VPS Hardening

Another necessary step in our configuration and setup of our VPS is the hardening of the system and access. We should limit our access to the VPS to SSH and disable all other services on the VPS. Finally, we will reduce the attack vectors to a minimum and provide only one possible access to our VPS, which we will secure in the best possible way. We should keep in mind that, if possible, we should not store any sensitive data on the VPS, or at least only for a short period when we perform an internal penetration test. In doing so, we should follow the principle that someone could gain access to the system sooner or later.

However, since in this case, the VPS is only used as a source for our organization and tools and we can access these resources via SSH, we should secure and harden the SSH server accordingly so that no one else (or at least no one other than the team members) can access it. There are many ways to harden it, and this includes the following precautions, but not limited to:

- `Install Fail2ban`
- `Working only with SSH keys`
- `Reduce Idle timeout interval`
- `Disable passwords`
- `Disable x11 forwarding`
- `Use a different port`
- `Limit users' SSH access`
- `Disable root logins`
- `Use SSH proto 2`
- `Enable 2FA Authentication for SSH`

`It is highly recommended to try these settings and precautions first in a local VM we have created before making these settings on a VPS.`

One of the first steps in hardening our system is updating and bringing the system `up-to-date`. We can do this with the following commands:

Update the System

VPS Hardening

```
[cry0l1t3@VPS ~]$ sudo apt update -y && sudo apt full-upgrade -y && sudo apt autoremove -y && sudo apt autoclean -
```

SSH Hardening

SSH is always installed on the VPS, giving us guaranteed access to the server in advance. Now we can change some of the settings in the configuration file `/etc/ssh/sshd_config` to enforce these security measures for our SSH server. In this file, we will comment out, change or add some lines. The entire list of possible settings that can be made for the SSH daemon can be found on the man page.

Install Fail2Ban

VPS Hardening

```
[cry0l1t3@VPS ~]$ sudo apt install fail2ban -y
```

Once we have installed it, we can find the configuration file at `/etc/fail2ban/jail.conf`. We should make a backup if we make a mistake somewhere and it does not work as it should.

Fail2Ban Config Backup

```
VPS Hardening

[cry0l1t3@VPS ~]$ sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.conf.bak
```

In this file, we will find the field commented out with `# [sshd]`. In most cases, we will find the first line commented out there. Otherwise, we add this and two more, as shown in the following example:

/etc/fail2ban/jail.conf

```
VPS Hardening

...SNIP...

# [sshd]
enabled = true
bantime = 4w
maxretry = 3
```

With this, we enable monitoring for the SSH server, set the `ban time` to four weeks, and allow a maximum of 3 `attempts`. The advantage of this is that once we have configured our `2FA` feature for SSH, fail2ban will ban the IP address that has entered the `verification code` incorrectly three times too. We should make the following configurations in the `/etc/ssh/sshd_config` file:

Editing OpenSSH Config

```
VPS Hardening

[cry0l1t3@VPS ~]$ sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.bak
[cry0l1t3@VPS ~]$ sudo vim /etc/ssh/sshd_config
```

Settings	**Description**
LogLevel VERBOSE	Gives the verbosity level that is used when logging messages from SSH daemon.
PermitRootLogin no	Specifies whether root can log in using SSH.
MaxAuthTries 3	Specifies the maximum number of authentication attempts permitted per connection.
MaxSessions 5	Specifies the maximum number of open shell, login, or subsystem (e.g., SFTP) sessions allowed per network connection.
HostbasedAuthentication no	Specifies whether rhosts or /etc/hosts.equiv authentication together with successful public key client host authentication is allowed (host-based authentication).
PermitEmptyPasswords no	When password authentication is allowed, it specifies whether the server allows login to accounts with empty password strings.
ChallengeResponseAuthentication yes	Specifies whether challenge-response authentication is allowed.
UsePAM yes	Specifies if PAM modules should be used for authentification.
X11Forwarding no	Specifies whether X11 forwarding is permitted.
PrintMotd no	Specifies whether SSH daemon should print /etc/motd when a user logs in interactively.

ClientAliveInterval 600	Sets a timeout interval in seconds, after which if no data has been received from the client, the SSH daemon will send a message through the encrypted channel to request a response from the client.
ClientAliveCountMax 0	Sets the number of client alive messages which may be sent without SSH daemon receiving any messages back from the client.
AllowUsers <username>	This keyword can be followed by a list of user name patterns, separated by spaces. If specified, login is allowed only for user names that match one of the patterns.
Protocol 2	Specifies the usage of the newer protocol which is more secure.
AuthenticationMethods publickey,keyboard-interactive	Specifies the authentication methods that must be successfully completed for a user to be granted access.
PasswordAuthentication no	Specifies whether password authentication is allowed.

2FA Authentication

With the configuration shown above, we have already taken essential steps to harden our SSH. Therefore, we can now go one step further and configure `2-factor authentication` (`2FA`). With this, we use a third-party software called Google Authenticator, which generates a six-digit code every 30 seconds that is needed to authenticate our identity. These six-digit codes represent a so-called `One-Time-Password` (OTP). `2FA` has proven itself an authentication method, not least because of its relatively high-security standard compared to the time required for implementation. Two different and independent authentication factors verify the identity of a person requesting access. We can find more information about `2FA` here.

We will use Google Authenticator as our authentication application on our Android or iOS smartphone. For this, we need to download and install the application from the Google/Apple Store. A guide on setting up Google Authenticator on our smartphone can be found here. To configure 2FA with Google Authenticator on our VPS, we need the Google-Authenticator PAM module. We can then install it and execute it to start configuring it as follows:

Installing Google-Authenticator PAM Module

VPS Hardening

```
[cry0l1t3@VPS ~]$ sudo apt install libpam-google-authenticator -y
[cry0l1t3@VPS ~]$ google-authenticator

Do you want authentication tokens to be time-based (y/n) y

Warning: pasting the following URL into your browser exposes the OTP secret to Google:
  https://www.google.com/chart?chs=200x200&chld=M|0&cht=qr&chl=otpauth://totp/cry0l1t3@parrot%3Fsecret%...SNIP...%

   [ ---- QR Code ---- ]

Your new secret key is: ***************
Enter code from app (-1 to skip):
```

If we follow these steps, then a `QR code` and a `secret key` will appear in our terminal, which we can then scan with the Google Authenticator app or enter the secret key there. Once we have scanned the QR code or entered the secret key, we will see the first `OTP` (six-digit number) on our smartphone. We enter this in our terminal to synchronize and authorize Google Authenticator on our smartphone and our VPS with Google.

The module will then generate several `emergency scratch codes` (`backup codes`), which we should save safely. These will be used in case we lose our smartphone. Should this happen, we can then log in with the `backup codes`.

Google-Authenticator Configuration

VPS Hardening

```
Enter code from app (-1 to skip): <Google-Auth Code>

Code confirmed
Your emergency scratch codes are:
  21323478
  43822347
  60232018
  73234726
  45456791

Do you want me to update your "/home/cry0l1t3/.google_authenticator" file? (y/n) y

Do you want to disallow multiple uses of the same authentication
token? This restricts you to one login about every 30s, but it increases
your chances to notice or even prevent man-in-the-middle attacks (y/n) y

By default, a new token is generated every 30 seconds by the mobile app.
In order to compensate for possible time-skew between the client and the server,
we allow an extra token before and after the current time. This allows for a
time skew of up to 30 seconds between authentication server and client. If you
experience problems with poor time synchronization, you can increase the window
from its default size of 3 permitted codes (one previous code, the current
code, the next code) to 17 permitted codes (the 8 previous codes, the current
code, and the 8 next codes). This will permit for a time skew of up to 4 minutes
between client and server.
Do you want to do so? (y/n) n

If the computer that you are logging into isn't hardened against brute-force
login attempts, you can enable rate-limiting for the authentication module.
By default, this limits attackers to no more than 3 login attempts every 30s.
Do you want to enable rate-limiting? (y/n) y
```

Next, we need to configure the PAM module for the SSH daemon. To do this, we first create a backup of the file and open the file with a text editor such as Vim.

2FA PAM Configuration

VPS Hardening

```
[cry0l1t3@VPS ~]$ sudo cp /etc/pam.d/sshd /etc/pam.d/sshd.bak
[cry0l1t3@VPS ~]$ sudo vim /etc/pam.d/sshd
```

We comment out the "`@include common-auth`" line by putting a "`#`" in front of it. Besides, we add two new lines at the end of the file, as follows:

/etc/pam.d/sshd

VPS Hardening

```
#@include common-auth

...SNIP...

auth required pam_google_authenticator.so
auth required pam_permit.so
```

Next, we need to adjust our settings in our SSH daemon to allow this authentication method. In this configuration file (`/etc/ssh/sshd_config`), we need to add two new lines at the end of the file as follows:

/etc/ssh/sshd_config

VPS Hardening

```
...SNIP...

AuthenticationMethods publickey,keyboard-interactive
PasswordAuthentication no
```

Finally, we have to restart the SSH server to apply the new configurations and settings.

Restart SSH Server

VPS Hardening

```
[cry0l1t3@VPS ~]$ sudo service ssh restart
```

Now we can test this and try to login to the SSH server with our SSH key and check if everything works as intended.

2FA SSH Login

VPS Hardening

```
FrogHunter95@htb[/htb]$ ssh cry0l1t3@VPS -i ~/.ssh/vps-ssh

Enter passphrase for key 'cry0l1t3': *************
Verification code: <Google-Auth Code>
```

Finally, we can transfer all our resources, scripts, notes, and other components to the VPS using SCP.

SCP Syntax

VPS Hardening

```
FrogHunter95@htb[/htb]$ scp -i <ssh-private-key> -r <directory to transfer> <username>@<IP/FQDN>:<path>
```

Resources Transfer

VPS Hardening

```
FrogHunter95@htb[/htb]$ scp -i ~/.ssh/vps-ssh -r ~/Pentesting cry0l1t3@VPS:~/

Enter passphrase for key 'cry0l1t3': *************
Verification code: <Google-Auth Code>
```

Closing Thoughts

We should now have an understanding of using common virtualization platforms such as VMWare Workstation/Player and VirtualBox and be comfortable with setting up VMs from scratch. After practicing the examples in this Module, we should also be comfortable setting up and hardening both a Windows and Linux attack machine for our penetration testing purposes. Finally, it is worth replicating the steps for standing up a VPS using a provider such as Vultr and practicing hardening it based on the steps in this section. It is important for us to be able to configure, harden, and maintain the systems that we use during our assessments.

Questions

Answer the question(s) below to complete this Section and earn cubes!

+ 10 What does the acronym Linux PAM stand for?

Pluggable Authentication Modules

Submit

Previous

+10 Streak pts

Finish

? Go to Questions

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left