

How to Harden Server SSH Access Using Advanced OpenSSH Features

Learn advanced techniques to secure your server's SSH access using OpenSSH features like key authentication, fail2ban, and port knocking to prevent unauthorized access.

Contents

| | | |
|----|--------------------------------------|----|
| 01 | Introduction | 3 |
| 02 | Control SSH Access | 4 |
| 03 | Limit Login Attempts | 10 |
| 04 | Grant SSH Access to Authorized Users | 13 |
| 05 | Restrict SSH User Actions | 15 |
| 06 | Display Warning Banners | 20 |
| 07 | Test | 21 |
| 08 | Conclusion | 22 |
| 09 | Next Steps | 22 |

Introduction

Secure Shell (SSH) is a network communication protocol that allows users to securely connect to a remote computer over insecure networks such as the Internet. SSH follows a client-server architecture where the client application connects to the SSH Daemon hosted on a destination server.

OpenSSH (OpenBSD Secure Shell) is an SSH daemon implementation that includes a suite of tools to securely listen for incoming connections, authenticate, and grant server access to permitted clients. To protect a public server from common network attacks, harden SSH configurations using advanced OpenSSH features that let you control who, how, and when your server accepts remote console connections.

This article explains how you can harden Server SSH access using Advanced OpenSSH features. You are to control SSH client activity, set connection trust levels, jail specific users to directories, and implement access intervals to protect the server from common SSH attacks.

Prerequisites

Before you begin:

- Deploy a [development server on Vultr](#)

This article uses a Ubuntu 22.04 server. To implement the steps on a production server, it's safe to test them on a development server

- Use [SSH to access the server](#)
- Create a non-root sudo user
- Switch to the sudo user account

```
# su example-user
```

SSH Configuration Overview

Hardening your SSH server configuration allows you to:

- Control how system users can log in to the server
- Limit user login attempts
- Decide what users can log in to the server over SSH
- Restrict what permitted users can do when logged in to the server

To change and harden the SSH server configurations, you must edit the main OpenSSH configuration file `/etc/ssh/sshd_config`. Every uncommented line in the file represents an active configuration you can set to match your desired settings.

Before making changes to the SSH server configuration, back up the file

```
$ sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.ORIG
```

The above command makes a copy of the configuration file you can rename back to roll back any changes

Control SSH Access

In this section, implement the first layer of SSH defense by applying controls on the authentication process. By setting how users can log in, and how long a session stays idle, you can protect your server from SSH brute force attacks. Control user access to the server as described in the following steps.

Using a text editor such as `Nano` or `Vim`, edit the main SSH configuration file

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

Generate Strong SSH Key Pairs

The `ssh-keygen` utility generates keys based on different algorithms including RSA, DSA, ECDSA, and ED25519. RSA keys are only secure with keys longer

than 2048 bits. ECDSA and ED25519 are both newer algorithms based on elliptic curve cryptography unlike DSA. Hence, it's advisable to use ED25519 keys as below.

1. On your local machine, generate an ED25519 key pair

```
$ ssh-keygen -t ed25519 -f /path/to/key/file
```

2. Copy the generated public key

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

3. To add your SSH key to the authorized keys, create the `.ssh` if its not created on your server

```
$ mkdir -p ~/.ssh
```

4. Using the `echo` utility, append the `~/.ssh/authorized_keys` file as below

```
$ echo "your-public-key" >> ~/.ssh/authorized_keys
```

5. Apply the correct permissions on the file

```
$ sudo chmod 600 ~/.ssh/authorized_keys
```

6. Restart the SSH daemon to save changes

```
$ sudo systemctl restart sshd
```

Disable Password Logins

Compared to SSH keys, attackers brute-force SSH servers using passwords. Disable password-based logins for all system users as described in the steps below.

1. To disable password-based logins and only accept SSH keys. Find the following configuration line

```
PasswordAuthentication yes
```

2. By default, it's set to `yes`, change it to `no` to disable password authentication

```
PasswordAuthentication no
```

3. To tighten the server setup, disable other authentication methods as below

```
ChallengeResponseAuthentication no  
KerberosAuthentication no  
GSSAPIAuthentication no
```

Save the file.

Disable Root Login

It's recommended to add administrative users to the `sudoers` group instead of using the `root` account directly. This is because rogue users and applications can compromise the entire server by accessing the `root` account. To tighten server security, disable SSH access for the `root` user as described below.

1. Find the following directive

```
PermitRootLogin yes
```

2. Change it to `no` as below

```
PermitRootLogin no
```

Optionally, to allow the root login, but limit password usage as below

```
PermitRootLogin prohibit-password
```

The above directive allows root login using SSH keys, but not the `root` password. This allows a single administrator to have exclusive access to the root shell

Save the file

Before disabling root login on the server, verify that you have active non-root sudo users to run administrative tasks on the server. To switch to the `root` shell using a sudo user account, run the following command

```
$ sudo su
```

Change the Default SSH Port

By default, SSH uses port `22` to secure remote connections to a server. Most SSH security threats target and test the default SSH port `22` on your server before an attack. To harden your production server security, change the default port as described in the steps below.

1. Find the following commented configuration line

```
#Port 22
```

2. Uncomment, and change the number to update the SSH listening port. For example, change it to `9010`

```
Port 9010
```

Save and close the file

3. To allow incoming SSH connections on the new TCP port `9010`, allow the port through the firewall

```
$ sudo ufw allow 9010/tcp
```

The above command allows port `9010` through the default UFW firewall active on Vultr Ubuntu servers. On CentOS and RHEL distributions, use Firewalld

4. Restart the firewall to apply changes

```
$ sudo ufw reload
```

5. In a new terminal window, test the SSH connection to your server on port `9010`

```
$ ssh -p 9010 example-user@SERVER-IP
```

The above command specifies the correct port `9010` using the `-p` option. When unspecified, SSH connects to port `22` which drops the connection

Changing the SSH port number is "security by obscurity". It does not prevent a determined attacker from discovering the correct port number, but saves the server from a majority of scanners and bots that search for vulnerable public servers. This keeps the server logs clean and lets you focus on more serious threats. Additionally, when the default port changes, instruct system users to specify the new SSH port to connect to the server.

Set Up Idle Timeouts

An idle open connection is a standalone threat as a malicious user can gain remote or physical access to the client machine, and execute commands on behalf of the user. To protect system users, the `ClientAliveInterval` configuration checks if a client has not issued any commands for a while, and sends a message to check if the client is still active. When the client is active, it responds to the message, or the server repeats the process based on the `ClientAliveCountMax` value before marking the client as inactive and closing the connection.

It's important to set lower Idle timeouts to only keep SSH connections open when a user is actively using the session. In this section, set your desired timeout values as described below.

1. Edit the main OpenSSH configuration file

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

2. Find the following commented configuration line

```
#ClientAliveInterval 0
```

3. Change it to a lower number. For example `120` to represent 2 minutes

```
ClientAliveInterval 300 # in seconds
```

4. Find the client keep alive directive

```
#ClientAliveCountMax 3
```

5. Uncomment, and change the value to `2` or lower to implement a strict idle timeout of `4` minutes

```
ClientAliveCountMax 3
```

Save and close the file

6. Test the SSH configuration for errors

```
$ sudo sshd -t
```

7. Restart the SSH daemon to apply changes

```
$ systemctl restart sshd
```

> The above settings allow a client to stay inactive for 4 minutes before the SSH connection closes. When setting the Idle timeouts, you must strike a balance between security and the user experience. Low values force the SSH session to exit after a short idle period. This makes it more secure. However, users log in repeatedly each time they keep the SSH session idle, which degrades their experience.

Limit Login Attempts

SSH controls prevent attackers from gaining unauthorized access to the server. However, the SSH server processes every login request before it grants or denies access to the user. Registering many unsuccessful login attempts throttles server performance, and contributes to a Denial of Service (DoS) attack.

In this section, limit SSH Login attempts on your server to mitigate potential DoS attacks.

Limit the Maximum Login Attempts

To limit the number of times a user should try to enter the correct password before ending the connection request, set the `MaxAuthTries` value to a lower number as below.

1. Find the following commented configuration line

```
#MaxAuthTries 6
```

2. Uncomment, and change the value to a lower number. For example `2`

```
MaxAuthTries 2
```

The above value sets the maximum log in attempts for every user to `2`. When a user enters a wrong password twice, the SSH connection closes with an error message, and the user can retry by launching the connection again.

Save the file.

The above configuration allows users to try accessing the server using a valid password twice. In cases where a user has multiple SSH keys, a timeout occurs depending on your maximum value. When using multiple SSH keys, a user must specify the key path to connect to the server without hitting the limit as below.

```
$ ssh -i /path/to/key/file example-user@SERVER-IP
```

Reduce the Login Grace Period

When a user connects to the server, the SSH server waits for 120 seconds (2 minutes) for the user to successfully authenticate with a correct password or SSH key. If the user does not authenticate within this time, the connection closes with an error. During this grace period, an attacker may try multiple methods such as script-uploads to gain access to the server. Reduce the maximum allowed Grace period to limit how long a user may delay before authenticating with the server.

1. Find the commented configuration line below

```
#LoginGraceTime 2m
```

2. Uncomment and change the value to a lower number in minutes. For example `30` to limit the grace time to 30 seconds

```
LoginGraceTime 30
```

Save the file.

Limit the Number of Unauthenticated Connections

By default, when 10 simultaneous connections are pending authentication, the SSH daemon randomly starts rejecting 30% of all new connection attempts. Control the rejection rate using the `MaxStartups` directive. By default it's set to `10:30:100`, update it to reject connections based on your values as below.

1. Find the commented line below

```
#MaxStartups 10:30:100
```

2. Uncomment and change the value to `5:30:10`

```
MaxStartups 5:30:10
```

The above value rejects new connections if 5 open connections are pending authentication, and does not accept any new connections if more than 10 unauthenticated connections are pending

Save the file.

Depending on your server users, large enterprise servers require higher values to handle the user traffic well. The above example value is suitable for smaller servers with 1 to 5 simultaneous SSH users.

Limit the Number of Active Sessions per User

Sometimes, SSH users may open a maximum of 5 SSH connections depending on the ongoing operations. However, sometimes attackers may launch extra sessions during MITM (Man in the Middle) attacks. To tighten your server security, limit the number of active sessions per user as described below.

1. Find the following commented line

```
#MaxSessions 10
```

2. Uncomment and change the value to `5`

```
MaxSessions 5
```

Save and close the file

The above value limits the number of allowed active sessions per user to `5`. When the user launches more than `5` sessions, the additional session does not connect successfully.

Grant SSH Access to Authorized Users

System users consist of administrative (sudo), and standard users. Depending on your server setup, only a specific user group, for example, `IT department` may require SSH privileges. To harden your SSH server, set what system users use SSH on the server using the `AllowUsers`, `DenyUsers`, `AllowGroups`, and `DenyGroups` directives as below.

This section uses the following example values, replace all occurrences with your actual details:

- **Users:** example-user, Johndoe, user2
- **Groups:** IT_admins
- **IP Addresses:** 192.0.2.100, 192.0.2.8, 192.0.2.50

1. Edit the OpenSSH configuration file

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

2. To only allow the users, `example-user` and `user2` to log in via SSH. Add the following directive at the end of the file

```
AllowUsers example-user user2
```

The above rule allows `example-user` and `user2` to log in from any IP address

3. To allow specific users to log in only from specific IP addresses, add the following directive

```
AllowUsers example-user@192.0.2.100 user2@192.0.2.8
```

4. To allow any permitted user to log in from trusted IP addresses, add the directive below

```
AllowUsers *@192.0.2.100 *@192.0.2.8 *@192.0.2.50
```

5. To allow any user from a block 8 subnet

```
AllowUsers *@trusted.ip.subnet/8
```

For example, if you allow users from the subnet `192.0.2.0/8`, it means all users with the source IP addresses `192.0.2.1` to `192.0.2.255` can log in to the server

To use wild card IP Addresses, you can rewrite the above directive as below

```
AllowUsers *@192.0.2.*
```

6. To grant SSH access to users of a specific group, for example, `IT_Admins`, add the following directive to the file

```
AllowGroups IT_admins
```

Save and close the file.

The above rule specifies that only users belonging to the `IT_admins` can log in to the server using SSH.

7. Test the SSH configuration for errors

```
$ sudo sshd -t
```

When `AllowUsers` and `AllowGroups` directives are not available in the OpenSSH configuration file, all users and groups can log in to the server using SSH. However, when directives are available, access is only granted to specific users and groups, no other system user can log in using SSH on the server.

Restrict SSH User Actions

In addition to allowing specific users and user groups, it's also possible to restrict the actions a particular user can perform on the server. This adds an extra security layer that limits any unauthorized users from performing specific actions on the server.

In this section, limit SSH user actions by commands, SSH keys, or command sets as described below.

User and Group Restrictions

1. To apply custom rules on individual users, use the `Match` directive. For example, to work on the `example-user`, add the following configuration at the end of the file

```
Match User example-user
```

2. To restrict the user to run a specific command on the server, for example, `top`, add the following directive immediately after the `match` declaration

```
Match User example-user
ForceCommand "top"
```

The above configuration block grants `example-user` permissions to run the `top` command, and denies the user permission to any other commands on the server. Upon login, the `top` command automatically runs in the user's session. When the user exits the `top` application, the session ends

3. To restrict a user to running only a specific script on the server. Add the following directive to your configuration

```
ForceCommand "/path/to/script.sh"
```

4. To match a specific user group instead of a single user, use `Match Group` to create a new block as below

```
Match Group IT_Admins
ForceCommand "top"
```

Save and close the file

The above Match block limits users in the group `IT_Admins` to strictly the `top` command. When a user exits the application, the SSH session ends

5. Test the SSH configuration for errors

```
$ sudo sshd -t
```

> A common use case when restricting users and groups using the `Match` block is [setting up SFTP-only accounts on the server.](#)

SSH Key Restrictions

To achieve add an extra layer of security, attach restrictions to the use of individual SSH keys. The `~/.ssh/authorized_keys` file contains the list of all SSH keys authorized to access that specific user account. By default, keys appear in the format below:

```
ALGORITHM KEY USER@HOST
```

For example:

```
ecdsa-sha2-nistp521 AAAAE2VjZ...KEY-BODY...Lf+FNXv00Pd+A== user@user's-PC
```

In the above example, `ecdsa-sha-nistp521` is the algorithm used to generate the key. The SSH Key body starts with `AAAA` and ends with `00Pd+A==`. `user@user's-PC` is the username and hostname of the user authorized to use this key.

1. To apply SSH Key-specific restrictions, add the restriction immediately before the listing. For example, to restrict the user to run only a specific command, use the `command=""` flag as below

```
command="ls"
```

The above declaration limits the SSH Key user to only the `ls` command

2. Apply the command restriction to an SSH Key

```
command="ls" ecdsa-sha2-nistp521 AAAAE2VjZ....KEY-BODY...Lf+FNXv00Pd+A==  
user@user's-PC
```

When the user starts an SSH session with this key, the command `ls` executes, and the session ends if the command runs successfully

Restrict Users to a Set of Commands

1. To restrict a user to a set of commands, create a custom script that allows a user to select from a pool of available commands. For example, the following script allows the user to choose between the `ls` and `top` commands.

```
#!/bin/bash  
  
echo "1. ls"  
echo "2. top"  
read -p 'Choose the number corresponding to one of the above options: ' choice #  
Read the choice from the user  
case $choice in  
  1)  
    ls  
    ;;  
  2)  
    top  
    ;;  
  *)  
    exit  
    ;;  
esac
```

Save the script in your home directory as `ssh_script.sh`

2. Grant execute privileges on the script

```
$ chmod u+x /home/example-user/ssh_script.sh
```

To execute the script, either use the `ForceCommand` or `command=` flags before an authorized SSH key

3. Apply the script to the target SSH Key user using `command=""` as below

```
command="/home/example-user/ssh_script.sh" ecdsa-sha2-nistp521  
AAAAE2VjZ...KEY-BODY...Lf+FNXv00Pd+A== an@fbsdvm
```

4. In the main OpenSSH configuration file, use the `Match` block to apply the script to `example-user` as below

```
Match User example-user  
ForceCommand "/home/example-user/ssh_script.sh"
```

Save and close the file

When `example-user` logs in to the server, `script.sh` runs with a condition for the user to select the target section by number. The user can either choose 1 for `ls`, or 2 for `top`. The system runs the selected command and closes the connection when complete.

Chroot System Users

OpenSSH allows you to jail system users to specific directories. This is important when limiting users to specific system directories instead of unlimited access to all server files. In return, only specific users can access certain directories which creates an extra security layer to safeguard server data.

In this section, jail `user2` to the home directory, and disable access to any other non-home directories as described below.

1. Edit the OpenSSH configuration file

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

2. Add a new `Match` block and specify the user jail directory as below

```
Match User user2
ChrootDirectory /home/user2/
```

Optionally, to limit an entire user group such as `Finance_Admins` to a shared directory such as `/usr/share/department` add the following directive

```
Match Group Finance_Admins
ChrootDirectory /usr/share/department/
```

Save and close the file

3. Test the SSH configuration for errors

```
$ sudo sshd -t
```

4. Link any shared resources to the jail directory. For example, to link and share files from the `/opt` directory, run the following command

```
$ sudo ln -s /opt/files /home/user2/files
```

5. Set the correct directory permissions

```
$ sudo chmod 755 /home/user2/files
```

6. When using Groups, change the Chroot directory ownership to the group as below

```
$ sudo chown :Finance_Admins /usr/share/department/
```

7. Grant all group users read, write, and execute privileges on the directory

```
$ sudo chmod 770 /usr/share/department/
```

By setting Chroot directives in your SSH configuration, you can jail users to a specific directory and link any necessary files to the directory for access.

Display Warning Banners

Depending on the number of SSH users on the server, it's important to display a banner MOTD with warnings or important notices. When a user establishes an SSH connection to the server, the message displays before any command runs in the session. A typical warning banner can include security policies, privacy policy, legal rights, and system usage rules.

1. To set up the SSH warning banner, create a new text file in an accessible location such as `/opt`

```
$ sudo touch /opt/ssh-banner.txt
```

2. Edit the file

```
$ sudo nano /opt/ssh-banner.txt
```

3. Add the following warning banner contents to the file

```
*****  
This is a private computer.  
By using this system, you consent to having all activity monitored.  
System administrators may provide activity logs to law enforcement.  
Unauthorized users or usage may be subject to legal action.  
Do not store private data on this computer.  
Users are responsible for securely storing their personal access keys.  
*****
```

Save and close the file

Change the above warning banner with your desired notice that displays alongside the system `/etc/motd` message.

4. To apply the warning banner, edit the OpenSSH configuration file

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

5. Add the following directive to the file

```
Banner /opt/ssh-banner.txt
```

Save and close the file

6. Test the OpenSSH configuration for errors

```
$ sudo sshd -t
```

7. Restart the SSH daemon to apply changes to your server configuration

```
$ sudo systemctl restart sshd
```

Test

To verify that your SSH server configurations are correctly applied, establish a new connection to the server, and test your configuration per user account as below.

1. Verify that you cannot connect to the server using a regular SSH command

```
$ ssh example-admin@SERVER-IP
```

2. Connect to the server using your custom port `9010`

```
$ ssh -p 9010 example-admin@SERVER-IP
```

3. When connected, end the SSH session

```
$ exit
```

4. Establish a new SSH connection using a restricted user account or member of a restricted group

```
$ ssh -p 9010 example-user@SERVER-IP
```

5. When connected Verify that the user can only access the `top` command
6. Press `:Key:Ctrl: + :Key:Q:` to exit the `top` application, and verify that your SSH connection ends
7. Establish a fresh SSH connection, and enter a wrong password for the user account. Verify that SSH blocks your request on the second try

```
$ ssh -p 9010 user2@SERVER-IP
```

Depending on your SSH configurations, you can restrict specific users and groups to specific actions and durations on the server to match your security policies.

Conclusion

In this article, you have hardened your SSH server using OpenSSH advanced features on a Vultr Ubuntu server. When your SSH server contains secure configuration directives, most security threats fail to pass through your server limitations, as such, your production data and applications run correctly without any corrupted files.

For more information about the available OpenSSH advanced configuration options, view the fifth section of the `sshd_config` manual page.

```
$ man 5 sshd_config
```

Next Steps

For more server configuration options, visit the following resources:

- [How to Use Two-Factor Authentication with Sudo and SSH on Linux with Google Authenticator](#)

- [Best Practices for SSH on a Production Cloud Server](#)
- [Use SSHFP Records to Verify SSH Host Keys](#)
- Securely Transfer Files Over a Virtual Private Cloud (VPC) with SCP or Rsync
- [Setup SFTP User Accounts on Ubuntu 20.04](#)
- [How to Use SSH with Vultr Servers](#)
- [Enable SSH Login Notification on Linux](#)



VULTR

