SERVERS FOR HACKERS

FEBRUARY 24, 2015

Configuring SSHD on the Server

Configure how SSH runs on the server for better security.
We'll log into a server and edit the `/etc/ssh/sshd_config` file, to change how users can use SSH to log into the server from remote locations.

Beginners

4 Series • 8 Articles

PREVIOUS: Using the SSH Config File

Configure how SSH runs on the server for better security.

We'll log into a server and edit the `/etc/ssh/sshd_config` file, to change how users can use SSH to log into the server from remote locations.We previously have used our local `~/.ssh/config` file to easily log into a server. Let's now see some SSH options on the remote server, to see how we can affect who can log in and how.

SSH configuration is generally found in `/etc/ssh/sshd_config`.

Change SSH Port

We can change the port users use to login away from port 22.

```
sudo vim /etc/ssh/sshd_config
```

Change the `Port` option to something other than 22:

```
# Formally "Port 22"
Port 2222
```

Then restart SSH:

```
sudo service ssh restart
```

We can then try to login from our local computer by adjusting the port to use:

```
ssh -p 2222 ssh-ex
```

I usually keep port on standard port 22 and use other security means to lock this down.

Create New Admin User

We don't want user root to be able to login over SSH, as that user has no limits in privileges.

Creating a new user, who can use "sudo", but isn't the root user, adds security:

1. SSH key usually provides a first (separate) password needed, so attackers need both the SSH private key and knowledge of this password
2. User then required to use their own password on top of that to run privileged commands via "sudo". This is missing when logged in as "root" directly.
3. We remove a vector of remote attack - root user cannot be logged into remotely

On the remote server, logged in as user root, we create a new user `fideloper`:

```
adduser fideloper

# When done, check some items to see that user/group
"fideloper" exists:
cat /etc/passwd | grep fideloper
cat /etc/group | grep fideloper
```

Then add user `fideloper` to group `sudo`, which allows that user to use "sudo" commands:

```
# Append secondary group "sudo"
# to user "fideloper":
usermod -a -G sudo fideloper

# Check that it worked by seeing
# groups assigned to user "fideloper":
groups fideloper
```

Log in as user "fideloper"

```
# Chang into user fideloper:
sudo su fideloper

# See that we can use the sudo command:
sudo service nginx status
```

Next we want to make sure we can log into user "fideloper". To do so, we get our previously created `id_sshex.pub` (on our local computer) and paste it into the `authorized_keys` file of the new "fideloper" user on the remote server.

```
# On local computer:
cat ~/.ssh/id_sshex.pub | pbcopy

# On the remote server, edit authorized_keys
# and paste in the public key:
vim ~/.ssh/authorized_keys
```

Then, once again locally, edit the `~/.ssh/config` file to adjust user from "root" to "fideloper":

```
# Edit local ssh config file:
sudo vim ~/.ssh/config
```

Make it look like this:

```
Host ssh-ex
    HostName 104.236.90.57
    User fideloper
    IdentitiesOnly yes
    IdentityFile ~/.ssh/id_sshex
```

Save that and then attempt to log into our server (again, from our local computer):

```
ssh ssh-ex
```

And we should get logged in as user "fideloper"!

Disable SSH login of user root

Back on the remote server, let's edit the `sshd_config` file some more and lock down who can login and how further.

```
sudo vim /etc/ssh/sshd_config
```

Disable the login of user root:

```
# Change this to no
PermitRootLogin no
```

Save that and restart SSH:

```
sudo service ssh restart
```

Back on our local computer, try to login as root again:

```
ssh -o "User root" ssh-ex
```

This will ask for a password but tell is permission is denied, even if using the right password.

Disable password authentication

On the remote server, edit `sshd_config` and turn off the ability to login over SSH using password:

```
sudo vim /etc/ssh/sshd_config
```

Edit the "PasswordAuthentication" directive:

```
# Uncomment and set to "no"
PasswordAuthentication no
```

Save that and restart SSH:

```
sudo service ssh restart
```

Locally, try to login as user "root" again and see you get permission denied:

```
ssh -o "User root" ssh-ex
```

Allow SSH login by user or group

On the remote server, edit `sshd_config` and explicitly set which users can SSH into the server:

```
sudo vim /etc/ssh/sshd_config
```

Add the "AllowUsers" directive:

```
AllowUsers fideloper
```

Save that and restart SSH:

```
sudo service ssh restart
```

Locally, log in as user "fideloper" successfully:

```
ssh -o "User fideloper" ssh-ex
```

Back on the remote server, let's use the "AllowGroups" directive instead:

```
sudo vim /etc/ssh/sshd_config
```

Add the "AllowUsers" directive:

```
# Comment out/remote "AllowUsers"
#AllowUsers fideloper

# Add "AllowGroups directive"
# Add groupname "allowssh"
AllowGroups allowssh
```

Save that and restart SSH:

```
sudo service ssh restart
```

Locally, add a new Terminal tab (**DON'T LOG OUT OF YOUR CURRENT SESSION**) and try to login again:

```
ssh ssh-ex
```

You'll get denied, as none of our users are in group "allowssh".

On the remote server, create that group and assign it to user "fideloper":

```
sudo addgroup allowssh

# Confirm group created
cat /etc/group | allowssh

# Add group to fideloper
# as user root
sudo su
usermod -a -G allowssh fideloper

# Confirm fideloper has that group
groups fideloper
```

Locally, try to login again and see that you can login:

```
ssh ssh-ex
```

In the end, I allow both "allowssh" and "sudo" group to login over SSH:

```
AllowGroup allowssh sudo
```

As always, save that and restart SSH:

```
sudo service ssh restart
```