Manifesto or listen to the show now.

Keep your server safe with a few extra steps.

SSH is essential to server management. This post will walk you through some of the options available to harden OpenSSH. The instructions may work for other flavors of Linux but is intended for Ubuntu 16.04 LTS.

Warning: *Messing with how SSH works can be dangerous. You can very easily lock yourself out of the server. Be careful.*

OpenSSH Server Configuration

The settings file for OpenSSH on Ubuntu 16.04 is located at `/etc/ssh/sshd_config`. You will need to be `root` or use `sudo` to edit and control the SSH server.

Backup Configuration File

It is always a good idea to make a backup of any configuration files before editing them.

```
cp /etc/ssh/sshd_config /etc/ssh/backup.sshd_config
```

Editing the Configuration File

I am not fancy so, I use `nano` for configuration file edits.

```
nano /etc/ssh/sshd_config
```

SSH Configuration Test

After editing the configuration file you should test that it is valid before reloading the service.

```
ssh -t
```

Reload the Configuration File

Once you think your edits are good, reload the SSH daemon.

```
sudo systemctl reload sshd
```

Check the Protocol

Our very first edit will be very simple. It is really more of a double check than an edit. Open `/etc/ssh/sshd_config` and check

the line that starts with `Protocol` . Make sure it is set to 2 and not 1. The current default is 2.

```
Protocol 2
```

Disable Root

Instead of using `root` , we should be using connecting as user with `sudo` permission. Make sure you have `sudo` setup properly before continuing. So let's disable the ability of root to login using SSH. Inside the configuration file find the line:

```
PermitRootLogin yes
```

Change that to no:

```
PermitRootLogin no
```

Disconnect Idle Sessions

Idle sessions can be dangerous. It is a good idea to log people out after a set amount of inactivity. The `ClientAliveInterval` is the amount of time in seconds before the server will send an alive message to the client after no data has been received. `ClientAliveCountMax` is the number of times it will check before disconnecting. In the example below, the server will check on the client after 5 minutes of inactivity. It will do this twice then disconnect.

```
ClientAliveInterval 300
ClientAliveCountMax 2
```

Whitelist Users

We can limit the users that are allowed to log in SSH. This is a whitelist. Only users in this list will be allowed. **Everyone else will be denied.** Let's say that I want to allow user `norton` to log in remotely through SSH. We will add the line:

```
AllowUsers norton
```

Don't forget to add your username to the `AllowUser` list.

Change Ports

My second least favorite way of hardening SSH is changing the default port. Normally SSH runs on port 22. The idea is that most script kiddies are only going to target that port. If you change your default port, maybe your attacks will decrease. I don't do this or recommend it. But, maybe you disagree. In the configuration file find the line:

```
Port 22
```

Then change it to another available like maybe 2222.

```
Port 2222
```

SSH Keys

By default you log into the system through SSH with a username and a password. These can be brute forced. People will try an enormous amount of username and password combinations until they find one works. So, instead of using passwords we should use SSH keys.

Generating a Key Pair

If you already have a key pair, skip ahead.

We are going to make some public key encryption keys. They come in pairs. Private and public. If you are not familiar with this system of encryption, then check out my video, A Very Brief Introduction to Public-key Cryptography.

Run the following command to generate your keys on the **client machine**. Do not run this command `sudo` . It will ask you for a passphrase to protect the key. You can keep this blank but I do not recommend that. A private SSH key with no passphrase protection can be used by anyone with possession of that key to access the server.

```
ssh-keygen
```

Share Your Public Key

Use `ssh-copy-id` to send your public key to the server.

```
ssh-copy-id jason@192.168.1.1
```

Now try logging in. You may be asked for your passphrase.

```
ssh jason@192.168.1.1
```

You should get a message back that looks similar too:

```
The authenticity of host '192.168.1.1 (192.168.1.1)' can't be established.
ECDSA key fingerprint is ff:fd:d5:f9:66:fe:73:84:e1:56:cf:d6:ff:ff.
Are you sure you want to continue connecting (yes/no)?
```

Say yes and you should be logged in without a password.

Disable Password Authentication

If we have SSH keys working we can just disable all password authentication. Find the line:

```
PasswordAuthentication yes
```

And change that to no.

```
PasswordAuthentication no
```

Disable X11 Forwarding

This guide is intended for use with remote servers. Generally speaking, there is no reason to use a GUI for a remote server. So disable X11 forwarding. Find the line:

```
X11Forwarding yes
```

and change that to no.

```
X11Forwarding no
```

Fail2Ban

This is a great program that can scan logs and ban temporarily ban IPs based on possible malicious activity. You will need to install Fail2ban.

```
apt-get install fail2ban
```

Once installed, we copy the fail2ban configuration file.

```
cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

Open the `/etc/fail2ban/jail.local` files and find the spot that starts `[sshd]`. Edit it like so, adding `enabled = true`:

```
[sshd]enabled = true
port      = ssh
logpath   = %(sshd_log)s
```

Then restart fail2ban

```
service fail2ban restart
```

Fail2ban will monitor your SSH logs for possible malicious activity and then temporarily ban the source IP.

Multi-Factor Authentication

We can also use TOTP (Time-Based One-Time Passwords) to harden our SSH security. In this example we will be using Google Authenticator (https://en.wikipedia.org/wiki/Google_Authenticator?source=post_page-----). When we attempt to log into the system we will be challenged to provide a verification code. We will use the Google Authenticator app to generate that code. First we need to install some software.

```
sudo apt-get install libpam-google-authenticator
```

Then run the initialization

```
google-authenticator
```


Change Default Ciphers and Algorithms

Continuing to follow the advice of sribika (https://sribika.github.io/2015/01/04/secure-secure-shell.html?source=post_page-----), mozilla (https://infosec.mozilla.org/guidelines/openssh?source=post_page-----), and the SSH audit report. We change our Key exchange algorithms, symmetric ciphers and, message authentication codes. Add or replace the following to the ssh configuration file.

```
KexAlgorithms curve25519-sha256@libssh.orgCiphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctrMACs hmac-sha2-512-etm@openssh.com (mailto:hmac-sha2-512-etm@openssh.com?source=post_page-----),hmac-sha2-256-etm@openssh.com (mailto:%2Chmac-sha2-256-etm@openssh.com?source=post_page-----),umac-128-etm@openssh.com (mailto:%2Cumac-128-etm@openssh.com?source=post_page-----)
```

Rerun the Audit

Let us see if our changes made the SSH audit happy.

```
python ssh-audit.py 173.255.250.98
```

Green is good.

That is looking better.

Regenerate Moduli

The `/etc/ssh/moduli` file contains prime numbers and generators used by the SSH server for the Diffie-Hellman key exchange. Your current `/etc/ssh/moduli` is probably not unique. Generating a new file may harden your server. Generating these file might take awhile.

```
ssh-keygen -G moduli-2048.candidates -b 2048
ssh-keygen -T moduli-2048 -f moduli-2048.candidates
cp moduli-2048 /etc/ssh/moduli
rm moduli-2048
```

Conclusion

Hopefully you have found this useful and your server will now be just a bit more hard to break into now. There is much more to learn about OpenSSH. Good Luck.

If you liked this post then you might like my YouTube channel, Mr. Rigden's Channel.

(https://www.youtube.com/channel/UCaznk2AsVEhIAI0u2fOLNsw?source=post_page-----)

Ayub Roti Technical Solutions Engineer, Serianu Limited.

Top 20 OpenSSH Server Best Security Practices - nixCraft

Content Courtesy of

<https://www.cyberciti.biz/tips/linux-unix-bsd-openssh-server-best-practices.html>

OpenSSH is the implementation of the SSH protocol. OpenSSH is recommended for remote login, making backups, remote file transfer via scp or sftp, and much more. SSH is perfect to keep confidentiality and integrity for data exchanged between two networks and systems. However, the main advantage is server authentication, through the use of public key cryptography. From time to time there are rumors (<https://isc.sans.edu/diary/OpenSSH+Rumors/6742>) about OpenSSH zero day exploit. This **page shows how to secure your OpenSSH server running on a Linux or Unix-like system to improve sshd security.**

Adblock detected 🙄

My website is made possible by displaying online advertisements to my visitors. I get it! Ads are annoying but they help keep this website running. It is hard to keep the site running and producing new content when so many people block ads. Please consider donating money to the nixCraft via

PayPal (https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=LJF8UGD7QKF3U)

/

Bitcoin (<https://www.cyberciti.biz/tips/donate#BTC>)

, or become a

supporter using Patreon (<https://www.patreon.com/nixcraft>)

OpenSSH defaults

- TCP port – 22
- OpenSSH server config file – sshd_config (located in /etc/ssh/)

1. Use SSH public key based login

OpenSSH server supports various authentication. It is recommended that you use public key based authentication. First, create the key pair using following ssh-keygen command on your local desktop/laptop:

DSA and RSA 1024 bit or lower ssh keys are considered weak. Avoid them. RSA keys are chosen over ECDSA keys when backward compatibility is a concern with ssh clients. All ssh keys are either ED25519 or RSA. Do not use any other type.

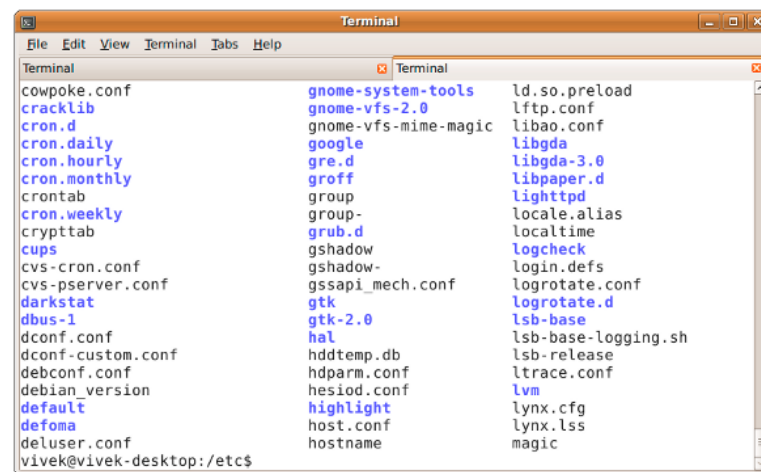
```
$ ssh-keygen -t key_type -b bits -C "comment"
$ ssh-keygen -t ed25519 -C "Login to production cluster at xyz corp"
$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa_aws_${date +%Y-%m-%d} -C "AWS key for abc corp clients"
```

Next, install the public key using ssh-copy-id command:

```
$ ssh-copy-id -i /path/to/public-key-file user@host
$ ssh-copy-id user@remote-server-ip-or-dns-name
$ ssh-copy-id vivek@rhel7-aws-server
```

When prompted supply user password. Verify that ssh key based login working for you:

```
$ ssh vivek@rhel7-aws-server
```



(<https://www.cyberciti.biz/tips/wp->

content/uploads/2009/07/OpenSSH-server-security-best-practices.png)

For more info on ssh public key auth see:

2. Disable root user login

Before we disable root user login, make sure regular user can log in as root. For example, allow vivek user to login as root using the sudo command.

How to add vivek user to sudo group on a Debian/Ubuntu

Allow members of group sudo to execute any command. Add user vivek to sudo group (<https://www.cyberciti.biz/faq/how-to-create-a-sudo-user-on-ubuntu-linux-server/>):

```
$ sudo adduser vivek sudo
```

Verify group membership with id command (<https://www.cyberciti.biz/faq/unix-linux-id-command-examples-usage-syntax/>):

```
$ id vivek
```

How to add vivek user to sudo group on a CentOS/RHEL server

Allows people in group wheel to run all commands on a CentOS/RHEL and Fedora Linux server. Use the usermod command to add the user named vivek to the wheel group:

```
$ sudo usermod -aG wheel vivek
$ id vivek
```

Test sudo access and disable root login for ssh

Test it and make sure user vivek can log in as root or run the command as root:

```
$ sudo -i
$ sudo /etc/init.d/sshd status
$ sudo systemctl status httpd
```

Once confirmed disable root login by adding the following line to sshd_config:

```
PermitRootLogin no
ChallengeResponseAuthentication no
PasswordAuthentication no
UsePAM no
```

See "How to disable ssh password login on Linux to increase security (<https://www.cyberciti.biz/faq/how-to-disable-ssh-password-login-on-linux/>)" for more info.

3. Disable password based login

All password-based logins must be disabled. Only public key based logins are allowed. Add the following in your sshd_config file:

```
AuthenticationMethods publickey
PubkeyAuthentication yes
```

Older version of SSHD on CentOS 6.x/RHEL 6.x user should use the following setting:

```
PubkeyAuthentication yes
```

4. Limit Users' ssh access

By default, all systems user can login via SSH using their password or public key. Sometimes you create UNIX / Linux user account for FTP or email purpose. However, those users can log in to the system using ssh. They will have full access to system tools including compilers and scripting languages such as Perl, Python which can open network ports and do many other fancy things. Only allow root, vivek and jerry user to use the system via SSH, add the following to sshd_config:

```
AllowUsers vivek jerry
```

Alternatively, you can allow all users to login via SSH but deny only a few users, with the following line in sshd_config:

```
DenyUsers root saroj anjali foo
```

You can also configure Linux PAM (<http://www.cyberciti.biz/tips/linux-pam-configuration-that-allows-or-deny-login-via-the-sshd-server.html>) allows or deny login via the sshd server. You can allow list of group name (<http://www.cyberciti.biz/tips/openssh-deny-or-restrict-access-to-users-and-groups.html>) to access or deny access to the ssh.

5. Disable Empty Passwords

You need to explicitly disallow remote login from accounts with empty passwords, update sshd_config with the following line:

```
PermitEmptyPasswords no
```

6. Use strong passwords and passphrase for ssh users/keys

It cannot be stressed enough how important it is to use strong user passwords and passphrase for your keys. Brute force attack works because user goes to dictionary based passwords. You can force users to avoid passwords against a dictionary (<http://www.cyberciti.biz/tips/linux-check-passwords-against-a-dictionary-attack.html>) attack and use john the ripper tool (<http://www.cyberciti.biz/faq/unix-linux-password-cracking-john-the-ripper/>) to find out existing weak passwords. Here is a sample random password generator (put in your ~/.bashrc):

```
genpasswd() {
    local l=$1
    [ "$l" == "" ] && l=20
    tr -dc A-Za-z0-9_ < /dev/urandom | head -c ${l} | xargs
}
```

```
genpasswd() { local l=$1 [ "$l" == "" ] && l=20 tr -dc A-Za-z0-9_ < /dev/urandom | head -c ${l} | xargs }
```

Run it:

```
genpasswd 16
```

Output:

```
uw8CnDVMwC6vOKgW
```

7. Firewall SSH TCP port # 22

You need to firewall ssh TCP port # 22 by updating iptables/ufw/firewall-cmd or pf firewall configurations. Usually, OpenSSH server must only accept connections from your LAN or other remote WAN sites only.

Netfilter (Iptables) Configuration

Update /etc/sysconfig/iptables (Redhat and friends specific file) to accept connection (<https://www.cyberciti.biz/faq/rhel-fedora-linux-iptables-firewall-configuration-tutorial/>) only from 192.168.1.0/24 and 202.54.1.5/29, enter:

```
-A RH-Firewall-1-INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 22 -j ACCEPT
-A RH-Firewall-1-INPUT -s 202.54.1.5/29 -m state --state NEW -p tcp --dport 22 -j ACCEPT
```

```
-A RH-Firewall-1-INPUT -s 192.168.1.0/24 -m state --state NEW -p tcp --dport 22 -j ACCEPT -A RH-Firewall-1-INPUT -s 202.54.1.5/29 -m state --state NEW -p tcp --dport 22 -j ACCEPT
```

If you've dual stacked sshd with IPv6, edit /etc/sysconfig/ip6tables (Redhat and friends specific file), enter:

```
-A RH-Firewall-1-INPUT -s ipv6network::/ipv6mask -m tcp -p tcp --dport 22 -j ACCEPT
```

```
-A RH-Firewall-1-INPUT -s ipv6network::/ipv6mask -m tcp -p tcp --dport 22 -j ACCEPT
```

Replace ipv6network::/ipv6mask with actual IPv6 ranges.

UFW for Debian/Ubuntu Linux

UFW is an acronym for uncomplicated firewall. It is used for managing a Linux firewall (<https://www.cyberciti.biz/faq/howto-configure-setup-firewall-with-ufw-on-ubuntu-linux/>) and aims to provide an easy to use interface for the user. Use the following command to accept port 22 from 202.54.1.5/29 (<https://www.cyberciti.biz/faq/ufw-allow-incoming-ssh-connections-from-a-specific-ip-address-subnet-on-ubuntu-debian/>) only:

```
$ sudo ufw allow from 202.54.1.5/29 to any port 22
```

Read "Linux: 25 Iptables Netfilter Firewall Examples For New SysAdmins (<https://www.cyberciti.biz/tips/linux-iptables-examples.html>)" for more info.

*BSD PF Firewall Configuration

If you are using PF firewall update /etc/pf.conf (<https://bash.cyberciti.biz/firewall/pf-firewall-script/>) as follows:

```
pass in on $ext_if inet proto tcp from {192.168.1.0/24, 202.54.1.5/29} to $ssh_server_ip port ssh fl
ags S/SA synproxy state
```

8. Change SSH Port and limit IP binding

By default, SSH listens to all available interfaces and IP address on the system. Limit ssh port binding and change ssh port (many brutes forcing scripts only try to connect to TCP port # 22). To bind to 192.168.1.5 and 202.54.1.5 IPs and port 300, add or correct the following line in sshd_config:

```
Port 300
ListenAddress 192.168.1.5
ListenAddress 202.54.1.5
```

Port 300 ListenAddress 192.168.1.5 ListenAddress 202.54.1.5

A better approach to use proactive approaches scripts such as fail2ban or denyhosts when you want to accept connection from dynamic WAN IP address.

9. Use TCP wrappers (optional)

TCP Wrapper is a host-based Networking ACL system, used to filter network access to the Internet. OpenSSH does support TCP wrappers. Just update your /etc/hosts.allow file as follows to allow SSH only from 192.168.1.2 and 172.16.23.12 IP address:

```
sshd : 192.168.1.2 172.16.23.12
```

See this FAQ about setting and using TCP wrappers (<http://www.cyberciti.biz/faq/tcp-wrappers-hosts-allow-deny-tutorial/>) under Linux / Mac OS X and UNIX like operating systems.

10. Thwart SSH crackers/brute force attacks

Brute force is a method of defeating a cryptographic scheme by trying a large number of possibilities (combination of users and passwords) using a single or distributed computer network. To prevent brute force attacks against SSH, use the following software:

- DenyHosts (<http://www.cyberciti.biz/faq/block-ssh-attacks-with-denyhosts/>) is a Python based security tool for SSH servers. It is intended to prevent brute force attacks on SSH servers by monitoring invalid login attempts in the authentication log and blocking the originating IP addresses.
- Explains how to setup DenyHosts (<http://www.cyberciti.biz/faq/rhel-linux-block-ssh-dictionary-brute-force-attacks/>) under RHEL / Fedora and CentOS Linux.
- Fail2ban (<https://www.fail2ban.org>) is a similar program that prevents brute force attacks against SSH.
- sshguard (<https://www.sshguard.net>) protect hosts from brute force attacks against ssh and other services using pf.
- security/sshblock (<http://www.bsdconsulting.no/tools/>) block abusive SSH login attempts.
- IPQ BDB filter (<https://savannah.nongnu.org/projects/ipqbdb/>) May be considered as a fail2ban lite.

11. Rate-limit incoming traffic at TCP port # 22 (optional)

Both netfilter and pf provides rate-limit option to perform simple throttling on incoming connections on port # 22.

Iptables Example

The following example will drop incoming connections which make more than 5 connection attempts upon port 22 within 60 seconds:

```
#!/bin/bash
inet_if=eth1
ssh_port=22
$IPT -I INPUT -p tcp --dport $ssh_port -i $inet_if -m state --state NEW -m recent --set
$IPT -I INPUT -p tcp --dport $ssh_port -i $inet_if -m state --state NEW -m recent --update --s
econds 60 --hitcount 5 -j DROP
```

```
#!/bin/bash inet_if=eth1 ssh_port=22 $IPT -I INPUT -p tcp --dport $ssh_port -i $inet_if -m state --state NEW -m recent --set
$IPT -I INPUT -p tcp --dport $ssh_port -i $inet_if -m state --state NEW -m recent --update --seconds 60 --hitcount 5 -j DROP
```

Call above script from your iptables scripts. Another config option:

```
$IPT -A INPUT -i ${inet_if} -p tcp --dport ${ssh_port} -m state --state NEW -m limit --limit 3/min
--limit-burst 3 -j ACCEPT
$IPT -A INPUT -i ${inet_if} -p tcp --dport ${ssh_port} -m state --state ESTABLISHED -j ACCEPT
$IPT -A OUTPUT -o ${inet_if} -p tcp --sport ${ssh_port} -m state --state ESTABLISHED -j ACCEPT
# another one line example
# $IPT -A INPUT -i ${inet_if} -m state --state NEW,ESTABLISHED,RELATED -p tcp --dport 22 -m limit
--limit 5/minute --limit-burst 5-j ACCEPT
```

```
$IPT -A INPUT -i ${inet_if} -p tcp --dport ${ssh_port} -m state --state NEW -m limit --limit 3/min --limit-burst 3 -j ACCEPT $IPT -A
INPUT -i ${inet_if} -p tcp --dport ${ssh_port} -m state --state ESTABLISHED -j ACCEPT $IPT -A OUTPUT -o ${inet_if} -p tcp
--sport ${ssh_port} -m state --state ESTABLISHED -j ACCEPT # another one line example # $IPT -A INPUT -i ${inet_if} -m state
--state NEW,ESTABLISHED,RELATED -p tcp --dport 22 -m limit --limit 5/minute --limit-burst 5-j ACCEPT
```

See iptables man page for more details.

*BSD PF Example

The following will limits the maximum number of connections per source to 20 and rate limit the number of connections to 15 in a 5 second span. If anyone breaks our rules add them to our abusive_ips table and block them for making any further connections. Finally, flush keyword kills all states created by the matching rule which originate from the host which exceeds these limits.

```
sshd_server_ip="202.54.1.5"
table <abusive_ips> persist
block in quick from <abusive_ips>
pass in on $ext_if proto tcp to $sshd_server_ip port ssh flags S/SA keep state (max-src-conn 20, max
-src-conn-rate 15/5, overload <abusive_ips> flush)
```

```
sshd_server_ip="202.54.1.5" table <abusive_ips> persist block in quick from <abusive_ips> pass in on $ext_if proto tcp to
$sshd_server_ip port ssh flags S/SA keep state (max-src-conn 20, max-src-conn-rate 15/5, overload <abusive_ips> flush)
```

12. Use port knocking (optional)

Port knocking (https://en.wikipedia.org/wiki/Port_knocking) is a method of externally opening ports on a firewall by generating a connection attempt on a set of prespecified closed ports. Once a correct sequence of connection attempts is received, the firewall rules are dynamically modified to allow the host which sent the connection attempts to connect to the specific port(s). A sample port Knocking example for ssh using iptables:

```
$IPT -N stage1
$IPT -A stage1 -m recent --remove --name knock
$IPT -A stage1 -p tcp --dport 3456 -m recent --set --name knock2

$IPT -N stage2
$IPT -A stage2 -m recent --remove --name knock2
$IPT -A stage2 -p tcp --dport 2345 -m recent --set --name heaven

$IPT -N door
$IPT -A door -m recent --rcheck --seconds 5 --name knock2 -j stage2
$IPT -A door -m recent --rcheck --seconds 5 --name knock -j stage1
$IPT -A door -p tcp --dport 1234 -m recent --set --name knock

$IPT -A INPUT -m --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A INPUT -p tcp --dport 22 -m recent --rcheck --seconds 5 --name heaven -j ACCEPT
$IPT -A INPUT -p tcp --syn -j door
```

```
$IPT -N stage1 $IPT -A stage1 -m recent --remove --name knock $IPT -A stage1 -p tcp --dport 3456 -m recent --set --name
knock2 $IPT -N stage2 $IPT -A stage2 -m recent --remove --name knock2 $IPT -A stage2 -p tcp --dport 2345 -m recent --set
--name heaven $IPT -N door $IPT -A door -m recent --rcheck --seconds 5 --name knock2 -j stage2 $IPT -A door -m recent
--rcheck --seconds 5 --name knock -j stage1 $IPT -A door -p tcp --dport 1234 -m recent --set --name knock $IPT -A INPUT -m
--state ESTABLISHED,RELATED -j ACCEPT $IPT -A INPUT -p tcp --dport 22 -m recent --rcheck --seconds 5 --name heaven -j
ACCEPT $IPT -A INPUT -p tcp --syn -j door
```

For more info see:

13. Configure idle log out timeout interval

A user can log in to the server via ssh, and you can set an idle timeout interval to avoid unattended ssh session. Open sshd_config and make sure following values are configured:

```
ClientAliveInterval 300
ClientAliveCountMax 0
```

You are setting an idle timeout interval in seconds (300 secs == 5 minutes). After this interval has passed, the idle user will be automatically kicked out (read as logged out). See how to automatically log BASH / TCSH / SSH users (<http://www.cyberciti.biz/faq/linux-unix-login-bash-shell-force-time-outs/>) out after a period of inactivity for more details.

14. Enable a warning banner for ssh users

Set a warning banner by updating `sshd_config` with the following line:

Banner /etc/issue

Sample /etc/issue file:

```
-----
You are accessing a XYZ Government (XYZG) Information System (IS) that is provided for authorized use only.
By using this IS (which includes any device attached to this IS), you consent to the following conditions:

+ The XYZG routinely intercepts and monitors communications on this IS for purposes including, but not limited to,
penetration testing, COMSEC monitoring, network operations and defense, personnel misconduct (PM), law enforcement (LE), and counterintelligence (CI) investigations.

+ At any time, the XYZG may inspect and seize data stored on this IS.

+ Communications using, or data stored on, this IS are not private, are subject to routine monitoring, interception, and search, and may be disclosed or used for any XYZG authorized purpose.

+ This IS includes security measures (e.g., authentication and access controls) to protect XYZG interests--not for your personal benefit or privacy.

+ Notwithstanding the above, using this IS does not constitute consent to PM, LE or CI investigative searching or monitoring of the content of privileged communications, or work product, related to personal representation or services by attorneys, psychotherapists, or clergy, and their assistants. Such communications and work product are private and confidential. See User Agreement for details.
-----
```

Above is a standard sample, consult your legal team for specific user agreement and legal notice details.

15. Disable .rhosts files (verification)

Don't read the user's `~/.rhosts` and `~/.shosts` files. Update `sshd_config` with the following settings:

IgnoreRhosts yes

SSH can emulate the behavior of the obsolete `rsh` command, just disable insecure access via `RSH`.

16. Disable host-based authentication (verification)

To disable host-based authentication, update `sshd_config` with the following option:

HostbasedAuthentication no

17. Patch OpenSSH and operating systems

It is recommended that you use tools such as `yum` (<http://www.cyberciti.biz/faq/rhel-centos-fedora-linux-yum-command-howto/>), `apt-get` (<http://www.cyberciti.biz/tips/linux-debian-package-management-cheat-sheet.html>), `freebsd-update` (<http://www.cyberciti.biz/tips/howto-keep-freebsd-system-up-to-date.html>) and others to keep systems up to date with the latest security patches:

18. Chroot OpenSSH (Lock down users to their home directories)

By default users are allowed to browse the server directories such as `/etc/`, `/bin` and so on. You can protect `ssh`, using `os` based `chroot` or use special tools such as `rssh` (<http://www.cyberciti.biz/tips/rhel-centos-linux-install-configure-rssh-shell.html>). With the release of OpenSSH 4.8p1 or 4.9p1, you no longer have to rely on third-party hacks such as `rssh` or complicated `chroot(1)` setups to lock users to their home directories. See this blog post (<https://www.debian-administration.org/articles/590>) about new `ChrootDirectory` directive to lock down users to their home directories.

19. Disable OpenSSH server on client computer

Workstations and laptop can work without OpenSSH server. If you do not provide the remote login and file transfer capabilities of SSH, disable and remove the SSHD server. CentOS / RHEL users can disable and remove `openssh-server` with the `yum` command (<http://www.cyberciti.biz/faq/rhel-centos-fedora-linux-yum-command-howto/>):

```
$ sudo yum erase openssh-server
```

Debian / Ubuntu Linux user can disable and remove the same with the `apt` command (<http://www.cyberciti.biz/faq/ubuntu-lts-debian-linux-apt-command-examples/>)/`apt-get` command (<http://www.cyberciti.biz/tips/linux-debian-package-management-cheat-sheet.html>):

```
$ sudo apt-get remove openssh-server
```

You may need to update your `iptables` script to remove `ssh` exception rule. Under CentOS / RHEL / Fedora edit the files `/etc/sysconfig/iptables` and `/etc/sysconfig/ip6tables`. Once done restart `iptables` (<http://www.cyberciti.biz/faq/howto-rhel-linux-open-port-using-iptables/>) service:

```
# service iptables restart
# service ip6tables restart
```

20. Bonus tips from Mozilla

If you are using OpenSSH version 6.7+ or newer try following (<https://wiki.mozilla.org/Security/Guidelines/OpenSSH>) settings:

```
#####[ WARNING ]#####
# Do not use any setting blindly. Read sshd_config #
# man page. You must understand cryptography to #
# tweak following settings. Otherwise use defaults #
#####

# Supported HostKey algorithms by order of preference.
HostKey /etc/ssh/ssh_host_ed25519_key
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key

# Specifies the available KEX (Key Exchange) algorithms.
KexAlgorithms curve25519-sha256@libssh.org,ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256,diffie-hellman-group-exchange-sha256

# Specifies the ciphers allowed
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr

#Specifies the available MAC (message authentication code) algorithms
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-512,hmac-sha2-256,umac-128@openssh.com

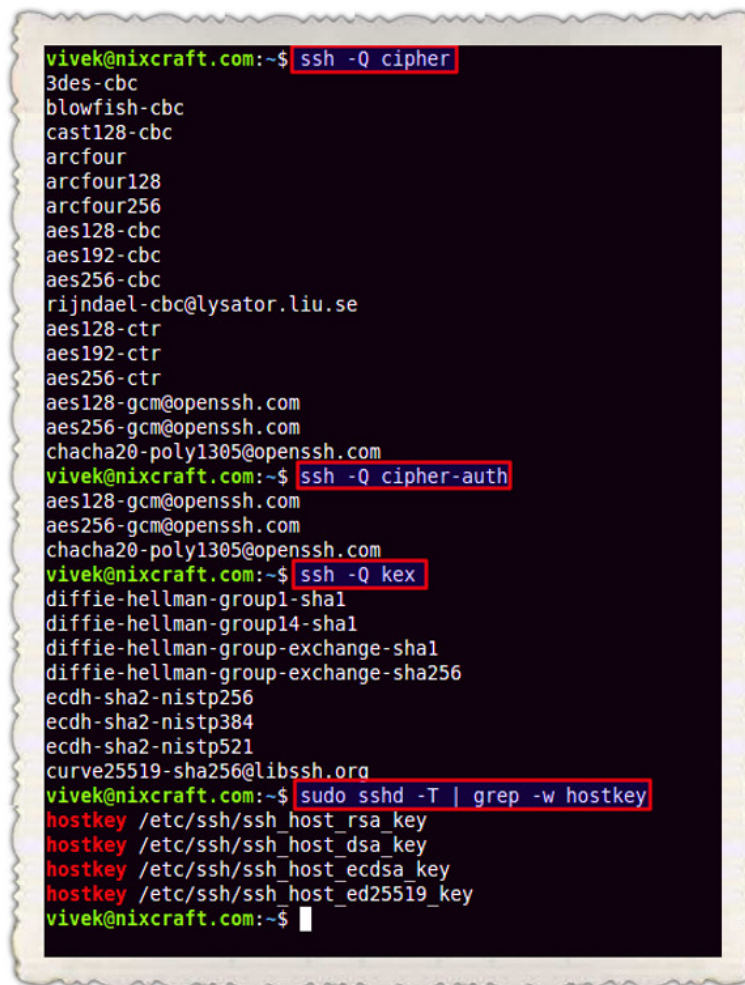
# LogLevel VERBOSE logs user's key fingerprint on login. Needed to have a clear audit track of which key was using to log in.
LogLevel VERBOSE

# Log sftp level file access (read/write/etc.) that would not be easily logged otherwise.
Subsystem sftp /usr/lib/ssh/sftp-server -f AUTHPRIV -l INFO
```

```
#####[ WARNING ]##### # Do not use any setting blindly. Read sshd_config # # man
page. You must understand cryptography to # # tweak following settings. Otherwise use defaults #
##### # Supported HostKey algorithms by order of preference.
HostKey /etc/ssh/ssh_host_ed25519_key HostKey /etc/ssh/ssh_host_rsa_key HostKey /etc/ssh/ssh_host_ecdsa_key # Specifies
the available KEX (Key Exchange) algorithms. KexAlgorithms curve25519-sha256@libssh.org,ecdh-sha2-nistp521,ecdh-sha2-
nistp384,ecdh-sha2-nistp256,diffie-hellman-group-exchange-sha256 # Specifies the ciphers allowed Ciphers chacha20-
poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr #Specifies
the available MAC (message authentication code) algorithms MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-
etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-512,hmac-sha2-256,umac-128@openssh.com # LogLevel
VERBOSE logs user's key fingerprint on login. Needed to have a clear audit track of which key was using to log in. LogLevel
VERBOSE # Log sftp level file access (read/write/etc.) that would not be easily logged otherwise. Subsystem sftp /usr/lib/ssh/sftp-
server -f AUTHPRIV -l INFO
```

You can grab list of cipher and alog supported by your OpenSSH server using the following commands:

```
$ ssh -Q cipher
$ ssh -Q cipher-auth
$ ssh -Q mac
$ ssh -Q kex
$ ssh -Q key
```



```
vivek@nixcraft.com:~$ ssh -Q cipher
3des-cbc
blowfish-cbc
cast128-cbc
arcfour
arcfour128
arcfour256
aes128-cbc
aes192-cbc
aes256-cbc
rijndael-cbc@lysator.liu.se
aes128-ctr
aes192-ctr
aes256-ctr
aes128-gcm@openssh.com
aes256-gcm@openssh.com
chacha20-poly1305@openssh.com
vivek@nixcraft.com:~$ ssh -Q cipher-auth
aes128-gcm@openssh.com
aes256-gcm@openssh.com
chacha20-poly1305@openssh.com
vivek@nixcraft.com:~$ ssh -Q key
diffie-hellman-group1-sha1
diffie-hellman-group14-sha1
diffie-hellman-group-exchange-sha1
diffie-hellman-group-exchange-sha256
ecdh-sha2-nistp256
ecdh-sha2-nistp384
ecdh-sha2-nistp521
curve25519-sha256@libssh.org
vivek@nixcraft.com:~$ sudo sshd -T | grep -w hostkey
hostkey /etc/ssh/ssh_host_rsa_key
hostkey /etc/ssh/ssh_host_dsa_key
hostkey /etc/ssh/ssh_host_ecdsa_key
hostkey /etc/ssh/ssh_host_ed25519_key
vivek@nixcraft.com:~$
```

(https://www.cyberciti.biz

/tips/wp-content/uploads/2009/07/OpenSSH-Security-Tutorial-Query-Ciphers-and-algorithms-choice.jpg)

How do I test sshd_config file and restart/reload my SSH server?

To check the validity of the configuration file and sanity of the keys (<https://www.cyberciti.biz/tips/checking-openssh-sshd-configuration-syntax-errors.html>) for any errors before restarting sshd, run:

```
$ sudo sshd -t
```

Extended test mode:

```
$ sudo sshd -T
```

Finally restart sshd on a Linux or Unix like systems (<https://www.cyberciti.biz/faq/howto-restart-ssh/>) as per your distro version:

```
$ sudo systemctl start ssh (https://www.cyberciti.biz/faq/howto-start-stop-ssh-server/) ## Debian/Ubuntu Linux##
```

```
$ sudo systemctl restart sshd.service (https://www.cyberciti.biz/faq/centos-stop-start-restart-sshd-command/) ## CentOS/RHEL/Fedora##
```

```
$ doas /etc/rc.d/sshd restart ## OpenBSD##
```

```
$ sudo service sshd restart ## FreeBSD##
```

Other suggestions

1. Tighter SSH security with 2FA (<https://www.cyberciti.biz/open-source/howto-protect-linux-ssh-login-with-google-authenticator/>) – Multi-Factor authentication can be enabled with OATH Toolkit (<https://www.nongnu.org/oath-toolkit/>) or DuoSecurity (<https://duo.com>).
2. Use keychain based authentication (<https://www.cyberciti.biz/faq/ssh-passwordless-login-with-keychain-for-scripts/>) – keychain is a special bash script designed to make key-based authentication incredibly convenient and flexible. It offers various security benefits over passphrase-free keys

See also:

If you have a technique or handy software not mentioned here, please share in the comments below to help your fellow readers keep their OpenSSH based server secure.

Posted by: Vivek Gite

The author is the creator of nixCraft and a seasoned sysadmin, DevOps engineer, and a trainer for the Linux operating system/Unix shell scripting. Get the **latest tutorials on SysAdmin, Linux/Unix and open source topics via RSS/XML feed**

(<https://www.cyberciti.biz/atom/atom.xml>) or weekly email newsletter (<https://www.cyberciti.biz/subscribe-to-weekly-linux-unix-newsletter-for-sysadmin/>).

Ayub Roti Technical Solutions Engineer, Serianu Limited.

Mitigating SSH based attacks - Top 15 Best SSH Security Practices

Content Courtesy of

<https://securitytrails.com/blog/mitigating-ssh-based-attacks-top-15-best-security-practices>

Here at SecurityTrails, we use it day by day to send and receive code securely. And while SSH (<https://www.openssh.com/>) is way more secure than the FTP protocol (as it transmits the information unencrypted in plain text format), it can still be attacked and cracked if you don't follow the best practices to harden it properly.

That's why today we will explore the most popular ways to harden your box once your Cloud provider gives you access to the new system. Let's start with the fun part: securing SSH.

15 Best SSH Hardening Tips

Follow these steps to turn your SSH server into a rock solid communication daemon.

Note: most of these tips will require you to restart the SSH service to apply the changes. You can do that by issuing:

```
service sshd restart
```

1. Set a custom SSH port

This is one of the oldest and most popular ways to obscure the SSH service. Why?

Because by default, SSH comes listening on port 22, which is widely known among attackers and security tools/port scanners that launch brute force attacks against it. While this is considered security by obscurity, it helps eliminating lots of noise on port 22.

Edit your SSH main config file (https://www.ssh.com/ssh/sshd_config/):

```
nano -w /etc/ssh/sshd_config
```

Search for: Port

Then set it to something different than 22, like:

```
Port 899
```

2. Use TCP Wrappers

This host-based ACL protection will help you to filter who can access the OpenSSH server.

TCP Wrappers (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sect-security_guide-tcp_wrappers_and_xinetd-tcp_wrappers_configuration_files) works using two files: /etc/hosts.allow and /etc/hosts.deny

First, let's deny all connections from unknown hosts:

```
nano -w /etc/hosts.deny
```

Then add this line:

```
ALL : ALL
```

If you want to allow access from your static home IP for example, you should add this code to the allow file:

```
nano -w /etc/hosts.allow
```

Then add this at the end of the file:

```
sshd : 11.22.33.44
```

Replace 11.22.33.44 with your real public IP, or with your private LAN IP, you can choose either.

This will help you to block the rest of the world, and allow only the specified IP addresses.

3. Filter the SSH port on your firewall

Using a server-side software firewall is one of the basic things that all servers should have configured after the OS is installed.

One of the most popular and effective firewalls we've found is CSF by ConfigServer (<https://configserver.com/cp/csf.html>).

Installing CSF (<https://download.configserver.com/csf/install.txt>) on plan based distros (without any control panel) is easy, and filtering the ports is even easier.

Edit the main csf.conf file:

```
nano -w /etc/csf/csf.conf
```

Then search for this variables:

```
# Allow incoming TCP ports
TCP_IN = "80,443,899"

# Allow outgoing TCP ports
TCP_OUT = "80,443"
```

In this case, we added our custom 899 SSH port to the list of TCP_IN connections at the end after configuring 80 and 443 ports.

Best security practices always suggest allowing only trusted public static IPs, or private LAN connections. Your SSH port should never be opened to external untrusted connections.

If you are using your own custom iptables rules, you can open TCP incoming connections by running:

```
iptables -A INPUT -p tcp -s 11.22.33.44 -m tcp --dport 899 -j ACCEPT
```

Make sure to replace 11.22.33.44 with your allowed IP.

4. Disable Root Login

Apart from having the port set to 22 by default, SSH servers come also with root login allowed on most Linux and Unix operating systems.

This allows anyone to connect to port 22, and use the root user as default, and then launch a ton of brute force attacks against your public server IP.

If you have password authentication enabled on your SSH service and your root password is weak, chances you are going to get hacked are pretty high. That's why disabling root login is one of the oldest and most used techniques to avoid system compromise on fresh OS installations.

You can check this out by leaving the SSH installation by default, parse the /var/log/secure file and see how many brute force attacks you will receive within a few hours- you will be surprised.

Even if you are using strong passwords, it's always recommended to disable root login and use a primary SSH user instead, so then if you need to gain root access you can do it using su command if required.

Edit SSH main config file:

```
nano -w /etc/ssh/sshd_config
```

Then search for the "PermitRootLogin" variable and set it to no:

```
PermitRootLogin no
```

If it is not present, add the AllowUsers variable to the file, and set your allowed users there:

```
AllowUsers sectrains9
```

What if you need root access from certain IPs? Is there any way to disable root access for all IP address and only allow specific ones?

Yes, it can be done by adding the root users and IPs inside the AllowUsers line, as you see below:

```
AllowUsers sectrains9 root@192.168.1.110
```

In this case, we are allowing SSH access to the system user sectrains9, and also to the root user coming from 192.168.1.110 IP address.

5. SSH Passwordless Login

Password-based logins are good if you have a strong set of characters like symbols, uppercase, lowercase, and numbers, however, they all have the risk of brute-force cracking sooner or later.

The best, in this case, is to replace the old password-based logins with key-based logins that will increase your security, but also allow you to set an immediate fast SSH login without any prompt in the middle, as it happens when the SSH password is requested.

Using SSH key-based logins will also allow you to run automated tasks that require SSH connections like rsync file synchronization across servers, specific file transfers, remote MySQL dumps, etc.

Create your SSH key using: ssh-keygen

It will prompt a few things, hit enter on all those.


```
[sectrails9@securitytrails.com ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/sectrails9/.ssh/id_rsa):
Created directory '/home/sectrails9/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/sectrails9/.ssh/id_rsa.
Your public key has been saved in /home/sectrails9/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:u72tY6os9QxWxmBF+Iqpz7Brr9RMqCuT8cjanq4HbAQ sectrails9@securitytrails.com
The key's randomart image is:
+---[RSA 2048]-----+
|  +o  |
|E+  |
|. . + |
| . . = |
|o . .o +S |
|o+ +o = . |
|+*o.oo +. |
|====..ooo. |
|*BO=+.o.oo+++. |
+---[SHA256]-----+
[sectrails9@server ~]$
```

As you see, we avoided setting the passphrase... and that's a contradiction, right? As we are setting a private key to authenticate without any passphrase, it is not as secure as you think.

What is the purpose of using a passphrase for ssh keys? Simple: encrypt your private key `/home/sectrails9/.ssh/id_rsa`. So in this case, if an attacker is able to find it, it will be useless unless he knows the passphrase.

But, passphrases for SSH keys are not the best if you need to run automated tasks. That's why most ssh keys don't have any passphrase set.

The workaround for this is an intermediate solution, like restrict SSH login using SSH keys to a particular IP address. Let's try it.

In order to restrict the user sectrails9 to access the SSH server remotely using ssh-keys from a single static IP, you can use the "from" variable inside the `authorized_keys` file from the remote system, as you see below:

```
nano -w /home/remoteuser/.ssh/authorized_keys
```

Then set this at the beginning of the file, before your private key:

```
from="192.168.1.105"
```

Replace `192.168.1.105` with your own IP.

It should look like this:

```
from="192.168.1.105" ssh-rsa AAAAB3NzaClyc2EAAAA...
```

6. Strong passwords/passphrase for ssh users and keys

If you are not able to set an ssh-key and still need to use login passwords, or, if you decide to use a passphrase on your keys, then make sure to use strong character combination.

Try to include:

- Uppercase & lowercase letters
- Symbols.
- Numbers.
- Up to 8 characters if possible.

Avoid using:

- Dictionary-based words.
- Personal birthday and anniversary dates.
- Family and pet names.

Following this tips will help you to set a strong password/passphrase and avoid getting hacked when a brute force attack is happening.

7. Set Idle Timeout Interval

Idle timeout value allows terminating ssh sessions that are not actively used.

This variable can be edited by altering the `ClientAliveInterval` value.

```
nano -w /etc/ssh/sshd_config
```

Set:

```
ClientAliveInterval 240
```

In this case, we set 240 which equals to 4 minutes, once the timeout has been reached, the SSH session will be logged out automatically.

8. Disable Empty Passwords

On Linux and Unix, the system allows administrators to create users with empty passwords.

And this can be a pretty bad thing if you want to keep attackers out of your SSH servers.

That's why the best thing you can do is to disable remote logins for accounts with an empty password, this can be easily done by editing the `sshd_config` file.

```
nano -w /etc/ssh/sshd_config
```

Then set:

```
PermitEmptyPasswords no
```

9. Set a custom SSH warning banner

A good practice for all Linux and Unix boxes is to set a custom welcome banner for SSH connections.

This is really not a security hardening tip, but just a security warning for any illegal access to your systems.

Once the user has gained access, a warning banner will be displayed, as you see below:

```
ALERT! You are entering a secured area! Your IP and login information
have been recorded. System administration has been notified.

This system is restricted to authorized access only. All activities on
this system are recorded and logged. Unauthorized access will be fully
investigated and reported to the appropriate law enforcement agencies.
```

By altering the `/etc/motd` file you can set a custom ASCII art banner (<http://patorjk.com/software/taag/#p=display&f=Modular&t=SecurityTrails>), like the one you see below:

On most Linux distros, just editing `your/etc/motd` file is enough.

If that does not work, you can also uncomment this line inside the `sshd_config` file:

```
#Banner none
```

And set the path to something else:

```
Banner /etc/custombanner
```

10. Block SSH brute force attacks automatically

On this post, we mentioned ways to mitigate brute force attacks several times. However, the best way to actually stop a brute force attack once detected is to block the offending IPs.

How can you detect this kind of attacks against SSH service automatically?

The manual way is by parsing system logs and check who is trying to connect to the server, and then block it using the system firewall. However, there are several tools that can do those manual tasks for you in an efficient and automated way.

There are a lot of tools to prevent brute force attacks, like SSHGuard (<https://www.sshguard.net/>), Fail2ban (https://www.fail2ban.org/wiki/index.php/Main_Page), or DenyHosts (<http://denyhosts.sourceforge.net/>).

However, if you are using CSF Firewall as we previously recommended, you already have an integrated brute force detection system called LFD.

Edit `csf.conf` file as you see below:

```
nano -w /etc/csf/csf.conf
```

Then search for this variables:

```
# [*]Enable login failure detection of sshd connections
LF_SSHD = "5"
LF_SSHD_PERM = "1"

LF_SSHD set to "5" is the number of max failed connections before blocking the IP address.

LF_SSHD_PERM set to "1" makes this blocking permanent.
```

On the other side, you can increase the brute force protection by using real time blocklists, which is a cool feature already included in CSF firewall.

All the lists can be located inside this file: `/etc/csf/csf.blocklists`, there you will find Dshield (<https://www.dshield.org/>), Blocklist.de (<http://www.blocklist.de/en/index.html>), among other block lists.

In this case we will use Dshield and Blocklist.de, you can do the same by just remove the comment from the following lines:

```
#DSHIELD|86400|0|http://www.dshield.org/block.txt

#BDEALI|86400|0|http://lists.blocklist.de/lists/all.txt
```

Simply remove the first character # at the beginning of the line, so it looks like this:

```
DSHIELD|86400|0|http://www.dshield.org/block.txt
BDEALL|86400|0| [http://lists.blocklist.de/lists/all.txt] [17]
```

Save the file and restart CSF + LFD:

```
csf -r
systemctl restart lfd
```

That's it, now you are protected against brute force attacks.

11. Disable OpenSSH server on the laptops and desktops

Some Linux distros come with OpenSSH Server enabled by default (and you know what that means: root login enabled, service listening on port 22, etc), and while on Dedicated, VPS and Cloud servers, SSH access is a must in order to work remotely. On home-based computers like laptops or PCs it is not really needed.

Make sure to remove OpenSSH server to avoid unnecessary attacks.

For RHEL based distros:

```
dnf remove openssh-server
```

Debian / Ubuntu based distros can do the same by using:

```
sudo apt-get remove openssh-server
```

12. Disable X11 forwarding

If you are running a remote server, having X11 (graphics server) forwarding capabilities doesn't have too much sense, as you will always stay stick to your black and white remote terminal.

In order to disable X11 forwarding, follow this steps:

```
nano -w /etc/ssh/sshd_config
```

Look for this variable:

```
X11Forwarding yes
```

Change it to be:

```
X11Forwarding no
```

Disabling the X11 protocol will help you to prevent a few types of attacks, as it was never built with security in mind, and it can be used by malicious attackers to open up a channel to the client and send remote commands that may end up really bad.

13. Limit max authentication attempts

Another good way to protect against brute force attacks is to set a low limit for the times an attacker can try to login with a failed password. MaxAuthTries variable can help you to mitigate this kind of attacks.

```
nano -w /etc/ssh/sshd_config
```

Search for `MaxAuthTries`.

Set it to 3, as you see below:

```
MaxAuthTries 3
```

14. Enable login notifications over email

One of the best things you can do in order to keep a close eye on your incoming SSH connections is to set up a quick script to send an alert once someone is logged in as root via SSH.

```
nano -w /root/.bashrc
```

Paste this at the end of the file:

```
echo 'ALERT - Root Shell Access (ServerName) on:' `date` `who` | mail -s "Alert: Root Access from `who | cut -d'(' -f2 | cut -d')' -f1`" your@email.com
```

Replace `your@email.com` with your real email address.

In order to have this working as expected, make sure to have "mailx" package installed and working. If you don't have it, you can install it by issuing this commands:

For RHEL based distros:

```
dnf install mailx
```

Debian/Ubuntu based distros:

```
apt-get install mailx
```

15. Keep SSH updated

The basic rule that will be repeated over and over: keep your server packages updated, and that also includes OpenSSH.

For RHEL based distros, you can use DNF or yum:

```
dnf update openssh*
yum update openssh*
```

For Debian/Ubuntu based distros:

```
apt-get update openssh-server
```

This should be enough to keep your OpenSSH server patched against new vulnerabilities.

Following these SSH security tips will help you to prevent the majority of the most popular SSH attacks on your Unix/Linux operating system. Do you have any extra SSH security tips that we are missing? Get in touch (/corp/contact) and share your knowledge with us.

Now that you have your ssh service secured, it's time to jump and start auditing and hardening other popular internet services like the DNS system: follow our Best Tips to Prevent DNS Attacks (/blog/8-tips-to-prevent-dns-attacks) to discover ways to harden your domain name system services.

And if you want to dig manually into how much information you are exposing from your domain names and IPs, check out our SecurityTrails (/) toolkit, as well as our free API access (/corp/pricing) to integrate our cool cybersecurity features into your own apps.

Ayub Roti Technical Solutions Engineer, Serianu Limited.

5 Linux SSH Security Best Practices To Secure Your Systems

Content Courtesy of

<https://phoenixnap.com/blog/linux-ssh-security>

Minimizing vulnerabilities in your Secure Shell (SSH) protocol is key to ensuring the security of your Linux environment.

In this article, I will talk about the most common **Linux SSH security** measures you can take to make your servers more secure. By changing the default SSH port, using key pairs, and following the other recommended best practices, you can significantly improve the overall safety of your system.

What is SSH?

The Secure Shell (SSH) protocol enables cryptographically protected remote system administration (<https://phoenixnap.com/kb/ssh-to-connect-to-remote-server-linux-or-windows>) and file transfers over insecure networks. Using multiple encryption methods, SSH secures the connection between a client and a server safeguarding the users' commands, authentication, and output against attacks.

SSH Protocol is now widely used in data centers and almost every major enterprise running on any of the UNIX variants.

When it comes to security measures, it is essential to combine them, apply them in layers, and not just pick one and stick with it.

1. Change the default SSH port.

Using a non-standard port for SSH connection aids in avoiding automated attacks on your server. It also helps reduce the chances for it to appear on the hackers' radar.

The majority of hackers who are looking for OpenSSH servers will aim at the default SSH port 22.

In that case, the scripts they are using will look for IP addresses on port 22. If your server falls into that group, every such automated attack will create an impact on your log files. Consequently, the load on your server may increase substantially since many SSH server (<https://phoenixnap.com/kb/ssh-to-connect-to-remote-server-linux-or-windows>) exploits are running around the clock knocking on every server's door that fits the criteria.

It is important to note that the default SSH port does not improve the security of your server. However, it helps in keeping away the automated attacks.

How to Change the Port number

Before you begin, you need to decide which port you will use instead of the default port 22. Before you make a decision, you should have a few things in mind:

- Make sure the port you chose is not in conflict with another application. If you select a port that is reserved for another service, you can run into issues. To make a decision, refer to the list of TCP and UDP port numbers (https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers) assigned by the Internet Assigned Numbers Authority (IANA).
- Only root users can listen on ports below 1024; these are well-known ports.
- Avoid the most common variations of the port 22, such as 222, 2222 and 22222.
- Verify that the port you decide to use is not blocked.

To change the port on your Linux server, follow these steps:

1. Connect to the server via SSH as you normally would.
2. Switch to the root user. One of the ways is using the `su` command, which will prompt you to enter the server password.
3. Use a text editor of your choice to edit the `sshd` configuration file located in the `/etc/ssh/` directory. If you have never used a text editor within the terminal, it is recommended to use Nano. Otherwise, use `vi` or `vim` since they are most commonly used today. We advise you to back up the original file before you make the changes.
4. Run this command to edit the configuration file:
5. `nano /etc/ssh/sshd_config` In the output of the `sshd_config` file locate the line which says Port 22.
6. Change the port number to the value of your choice. Make sure there is no `#` at the beginning.
7. Exit the editor and confirm that you want to save the changes.
8. For changes to take effect, restart the `sshd` service with this command: `service sshd restart`
9. Verify that SSH is listening on the port you specified by connecting to it

Note that now you need to specify the port when connecting since your client will always use the default SSH port unless told otherwise.

Benefits

While the procedure of changing the default SSH port does not increase the level of security itself, it takes you off the radar of the most random scans. One easy way to test this is to let your server run for a few days with `sshd` listening on the default port and then change to a non-standard one. Compare the number of failed logins on your server, and you will see it will decrease substantially. By using a non-standard port for SSH:

1.
 1.
 - You avoid being seen by random scans.
 - It is more difficult to find your server. Most of the attacks will scan the default port or some variants of it, but move on once the connection is refused.
 - SSH daemon can take a break since it will not get connection requests from scripted attacks. The server load is reduced, and the log file stays clean saving you time in reviewing it.
 - You do not receive as many alerts for the failed logins. If you are using a non-standard port and someone still tries to access your server, it probably means that your server specifically is the target, so the alarm does not come from a scripted attack.
 - You are less exposed to being hacked due to the bugs in `sshd` or weak private keys.
 - Most hackers will be repelled if they see that you are not using the default port. It will be a sign that the server is properly protected and that there are probably other security measures as well, making your server an undesirable target.

Drawbacks

There are some precautions to have in mind before you decide to change the default port for SSH. The disadvantages of running a non-standard port can mean that:

1.
 1.
 - Anybody who should be able to connect to your server will need to be informed of the change and start using the new port.
 - If you are using outsourced monitoring for your server, you also need to make them aware of the change. Otherwise, they may treat this as a potential threat which may lead to some downtime for your server.
 - The firewall rules related to SSH service have to be inspected and modified according to the changes you make.

Some of these disadvantages probably do not apply to your use case, but should be taken into consideration. The benefits of changing the port outweigh the drawbacks and prove to be a good additional layer of security of your server.

2. Enhance Linux SSH security using key pairs.

One of the most secure methods to authenticate clients to servers is by using **SSH key pairs**. Strong passwords may be sufficient to keep your server safe, but persistent brute force attacks can still crack them. This is why you need additional SSH hardening with key pairs.

SSH keys are resilient to such attacks and virtually impossible to decrypt. An SSH key pair consists of two long series of characters: a **private key** which is kept secret, and a **public key** which can be safely shared. Their purpose is similar to passwords, and they allow you to **automatically establish an SSH session** without the need to type in a password.

How to Generate a Key Pair

To set up SSH keys, you will need to generate a key pair on the client computer which will be used to connect to the server. To do so:

1.
 1.
 1. Start the terminal and run the SSH keygen utility, available with the standard OpenSSH tool. `ssh-keygen -t rsa`
 2. You will get the message *'Generating public/private RSA key pair.'* If you want to save the key to the default location, press Enter when you get the prompt. The key will be saved in the home user's directory, in the `~/.ssh` directory. To change the location, just type in the new path. The recommendation is to stick with the default location, so you do not have to make any changes to your SSH client. The private, or the identification key, will be saved as `id_rsa` and the corresponding public key as `id_rsa.pub`.

3. Optionally, you can insert the passphrase. If you do not wish to use it, press Enter to continue. The passphrase provides an additional layer of security by encrypting the private key on the local machine. To crack the passphrase, a hacker will need to have access to the system first, since the private key is not exposed on the network. Even then, it will take time to succeed, allowing you to change the used key before the hacker gains access to other servers. The downside is that you will have to enter it every time you try to connect using that key.

The process of generating a key pair is complete.

The final screen will look similar to this:

```
ssh-keygen -t rsa

Generating public/private rsa key pair.

Enter file in which to save the key (/home/demo/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/demo/.ssh/id_rsa.

Your public key has been saved in /home/demo/.ssh/id_rsa.pub.

The key fingerprint is:

8b:cd:0b:f7:38:4a:3f:ed:24:18:8d:54:34:2c:63:56 your_username@host

The key's randomart image is:

+--[ RSA 2048]-----+
|           ..o.  |
|           . E.o  |
|           + . o  |
|           . = = .  |
|           ..S   |
|           = + = +  |
|           . o + o .  |
|           . + + o  |
|           ..   |
|                   |
+-----+

```

Ayub Roti Technical Solutions Engineer, Serianu Limited.

Advanced OpenSSH Hardening Guide for Improved security.

Content Courtesy of

<https://www.debyum.com/ssh-hardening-guide/>

OpenSSH is also known as **OpenBSD** Secure Shell, released under simplified **BSD License**, is used to provide secure network communications via encrypting network traffic.

Encrypting all traffic provides extra security from network sniffing and other attacks and it is available for multiple platforms.

We can also implement **SSH port forwarding** to provide protection to data which would otherwise be transferred insecurely.

For more details, please read **More About OpenSSH.** (<https://en.wikipedia.org/wiki/OpenSSH>)

Here are some other similar posts on security harden your Server and Applications which you should check.

The **OpenSSH** suite consists of many tools:

- **ssh**, **scp** and **sftp** for authentication and transferring data securely.
- **ssh-add**, **ssh-keyscan** and **ssh-keygen** for creating and managing keys.
- **sshd**, **ssh-agent** and **sftp-server** for managing services.

We use **OpenSSH** tools almost daily for various tasks, from logging in to the remote server to transferring data between servers. SSH server is widely used for server authentication. It's very secure but we can still do some tweaks to make it more secure. Here are a few things we can tweak in order to make **OpenSSH** server more secure.

Installation of OpenSSH in Linux.

To install **OpenSSH**, run these commands with superuser permissions in the terminal.

For **Ubuntu/Debian**, use:

```
shell> apt-get install openssh-server openssh-client -y
```

For **RHEL/Centos/Fedora**, use:

Type the following yum command to install **OpenSSH client** and **server**.

```
shell> yum install openssh-server openssh-clients -y
```

Configuration File of SSH.

The SSH server configuration files are located in **/etc/ssh/** directory. we can edit **/etc/ssh/sshd_config** file to make it more secure.

Before making any changes to the **sshd_config** file we will take a backup of this file.

```
shell> cp /etc/ssh/sshd_config /etc/ssh/sshd_config.$(date +%Y-%m-%d)
```

now we can start editing **SSH configuration file**.

Use only SSH protocol 2.

One of the most important things to do is use the secure version of SSH protocol > **SSHv2**.

SSHv2 is an improved version of **SSHv1**. **SSHv2** has a number of new features like **Encryption ciphers** and support for **public key certificates** to provide more security.

To use SSHv2 we can edit **/etc/ssh/sshd_config** and change

```
# The default requires explicit activation of protocol 1
Protocol 2,1
```

to

```
# The default requires explicit activation of protocol 1
Protocol 2
```

Change SSH Port.

By default, SSH listens for connections on **Port 22** which is the first port **Attackers** use to attack. We will change it to some random Private Port number from **49152** through **65535**.

Edit the **/etc/ssh/sshd_config** file and change this line:

Port 22

to

Port 50150

Now we will configure Firewall to allow this port.

for **Iptables**:

```
shell> iptables -A INPUT -p tcp -m tcp --dport 50150 -j ACCEPT
shell> /sbin/service iptables save
```

for **Firewalld**.

```
shell> firewall-cmd --zone=public --add-port=50150 --permanent
shell> firewall-cmd --reload
```

If you're using **CentOS** or **Red hat** and you have **SELinux** in enforcing mode then you also need to configure **SELinux** to allow new **SSH Port**.

```
shell> semanage port -a -t ssh_port_t -p tcp 50150
```

(If your server has only **512MB RAM** then you need to Create at least **1GB** swap space to finish this operation.)

Now restart the SSH service:

```
shell> /etc/init.d/ssh restart
```

or

```
shell> systemctl restart sshd.service
```

Try to do ssh access using new **port 50150**:

```
shell> ssh -p 50150 [email protected] (/cdn-cgi/l/email-protection)
```

Create a custom SSH banner.

You can create a banner to convey a specific message to your users. To do this create a text file and put your message into it.

This is my favorite one which I found on the Internet. Simple and effective.

```
shell> vi /etc/ssh/banner.text

*****
*The use of this SSH service is restricted to authorized user's only.*
*All the acitvities on this SSH server will be logged.*
*****
```

Save this file and then edit **sshd_config**.

```
shell> vi /etc/ssh/sshd_config
# no default banner path
#Banner none
Banner /etc/ssh/banner.txt
```

Restrict the root account to console access only.

There's a big debate on whether you should allow **root logins** via **SSH** or not. If you don't want **root logins** via **SSH** then you need to change this:

```
shell> vi /etc/ssh/sshd_config
PermitRootLogin no
```

Make sure you have another user that can switch to superuser. (make sure user is a part of **wheel** group)

Allow only specific users via SSH.

By default, all users can log in using **SSH**. But if you want to allow the only specific user's to **login** via **SSH** then you can use **AllowUsers** and **DenyUsers** to configure that.

If you want only "**mohan**" and "**john**" to log in to the system via SSH then use:

```
AllowUsers root mohan john
```


Similarly, if you don't want a user, say "**Jingping**" to access system through SSH then use:

```
DenyUsers jingping
```

Prevent Unattended SSH Sessions.

set up an **idle timeout interval** in **seconds** to avoid unattended SSH sessions.

```
shell> vi /etc/ssh/sshd_config
ClientAliveInterval 600      # 10 minutes
ClientAliveCountMax 0
```

After **10 mins** of inactivity, the user will be **logged out** automatically.

Disable user's .rhosts files.

Since we are using **SSH** we will disable **.rhosts** authentication which will prevent users from using the **obsolete rsh commands** such as **rcp** and **rsh** on the local system without a password.

```
shell> vi /etc/ssh/sshd_config
IgnoreRhosts yes
```

Disable Host-Based Authentication.

Host-Based authentication is a security risk and is **disabled by default**. If it's not disabled by default then you can disable it by:

```
shell> vi /etc/ssh/sshd_config
HostbasedAuthentication no
```

Set a login grace timeout

We will set "**LoginGraceTime**" to specify how long after a connection request the SSH server will wait before disconnecting. By default, it is set to **2 minutes** in SSH config file but we will change it to **60 seconds**.

we can change this value by editing following line:

```
LoginGraceTime 60
```

Configure SSH logging level.

By default, **SSH logs everything**. Depending upon the amount of information you want, you can change the value from **INFO** to **VERBOSE**.

we can change this value by editing following line:

```
shell> vi /etc/ssh/sshd_config
LogLevel VERBOSE
```

Disable Empty Passwords.

For security reasons, we will **deny login** from accounts with **empty (blank) passwords**. By default this option is **disabled**, if not then we can change this value by editing following line:

```
PermitEmptyPasswords no
```

Secure SSH using TCP wrappers.

TCP wrappers are useful in providing **host-based access control system** which can be used to limit network access to the

internet.

we can use **/etc/hosts.allow** or **/etc/hosts.deny** to allow or deny SSH from specific IP address(s).

To allow SSH only from specific IP address for e.g **192.168.231.1**, **172.16.30.13** we can edit **/etc/hosts.allow** file as following.

```
shell> vi /etc/hosts.allow
sshd : 192.168.231.1 172.16.30.13
```

To deny access to the bad guy we can use.

```
shell> vi /etc/hosts.deny
ALL: 192.168.231.2
```

Restrict SSH logins to specific IP address(s).

If you have more than one interface on your machine and you want **SSH server** to **listen only** on a specific Interface or **IP Address** then you can easily do this by editing the **sshd_config** file.

You can also configure SSH server to listen on **multiple IP addresses** of the machine. We can force **SSH server** to listen only on a specific Ip Address of our machine, we can do this by editing the following line.

To make SSH server listen only on Ip address **192.168.111.22**,

```
ListenAddress 192.168.111.22
```

To verify this use,

```
[[email protected] (/cdn-cgi/l/email-protection) ~]# ss -plant | grep ssh
LISTEN      0      128      192.168.111.22:50150      *:*      users: (("ss
hd",pid=28189,fd=3))
ESTAB       0      36      192.168.111.22:50150      112.211.73.45:62857      user
s: (("sshd",pid=28062,fd=3))
```

If you want to add another Ip address **192.168.111.33** in this list then just add another line like.

```
ListenAddress 192.168.111.22
ListenAddress 192.168.111.33
```

Secure SSH with the help of Firewall.

Before we make any changes in rules of **firewall**, You may like to save your rules;

```
shell> sudo iptables-save > firewall-old-rules.txt.
```

Now we can configure **Firewall** to secure **SSH server** by allowing only **Allowed Ip address** and Limit the number of attempts that can be made in a given time frame for **SSH connection** on a port.

If you want to **allow** connection only from a specific **IP ADDRESS** like e.g **192.168.231.21**, then we can use;

```
shell> iptables -A INPUT -p tcp -m state --state NEW --source 192.168.231.21 --dport 50150 -j ACCEPT
```

If you want to **allow connection** only from a specific subnet like e.g **192.168.231.0/25** which allows about 62 IP addresses then we can use ;

```
shell> iptables -A INPUT -s 192.168.231.0/25 -m state --state NEW -p tcp --dport 50150 -j ACCEPT
```

where **SSH Port no** is **50150** and **subnet** we allow is **192.168.231.0/25**

we can also configure **Firewall** to limit the number of incoming connections for SSH port within a **time** frame.

Here we will configure the **firewall** to allow only 3 connection attempts on **SSH port 50150** within **30 seconds**. This will **drop all** the incoming connections which will try to make more than 3 connection attempts on **SSH port 50150** within 30 seconds.

Rate-limit the incoming SSH port connections.

We can configure **firewall's rate-limit** option to control the incoming connections on SSH port.

```
shell> iptables -A INPUT -i eth0 -p tcp --dport 50150 -m state --state NEW -m limit --limit 3/min
--limit-burst 3 -j ACCEPT
shell> iptables -A INPUT -i eth0 -p tcp --dport 50150 -m state --state ESTABLISHED -j ACCEPT
shell> iptables -A OUTPUT -o eth0 -p tcp --sport 50150 -m state --state ESTABLISHED -j ACCEPT
```

we can also configure iptables to allow only **three connection** attempts on **port 50150** within **30 seconds**:

```
shell> iptables -I INPUT -p tcp --dport 50150 -i eth0 -m state --state NEW -m recent --set
shell> iptables -I INPUT -p tcp --dport 50150 -i eth0 -m state --state NEW -m recent --update --sec
onds 30 --hitcount 3 -j DR
```

To implement the same with **Firewalld** you need to follow these steps.

Setup a **xt_recent** configuration file.

we will create a file **/etc/modprobe.d/xt_recent.conf**.

```
shell> vi /etc/modprobe.d/xt_recent.conf
options xt_recent ip_pkt_list_tot=30
```

Now we will reinstall the **xt_recent** module.

```
shell> sudo modprobe -r xt_recent
shell> sudo modprobe xt_recent
```

*(you need to repeat these steps everytime you want to change the **xt_recent** config.)*

Now we can add these rules to rate-limit the incoming connection in **firewall-cmd**.

```
shell> firewall-cmd --permanent --direct --add-rule ipv4 filter Rate_limit 0 -p tcp --dport 50150 -m
state --state NEW -m recent --set
success
shell> firewall-cmd --permanent --direct --add-rule ipv4 filter Rate_limit 1 -p tcp --dport 50150 -
m state --state NEW -m recent --update --seconds 30 --hitcount 3 -j REJECT --reject-with tcp-reset
success
shell> firewall-cmd --reload
```

Hide the last login.

You can also hide last login user by editing the following line.

```
shell> vi /etc/ssh/sshd_config
PrintLastLog no
```

Configure Port forwarding.

If you don't need **Port Forwarding** then it's better to disable it.

we can disable this by setting these values:

```
AllowTcpForwarding no
X11Forwarding no
```

Setting up maximum startup connections.

we can configure **sshd_config** file to setup a **limit** on **concurrent connections** to the **SSH** Daemon.

This is very useful to protect **SSH server** from **Brute-Force Attack**.

We will allow here, for this example, only 2 concurrent connections.

```
MaxStartups 2
```

Log in with SSH keys instead of a password.

Using **Password** for SSH authentication is not recommended and is insecure. A weak password can compromise the security of your server. Alternative is to use **SSH keys** for authentication.

Here's how we can create and use **SSH keys** for authentication. (*we will use the simplest and fastest way*)

Generate SSH keys

we will generate **SSH keys** on your client machine by running the following command:

```
[[email protected] (/cdn-cgi/l/email-protection) ~]$ ssh-keygen -b 2048 -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
f0:12:1f:0b:17:28:1c:e4:bc:f2:c5:54:a7:36:b9:53 [email protected] (/cdn-cgi/l/email-protection)
The key's randomart image is:
+--[ RSA 2048 ]-----+
|
|      .
|     . +
|    . . S
|   + o + . * . + .
|  . * . + . * . *
| . . . + E + . o
|   o o o   . .
+-----+
```

Copy the public key file to your remote server.

Before copying the Public key file we will first create the SSH folder on our remote server.

```
[[email protected] (/cdn-cgi/l/email-protection) ~]$ mkdir -p ~/.ssh/
```

Now we will copy the public key from host to the `authorized_keys` file on the destination Server e.g (/cdn-cgi/l/email-protection)

```
shell> scp -p /root/.ssh/id_rsa.pub [email protected] (/cdn-cgi/l/email-protection):/home/debyum/.ssh/authorized_keys
```

On server:

Now we will secure these keys by using these steps.

```
[[email protected] (/cdn-cgi/l/email-protection) ~]$ chmod 700 .ssh
[[email protected] (/cdn-cgi/l/email-protection) ~]$ chmod 600 .ssh/authorized_keys
```

Now you can log into your **SSH server** without a password.

```
shell> ssh -p 50150 [email protected] (/cdn-cgi/l/email-protection)
```

Keep OpenSSH server up-to-date.

For security purposes, it's very important to keep your server up-to-date with latest security patches.

Conclusion

There is other software which you can use to help secure your **OpenSSH** server like `fail2ban` (https://www.fail2ban.org/wiki/index.php/Main_Page) which once configured, automatically scans log files and bans IPs that show the malicious signs.

I have tried to cover all the basic to advance concepts with their examples. Still, if I have missed anything please update us

through comment box to help your fellow readers. I will keep updating the same based on feedback's received. Thanks.

Thanks for visiting this Page and have a Great day. 😊

Ayub Roti Technical Solutions Engineer, Serianu Limited.

Use Advanced OpenSSH Features to Harden Access to Your Linode

Content Courtesy of

<https://www.linode.com/docs/security/advanced-ssh-server-security/>

Use Advanced OpenSSH Features to Harden Access to Your Linode

Updated Friday, June 1, 2018 by Linode Contributed by Damaso Sanoja

Use promo code **DOCS10** for \$10 credit on a new account.

There's a good chance you've been using SSH (Secure Shell) to access your Linode from your computer. Although SSH is a secure protocol, most system compromises are a result of human error or failure to take advantage of the security features offered. In this guide, we'll cover a few key features provided by OpenSSH.

Use Advanced OpenSSH Features to Harden Access to Your Linode

OpenSSH (<http://www.openssh.com/>) is a suite of connectivity tools that sysadmins use daily to access remote servers. From a security point of view, it's the front door for remote logins so it is extremely important to harden SSH as much as possible. The aim of this guide is to build upon our [Securing Your Server](/docs/security/securing-your-server/) ([docs/security/securing-your-server/](/docs/security/securing-your-server/)) guide with easy steps that can be implemented in just a few minutes.

Assumptions:

- You'll be deploying a server open to the internet 24/7.
- Your primary concern is security over simplicity or convenience. Be aware that in most cases, more security implies less convenience.

Before You Begin

- Complete the [Getting Started](/docs/getting-started/) ([docs/getting-started](/docs/getting-started/)) guide.
- Follow the [Securing Your Server](/docs/security/securing-your-server/) ([docs/security/securing-your-server/](/docs/security/securing-your-server/)) guide to create a standard user account, harden SSH access, create a basic firewall rule set and remove unnecessary network services.
- Log in to your Linode via SSH and check for updates using the corresponding package manager: `apt` (Ubuntu/Debian) or `yum` (RHEL/CentOS).

```
sudo apt update && sudo apt upgrade
sudo yum update
```

- Make a backup of your server's `sshd_config` file.

```
sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.BACKUP
```

Note

This guide is written for a non-root user. Commands that require elevated privileges are prefixed with

```
sudo
```

. If you're not familiar with the

```
sudo
```

command, see our

[Users and Groups](/docs/tools-reference/linux-users-and-groups/) ([docs/tools-reference/linux-users-and-groups](/docs/tools-reference/linux-users-and-groups/)) guide.

Use a Stronger Diffie-Hellman Algorithm

Diffie-Hellman (https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange) is the name of an asymmetric algorithm used to securely exchange cryptographic keys over public channels (like the Internet). OpenSSH uses the D-H algorithm to generate keys from a configuration file located at `/etc/ssh/moduli`. Several IT news outlets have reported the

possibility that nations may have broken commonly used primes, which mean they could read encrypted traffic transmitted over “secure” channels.

Protecting your server against that threat is fairly simple, just edit the `/etc/ssh/moduli` file and comment the lines where the fifth field is 1023 or 1535 (which denotes the size). This forces the algorithm to use keys from Group 14 (2048-bit or more). Higher groups mean more secure keys that are less likely to be broken in near future, but also require additional time to compute.

Enforcing a stronger Diffie-Hellman algorithm

Optional: Generate Your Own Keys

The `/etc/ssh/moduli` file ships with OpenSSH, so assuming two servers have the same version of OpenSSH, their default `moduli` files will be identical. This does not mean they are insecure; because Diffie-Hellman is an asymmetric algorithm, the keys it generates are incredibly difficult to crack when generated from sufficiently large primes. That being said, if you want to take the extra time to do so, you may generate new Diffie-Hellman parameters yourself:

```
ssh-keygen -G "${HOME}/moduli" -b 2048
sudo ssh-keygen -T /etc/ssh/moduli -f "${HOME}/moduli"
rm "${HOME}/moduli"
```

Caution

Before running these commands on a production server, be aware that depending on the size of the keys you’re generating, this will use significant CPU power and may take anywhere from a minute to several hours.

This sequence of commands generates a new file containing thousands of candidate primes for the Diffie-Hellman algorithm. Next, it tests the candidates and adds suitable primes to your `moduli` file. Note that these keys append to your existing ones; they do not overwrite the file, so it is still possible that your SSH connection will use a precomputed prime in its key exchange. As stated above, however, this is not a vulnerability.

In our example, we generated Diffie-Hellman parameters with 2048 bits. This is generally considered secure, but you are welcome to modify this value to generate larger prime candidates.

Remember to restart the SSH service after making these changes, whether you edited the file or generated your own:

```
sudo systemctl restart ssh
```

For more detailed information on how to decide whether to edit the existing file or generate your own, please refer to RFC4419 (<https://tools.ietf.org/html/rfc4419#section-7>).

SSH Access Control

Ideally, you should limit the number of users with server access to a bare minimum. Fortunately, OpenSSH Access Control Directives bring an excellent mechanism to set permissions and establish restrictions to users and groups that connect to your server.

There are four directives available executed in sequential order:

DenyUsers : block listed users from SSH service. **AllowUsers** : login is allowed only for users listed in this directive. **DenyGroups** : similar to **DenyUsers**, any user who belongs to this group(s) is blocked. **AllowGroups** : only users part of this group(s) can access the SSH service.

The default behavior of SSH is to allow any user to log in to the server, but this can be changed by adding access control directives to your `/etc/ssh/sshd_config` file. The following examples show how to filter who can access the service:

Selectively Deny Users

DenyUsers `phpadmin tempuser dbuser@198.51.100.0` - The users `phpadmin` and `tempuser` have no SSH access at all, but the user `dbuser` is only blocked if is connecting from the specific IP address of 198.51.100.0.

Selectively Allow Users

AllowUsers `superadmin megauser@yourdomain.tld dbadmin@198.51.100.0` - only the users `superadmin`, `megauser` and `dbadmin` have SSH access, but `megauser` and `dbadmin` can only connect from specific domains/IP addresses. On the other hand,

`superadmin` has the ability to connect from any computer.

Custom Rules Example

`/etc/ssh/sshd_config`

```
1 DenyUsers adam ben clark@198.51.100.0/24
2
3 AllowUsers clark dan@192.168.5.200 eva
```

Let's look at these rules in more detail. In the first line, Adam and Ben are blocked from accessing the server via SSH under any circumstances. Clark is not able to connect from a specific range of IP addresses, specified by `198.51.100.0/24`. In the second line, only Clark, Dan, and Eva will have SSH access. However, Clark's previous restriction from the `DenyUsers` line applies and Dan is only able to connect from one specific IP address. Eva is the only one who is able to connect via SSH from any computer.

Remember to restart your SSH service after changes have been made:

```
sudo systemctl restart ssh
```

Use a Strong Password for your Key-pair

In the *Securing Your Server* (</docs/security/securing-your-server/>) guide, you're encouraged to use SSH Key Pair Authentication. This is not optional if you are serious about security. But what about remote users that connect to the server with their laptops, which are susceptible to be stolen or lost? Here is where protecting your private key with a strong password or passphrase comes in, at least to gain time before changing the server keys. A strong password shouldn't be dictionary based. If security is your main concern, the convenience of an easy to remember password isn't adequate. [OpenSSL](https://www.openssl.org/) (<https://www.openssl.org/>) offers an easy way to generate pseudo-random passwords:

```
openssl rand -base64 32
```

The command will generate a 44 character long, base64-encoded password like this:

```
C4b7Yep8HGQFeG6kbnpEtrm+bg0+958xf1f85PU3/e0=
```

This kind of password is very hard to crack, but has the obvious disadvantage of being inconvenient each time you log to your server. The recommended method is to use a good password manager, because using a plain-text file for this dilutes the purpose of using the password in the first place.

To add the generated password to your existing private key:

```
ssh-keygen -p -f ~/.ssh/id_rsa
```

This assumes you keep your client's private SSH key in its default location, `~/.ssh/id_rsa`. You can modify the file location as needed and use the same command to change your password in the future.

An alternative to this method is the use of a GPG smartcard (https://en.wikipedia.org/wiki/OpenPGP_card) or YubiKey (<https://www.yubico.com/products/yubikey-hardware/>) which should be stored in a different place from your laptop. You can find more information about this implementation in the guide *How to use a GPG key for SSH authentication* (</docs/security/gpg-key-for-ssh-authentication/>)

Chroot Users

OpenSSH has a useful directive called `ChrootDirectory` that can be used in `sshd_config` to 'jail' a user into any directory, similar to the behavior of `chroot_local=YES` in an FTP server. This allows sysadmins to implement an additional layer of security for those users that don't need access to all of a server's resources.

However, this strategy involves a time-consuming process to configure the jailed user. For the purposes of this guide we'll setup a bare environment with some basic commands available for a user we'll create called `restricted-user`.

1. Create a jail directory in any desired location for `restricted-user`:

```
sudo mkdir -p /home/chroot/restricted-user
```

2. Create the required `/dev` nodes using `mknod`:

```
sudo mkdir -p /home/chroot/restricted-user/dev/
sudo mknod -m 666 /home/chroot/restricted-user/dev/null c 1 3
sudo mknod -m 666 /home/chroot/restricted-user/dev/tty c 5 0
sudo mknod -m 666 /home/chroot/restricted-user/dev/zero c 1 5
sudo mknod -m 666 /home/chroot/restricted-user/dev/random c 1 8
```

3. Establish permissions as needed for the jail. In this example, we'll give ownership to root :

```
sudo chown root:root /home/chroot/restricted-user
sudo chmod 0755 /home/chroot/restricted-user
```

4. Create the remaining directories needed by restricted-user :

```
sudo mkdir -p /home/chroot/restricted-user/bin /home/chroot/restricted-user/lib/ /home/chroot/restricted-user/lib64/ /home/chroot/restricted-user/lib/x86_64-linux-gnu/ /home/chroot/restricted-user/etc/
```

5. Install the bash shell into the jail:

```
sudo cp -v /bin/bash /home/chroot/restricted-user/bin
```

6. For each binary copied into the jail, we need to determine which shared libraries are needed. This is done through the ldd command:

```
sudo ldd /bin/bash
```

7. Manually copy each library from the output to the corresponding location in our restricted-user directory. This is necessary because our user won't have access outside its own jail.

```
cp -v /lib/x86_64-linux-gnu/{libncurses.so.5,libtinfo.so.5,libdl.so.2,libc.so.6} /home/chroot/restricted-user/lib/
cp -v /lib64/ld-linux-x86-64.so.2 /home/chroot/restricted-user/lib64/
cp -va /lib/x86_64-linux-gnu/libnss_files* /home/chroot/restricted-user/lib/x86_64-linux-gnu/
```

8. Next, create the restricted-user . You may use a different name for your user if you like:

```
sudo adduser restricted-user
```

9. Copy user information to the jail:

```
sudo cp -vf /etc/{passwd,group} /home/chroot/restricted-user/etc/
```

10. Finally, edit your /etc/ssh/sshd_config file to configure your new user:

/etc/ssh/sshd_config

```
1 Match User restricted-user
2 ChrootDirectory /home/chroot/restricted-user
```

11. Restart your SSH service to apply these changes.

```
sudo systemctl restart sshd
```

Keep in mind that our restricted user can't use any command or binary that is not manually installed in its jail environment. The complete process to set up a user for specific tasks is overwhelming but has the advantage that anything outside the jail is protected in the event that the user is compromised.

Regularly Update your Revoked Keys List

There are cases where you want to revoke specific public keys to prevent attempts to log in with them. For example, if you rotate your SSH keys every few months, you may want to disable them from being used in the future. OpenSSH has a directive to do just that:

`RevokedKeys` . Simply edit your `/etc/ssh/sshd_config` and add the desired location for revoked keys list:

/etc/ssh/sshd_config


```
1 RevokedKeys /etc/ssh/revoked_keys
```

The list should contain one key per line in plain text format. Remember to restart your SSH service each time you add a new key to the file.

Reduce Timeout Interval and Login Grace Time

Unattended SSH sessions are a major security risk. Leaving a remote connection open may allow unauthorized users the power to do whatever they like with your server. After all, you're already logged. The idle time interval comes in useful to prevent this situation, all you need is to configure two directives:

`ClientAliveInterval` set the time in seconds before the server sends a message requesting a client response. Default is 0, which means the server won't request a client response (the session can last forever). `ClientAliveCountMax` specifies the number of client alive messages sent by the server before closing the connection if no response is returned. The default is 3.

For example, if you decide to limit an idle SSH connection to 2 minutes before disconnecting the client, you can configure it as follows:

```
ClientAliveInterval 40
ClientAliveCountMax 3
```

This sends 3 messages to your client (every 40 seconds) requesting some action, after 120 seconds without response the user will be logged out. An alternative configuration could be:

```
ClientAliveInterval 120
ClientAliveCountMax 0
```

With these settings, if no messages are sent, the server will disconnect any SSH session after 2 minutes of inactivity.

To apply any combination that suits your needs just edit your server's `/etc/ssh/sshd_config` and restart your SSH daemon afterwards.

Use a Warning Banner

This step will not harden your server security, but your legal-conscientious-parameters. In some locations, the simple fact of warning unauthorized users of the consequences of their actions is determinant for taking legal action. To display a warning banner each time a user logs to your server, add the following line to the `/etc/ssh/sshd_config` configuration:

`/etc/ssh/sshd_config`

```
1 Banner /location/of/WarningMessage
```

The value `/location/of/WarningMessage` is the path to your banner text file. The following banner example was taken from Ubuntu's Community Help page (<https://help.ubuntu.com/community/SSH/OpenSSH/Configuring>) for OpenSSH:

```

*****
****
                                NOTICE TO USERS

This computer system is the private property of its owner, whether
individual, corporate or government. It is for authorized use only.
Users (authorized or unauthorized) have no explicit or implicit
expectation of privacy.

Any or all uses of this system and all files on this system may be
intercepted, monitored, recorded, copied, audited, inspected, and
disclosed to your employer, to authorized site, government, and law
enforcement personnel, as well as authorized officials of government
agencies, both domestic and foreign.

By using this system, the user consents to such interception, monitori
ng,
recording, copying, auditing, inspection, and disclosure at the
discretion of such personnel or officials. Unauthorized or improper use
of this system may result in civil and criminal penalties and
administrative or disciplinary action, as appropriate. By continuing t
o
use this system you indicate your awareness of and consent to these te
rms
and conditions of use. LOG OFF IMMEDIATELY if you do not agree to the
conditions stated in this warning.

*****
*****

```

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

Find answers, ask questions, and help others. (<https://www.linode.com/community/questions/>)

This guide is published under a CC BY-ND 4.0 (<http://creativecommons.org/licenses/by-nd/4.0>) license.