10 Steps to Secure Open SSH

Devolutions

WE ARE FULL THROTTLE ON SSH!

Well we might as well call this month our SSH month since we are full throttle on SSH! Following our "Reverse SSH Port Forwarding" and "How to Configure SSH Tunnel on Putty", we thought it would be good to look at securing Open SSH, since security is RDM's number one priority!

When talking about remote access standard, Open SSH has become the norm. It has made other protocols such as Telnet unnecessary since SSH encrypts your connection and passwords are no longer sent in plain text.

Still, using the default installation for SSH can have its own downfalls when it comes to security. When running an SSH server, there are a few easy steps that will considerably increase the installation's level of security.

Here is our top 10 list for how to secure your Open SSH:

1. Strong Usernames and Passwords

If you have SSH running and exposed to the outside world you will probably notice some log attempts made by hackers that are trying to guess your username and password. A hacker will usually try to scan for port 22 (default port) to discover machines running SSH; they will then try a heavy-handed attack against it. A strong password will help win the fight against an attack! Use RDM Password Generator to generate strong and random passwords and use our list of Forbidden Passwords to prevent anyone from using 123456 or other such nonsense.

2. Configure Idle Timeout Interval

To avoid having an unattended SSH session, you can set an Idle timeout interval. Open your /etc/ssh/sshd_config file and add the following line:

```
ClientAliveInterval 360
ClientAliveCountMax 0
```

The idle timeout interval you are setting is in seconds (360 secs = 6 minutes). Once the interval has passed, the idle user will be automatically logged out.

3. Disable Empty Passwords

You need to prevent remote logins from accounts with empty passwords for added security. Open your /etc/ssh/sshd_config file and update the following line:

```
PermitEmptyPasswords no
```

4. Limit Users' SSH Access

To provide another layer of security, you should limit your SSH logins to only certain users who need remote access. This way, you will minimize the impact of having a user with a weak password.

Open your /etc/ssh/sshd_config file to add an 'AllowUsers' line, followed by the list of usernames, and separate them with a space:

```
AllowUsers user1 user2
```

Then restart your SSHD service by entering one of the following commands:

```
/etc/init.d/sshd restart
```

or

```
service sshd restart
```

5. Disable Root Logins

One of the most dangerous security holes you can have in your system is to allow direct logging in to root through SSH. By doing so, any hackers attempting brute force on your root password could hypothetically access your system; and if you think about it, root can do a lot more damage on a machine than a standard user can do.

To disable your Root Logins, you'll need to edit the SSHD configuration file. All your SSH server settings are stored in the /etc/ssh/sshd_config file. Open that file while logging on as root and find the section in the file containing `#PermitRootLogin` in it.

To disable logging in through SSH as root, change the line to this:

```
PermitRootLogin no
```

The # symbol (comment) tells the server to ignore anything after it on the same line. Remove the # for the changes to take effect.

Then restart your SSHD service by entering one of the following commands:

```
/etc/init.d/sshd restart
```

or

```
service sshd restart
```

You are now safe from brute force root login. If you then need to access root, simply log in as a normal user and use the **su** command.

6. Only Use SSH Protocol 2

SSH has two protocols that it can use. Protocol 1 is older and is less secure. Protocol 2 is what you should be using to harden your security. If you are looking for your server to become PCI compliant, then you must disable protocol 1.

Open your /etc/ssh/sshd_config file and look for the following line:

```
#Protocol 2, 1
```

Remove the 1, then uncomment it (remove the #). The final line should look like this:

```
Protocol 2
```

Then restart your SSHD service by entering one of the following commands:

```
/etc/init.d/sshd restart
```

or

```
service sshd restart
```

7. Use Another Port

One of the main benefits of changing the port and using a non-standard port is to avoid being seen by casual scans. The vast majority of hackers looking for any open SSH servers will look for port 22, since by default, SSH listens for incoming connections on that port. If it's harder to scan for your SSH server, then your chances of being attacked are reduced.

You can choose any unused port as long as it's not used by another service. A lot of people might choose to use 222 or 2222 as their port since it's pretty easy to remember, but for that very reason, hackers scanning port 22 will likely also be trying ports 222 and 2222. Try and select a port number that is not already used, follow this link for a list of port numbers and their known services.

To change your port, open your /etc/ssh/sshd_config file and add the following lines:

```
#Run SSH on a non standard port
Port 2025 #Change me
```

Then restart your SSHD service by entering one of the following commands:

```
/etc/init.d/sshd restart
```

or

```
service sshd restart
```

Don't forget to make the changes to port forwarding in your router and any necessary firewall rules. You will also need to advise your client of any port changes so they know which port to connect to since SSH will no longer be listening for connections on the standard port.

8. Allow Only Specific Clients

If you want your server to be reachable from only a specific IP address on port 22, then you should consider filtering connections at your firewall by adding a firewall rule on your router or update your iptables like this:

```
iptables -A INPUT -p tcp -s YourIP --dport 22 -j ACCEPT
```

With that rule, you are opening the SSH port only to YourIP. If it is essential for you to open the SSH port globally, then iptables can still help prevent heavy-handed attacks by logging and blocking repeated attempts to login from the same IP address.

The following rule records the IP address of each new attempt to access port 22:

```
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --se
```

The following rule verifies if that IP address has tried to connect three times or more within the last 90 seconds. If it hasn't, then the packet is accepted (this rule would need a default policy of DROP on the input chain).

```
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent ! --
```

Filtering at the firewall is a tremendously useful method of securing access to your SSH server.

9. Enable Two-Factor Authentication

Your SSH servers should be secured with Two-Factor Authentication configured on it. It is one of the main protections you can add to your SSH servers to protect them from unauthorized access since each user login must tie back to a configured 2FA user.

Even if a hacker manages to get a hold of your password or breaks into your SSH server, they will still get blocked by the 2FA. Follow this link to learn more about securing SSH with two factor authentication using Google Authenticator.

10. Use Public/Private Keys for Authentication

Public/Private Keys authentication is certainly more secure and a much better solution than password authentication. Each key is a large number with different mathematical properties. The Private Key is stored on the computer you login from, while the public key is stored on the .ssh/authorized_keys file on each computer you want to login to.

This is particularly important if the computer is visible on the Internet. Using encrypted keys for authentication is useful as you won't need to enter a password anymore. Once the public/private key-pair authentication has been configured on the server, you can completely disable password authentication; this means that no one without an authorized key will be able to gain access. Even the most inventive hackers won't be able to interfere or sneak onto a session, and no more cracking password attempts.

Here's how to create a public/private key pair and install them for use on your SSH server:

Start by generating your key-pair, a public key and a private key. The public key will be placed on the server and you will login with your private key (this needs to be performed on each client machine from which you connect):

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

This will create two files in your (hidden) ~/.ssh directory called: id_rsa and id_rsa.pub The first: id_rsa is your private key and the other: id_rsa.pub is your public key.

You will then be asked for a passphrase, which is your password to unlock a given public key each time you connect. It is your choice to add a passphrase protective encryption to your key when you create it. If you prefer not to use one, simply press Enter when asked for the passphrase when creating your key pair. Be aware that if you do not passphrase protect your key, anyone gaining access to your local machine will automatically have SSH access to the remote server.

Copy the public key (id_rsa.pub) to the server (the remoteuser should never be root; select the default non-root user as remoteuser):

```
Scp –p id_rsa.pub remoteuser@remotehost:
```

Then login with SSH and copy the public key to the right place:

```
ssh remoteuser@remotehost mkdir ~/.ssh chmod 700 ~/.ssh cat id_rsa.pub
```

Then delete the public key from the server, otherwise the SSH client won't allow you to login to the server:

```
rm id_rsa.pub
```

And finally, set file permissions on the server:

```
$ chmod 700 ~/.ssh
$ chmod 600 ~/.ssh/authorized_keys
```

If StrictModes is set to yes in your etc/ssh/sshd_config file, then the above permissions are required. When logging in to the server, you will be prompted for the key passphrase (depending on your configuration).

Once you have verified that you can successfully login using your public/private key, you can then completely disable password authentication. Open your /etc/ssh/sshd_config file and add the following lines:

```
# Disable password authentication forcing use of keys
PasswordAuthentication no
```

And there you go! Our top 10 steps to harden your SSH installation! We truly hope this will help you with the sudden rise in SSH brute force attacks. Securing SSH is more important now than ever!

As always, please let us know your thoughts by using the comment feature of the blog. You can also visit our forums to get help and submit feature requests, you can find them here.