Part 86
-
Protecting Internet-connected Raspberry Pi's

Version: 2024-05-16

Hardening Your Raspberry Pi

We are going to discuss some tips on hardening your Raspberry Pi to make it harder for hackers to abuse your internet-connected pi. Like anything else connected to the internet even if you follow every best practice for security there is a chance one day someone will get in. The goal is not to make a 100% unhackable device because they don't exist. Our goal is to make it difficult enough, it is not worth the time or effort of breaking in.

Updating Your Raspberry Pi

Before going any further one of the easiest ways to exploit a machine is to find a vulnerable service running on that system. New exploits are released into the wild every day, and when they are software makers (*should*) update their software so it is a good idea to always keep your packages updated. We will be using the built-in APT package manager to do this.

```
sudo apt update
sudo apt upgrade -y
sudo reboot
```

This commands will update the list of packages from our trusted repositories and check to make sure we are running the latest versions. Then, we are going to run the update process, if you exclude the -y (yes) flag you will be prompted to enter Y/N to verify the packages we want to update. I like to keep all my packages updated so I let the update run on all packages this way.
Finally, we will be rebooting the Raspberry Pi so all applied updates can take effect and we can make sure everything is the way it should be.

Changing the Default SSH Port

Almost everyone uses SSH if you are not you should just go ahead and disable it. If you do though, you may want to consider changing the default SSH port. Most of the attacks these days are fully automated and knowing this we also know they will follow a standard set of rules when attacking. They will check the most common ports looking for known exploitable or brute-forcible services so if we change our port we can disguise a nice chunk of our attack surface and help with hardening our Raspberry Pi.

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

After running the above command `nano` will open the config file and allow us to make one change inside this file. Look for the following line:

```
#Port 22
```

We are going to remove the pound sign (#) from the beginning of the line and change 22 to the port we want SSH to run on. I like to use high port numbers between 20,000 and 50,000 most port scanners will not scan this high without changing default options. So our new line should look something like this but feel free to pick your own port number:

```
Port 24863
```

After making this change you save and exit nano by pressing Ctrl + X and then Y (for yes). Finally, we need to restart SSH for the changes to take effect.

```
sudo systemctl restart ssh.service
```

Removing Default Pi User
Similar to the above SSH changes, the Pi username is one of the most commonly tried usernames when cracking passwords. We want to make it as difficult as we can for people to be able to guess our username and password combinations. Before removing the user we need to add a new, it's pretty straightforward.

```
sudo adduser username_here
```

We are going to want to give our new user account sudo privileges so we are still able to make system changes from our new account or we will be limited on what we are able to do.

```
sudo adduser username_here sudo
```

At this point, let's go ahead and restart our Pi and log in as the new user we created. We can restart using the following:

```
sudo reboot
```

After logging in as the new user we created we still need to remove the pi user.

```
sudo deluser -remove-home pi
```

This will remove the pi user and the home directory for it.

Final word
After doing these 3 relatively simple things you will have mitigated 95% of attacks.
Make sure you keep your system updated by running

```
sudo apt update
sudo apt upgrade -y
sudo reboot
```

on a regular base

Using IPTables on your Raspberry Pi

When exposing your device to the outside world you want to make sure you add a firewall this will keep nasty intruders out. This is not an "end-all" solution, but you want to limit access to ports people do not need access to. There are lots of solutions you could use as your firewall, most are built on top of Linux `iptables`. So we will just use `iptables` themselves.

```
sudo apt update
sudo apt -y upgrade
sudo apt -y install iptables-persistent --upgrade
```

This will install `iptables-persistent` and any dependencies. Next, we need to set up some basic rules, please make sure to set up rules for the services you are using or you will be blocked from them by the firewall.

```
sudo iptables -F
sudo iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP
sudo iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
sudo iptables -A INPUT -p tcp --tcp-flags ALL ALL -j DROP
sudo iptables -A INPUT -i lo -j ACCEPT
```

First please note all commands use `sudo`, you need higher privileges to add firewall rules. Using the above rules we block some basic attacks we will start adding rules to open specific ports we will use. So I will list the service and then the command to open the port.

Allow SSH connections

```
sudo iptables -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
```

If you only will allow an SSH connection from a specific IP-address eg: 192.168.12.123 then the command will like this

```
sudo iptables -A INPUT -p tcp -m tcp -s 192.168.12.123 --dport 22 -j ACCEPT
```

Allow HTTP server

```
sudo iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
```

Allow HTTPS server

```
sudo iptables -A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
```

SMTP server

```
sudo iptables -A INPUT -p tcp -m tcp --dport 25 -j ACCEPT
sudo iptables -A INPUT -p tcp -m tcp --dport 465 -j ACCEPT
```

POP3 server

```
sudo iptables -A INPUT -p tcp -m tcp --dport 110 -j ACCEPT
sudo iptables -A INPUT -p tcp -m tcp --dport 995 -j ACCEPT
```

IMAP server

```
sudo iptables -A INPUT -p tcp -m tcp --dport 143 -j ACCEPT
sudo iptables -A INPUT -p tcp -m tcp --dport 993 -j ACCEPT
```

Home Assistant

```
sudo iptables -A INPUT -p tcp -m tcp --dport 8123 -j ACCEPT
```

You may have other services you want to accept connections to, we can add more rules for any port with a simple command just replace [TYPE] with the type of connection TCP, or UDP. Replace [PORT] with the port you will be accepting connections on.

```
sudo iptables -A INPUT -p [TYPE] -m [TYPE] --dport [PORT] -j ACCEPT
```

Finally, we need to allow outgoing connections we will need our machine to be able to update, use wget or other outside connections.

```
sudo iptables -I INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
sudo iptables -P OUTPUT ACCEPT && sudo iptables -P INPUT DROP
```

So now we have all our rules added, but do we actually? Let's double-check by running the following:

```
sudo iptables -L -n
```

To make sure that all connections from or to a specific IP address, eg: 192.168.12.123, are accepted, insert these 2 rules:

```
sudo iptables -I INPUT  -p tcp -s 192.168.12.123 -j ACCEPT
sudo iptables -I OUTPUT -p tcp -d 192.168.12.123 -j ACCEPT
```

We can save the rules using:

```
sudo iptables-save  >/etc/iptables/rules.v4
sudo ip6tables-save >/etc/iptables/rules.v6
```

And we are all done! Our system is more secure than it was before and it didn't even take too long. Please note adding these rules is not a solution to solve all your problems. It just adds one more layer of security, finding open ports is half the battle when cracking a machine.

How To Protect SSH with fail2ban

Introduction

SSH is the de facto method of connecting to a cloud server. It is durable, and it is extensible — as new encryption standards are developed, they can be used to generate new SSH keys, ensuring that the core protocol remains secure. However, no protocol or software stack is totally foolproof, and SSH being so widely deployed across the internet means that it represents a very predictable *attack surface* or *attack vector* through which people can try to gain access.

Any service that is exposed to the network is a potential target in this way. If you review the logs for your SSH service running on any widely trafficked server, you will often see repeated, systematic login attempts that represent brute force attacks by users and bots alike. Although you can make some optimizations to your SSH service to reduce the chance of these attacks succeeding to near-zero, such as disabling password authentication in favor of SSH keys, they can still pose a minor, ongoing liability.

Large-scale production deployments for whom this liability is completely unacceptable will usually implement a VPN such as WireGuard in front of their SSH service, so that it is impossible to connect directly to the default SSH port 22 from the outside internet without additional software abstraction or gateways. These VPN solutions are widely trusted, but will add complexity, and can break some automations or other small software hooks.

Prior to or in addition to committing to a full VPN setup, you can implement a tool called fail2ban. fail2ban can significantly mitigate brute force attacks by creating rules that automatically alter your firewall configuration to ban specific IPs after a certain number of unsuccessful login attempts. This will allow your server to harden itself against these access attempts without intervention from you.

In this guide, you'll see how to install and use fail2ban on a Debian 12 server.

Step 1 — Installing fail2ban

fail2ban is available in Debian's software repositories. Begin by running the following commands as a non-root user to update your package listings and install fail2ban:

```
sudo apt update
sudo apt -y install iptables --upgrade
sudo apt -y install fail2ban --upgrade
```

fail2ban will automatically set up a background service after being installed. You can check its status by using the systemctl command:

```
sudo systemctl status fail2ban.service
```

Step 2 – Configuring fail2ban

During the installation process, fail2ban will generate a file called "`jail.conf`" as well as "`fail2ban.conf`".
We need to make a file, named "`jail.local`". fail2ban will automatically detect this file and load in its configuration for it.

```
sudo nano /etc/fail2ban/jail.local
```

Add the following lines

```
[DEFAULT]
backend = systemd

[sshd]
enabled = true
port = ssh
filter = sshd
banaction = iptables-multiport
bantime = -1
maxretry = 3
logpath = %(sshd_log)s
backend = %(sshd_backend)s
```

The first 2 lines are needed to prevent fail2ban not to start and generate an error

```
[1329]: ERROR   Failed during configuration: Have not found any log file for sshd jail
```

The next lines are set for the SSH daemon. The first lines in this section that we are adding enables fail2ban to process those rules for the specified port.
The next lines tells fail2ban that it needs to use the "`/etc/fail2ban/filter.d/sshd.conf`" file to filter connections to the SSH port.

In addition to being able to enable it and setting the filter, we can also change what fail2ban does when someone triggers the filters.
To set the ban action you can utilize the following line. In our example that we have below, we will be using the "`iptables-multiport`" ban action.
This action will ban the user that triggered the filter and restrict them from accessing any ports on the device.
You can find additional actions by checking out the "`/etc/fail2ban/action.d/`" folder, typically though you will want to block an attacker on all ports.

In addition to being able to set the ban action, you can also set the number of attempts a user gets before they are banned as well as how long that they should be banned for.
To do this we can utilize the following two values, we have set some example values that we will explain below.
The line above "`bantime = -1`", sets how long you want the user to be banned for. This value needs to be in seconds, for example, `1800` will ban the user for 30 minutes. If you want to ban the user indefinitely, you can set this value to -1 as we have in our example above.
The line "`maxretry = 3`", defines how many tries the user gets before the ban action is run. In our example, we set this to 3 meaning the user will have three chances before they are banned from accessing the device on all ports.

The line `logpath = %(sshd_log)s` lets us determine which logs to monitor. fail2ban works by getting information from SSH logs.

The line `backend = %(sshd_backend)s` specifies the backend used to get files modification.
When you are happy with your changes, go ahead and save the file by pressing `CTRL + X` then `Y` and finally `ENTER`.

You should now have the Raspberry Pi fail2ban up and running successfully. To get the fail2ban software to load up your changes on your Raspberry Pi you need to go ahead and enter the following command.

```
sudo systemctl restart fail2ban.service
```

Now check its status

```
sudo systemctl status fail2ban.service
```

Should you see this warning

```
[1348]: WARNING 'allowipv6' not defined in 'Definition'. Using default one: 'auto'
```

then you can prevent this by creating a `fail2ban.local` file and adding these lines

```
sudo nano /etc/fail2ban/fail2ban.local
```

add

```
[DEFAULT]
allowipv6 = auto
```

Save and restart fail2ban. Check its status again.

Step 3 — Testing the banning policies (Optional)

From another server, one that won't need to log into your fail2ban server in the future, you can test the rules by getting that second server banned. After logging into your second server, try to SSH into the fail2ban server. You can try to connect using a nonexistent name:

```
ssh blah@your_server
```

Enter random characters into the password prompt. Repeat this a few times. At some point, the error you're receiving should change from Permission denied to Connection refused. This signals that your second server has been banned from the fail2ban server.

On your fail2ban server, you can see the new rule by checking your iptables output. iptables is a command for interacting with low-level port and firewall rules on your server. If you followed DigitalOcean's guide to initial server setup, you will be using ufw to manage firewall rules at a higher level. Running iptables -S will show you all of the firewall rules that ufw already created:

```
sudo iptables -S
```

Output

```
-P INPUT DROP
-P FORWARD DROP
-P OUTPUT ACCEPT
-N f2b-sshd
-N ufw-after-forward
-N ufw-after-input
-N ufw-after-logging-forward
-N ufw-after-logging-input
-N ufw-after-logging-output
-N ufw-after-output
-N ufw-before-forward
-N ufw-before-input
-N ufw-before-logging-forward
-N ufw-before-logging-input
-N ufw-before-logging-output
…
```

If you pipe the output of `iptables -S` to grep to search within those rules for the string f2b, you can see the rules that have been added by fail2ban:

```
sudo iptables -S | grep f2b
```

Output

```
-N f2b-sshd
-A INPUT -p tcp -m multiport --dports 22 -j f2b-sshd
-A f2b-sshd -s 134.209.165.184/32 -j REJECT --reject-with icmp-port-unreachable
-A f2b-sshd -j RETURN
```

The line containing `REJECT --reject-with icmp-port-unreachable` will have been added by fail2ban and should reflect the IP address of your second server.

Apache Web Servers
You can protect your Apache web server using fail2ban as well. The setup is very similar to what we did for SSH. I will quickly go through an example for Apache below.
If you want to enable protection for Apache against bad bots, then you will need to open the jail local file.

```
sudo nano /etc/fail2ban/jail.local
```

Add the following

```
[apache-badbots]
enabled = true
port = http,https
filter = apache-badbots
bantime = 48h
maxretry = 1
logpath = %(apache_access_log)s
```

The filter name will typically be the same name as the module unless you're using a custom configuration file. So, `[apache-badbots]` will have a filter name of `apache-badbots`.
You can find all the filter configuration files in the following directory, use ls to list all the files.

```
ls /etc/fail2ban/filter.d/
```

Once you're done editing the "`jail.local`" file, save the file by pressing `CTRL + X` then `Y` and finally `ENTER`.

Lastly, remember to restart fail2ban on the Raspberry Pi whenever you make a change and check its status.

```
sudo systemctl restart fail2ban.service
sudo systemctl status fail2ban.service
```

For protecting other services, have a look in `jail.conf`. If you want to add more services, copy the base setting from `jail.conf` to `jail.local`, enable it in `jail.local` and add also its filter. Change `bantime` and `maxretries` as needed.

Important Note: Do not alter `jail.conf` nor `fail2ban.conf`. Changing the behavior of fail2ban is done in `jail.local` and `fail2ban.local`

Conclusion
You should now be able to configure some banning policies for your services. fail2ban is a useful way to protect any kind of service that uses authentication.

How to unban an IP properly with fail2ban
One way you can avoid this in the future is to configure your jail to include your IP address in the ignoreip value. Adding your IP address to that ignoreips attribute will allow you to run any nefarious command without banning yourself.

```
sudo nano /etc/fail2ban/jail.conf
```

Add

```
[DEFAULT]
ignoreip = 192.168.0.1/24 ::1
```

To ignore more than 1 range `ignopreip` takes space delimited CIDR blocks. CR's are ignored, but the next line has to start with a space, so you can break up a very long line as follows:

```
ignoreip = 0.0.0.0/8 10.0.0.0/8 23.135.225.0/24 23.151.160.0/24
 27.123.224.0/22 27.124.64.0/20 27.126.156.0/22 36.50.0.0/16
 43.225.128.0/22 43.227.184.0/22 43.228.104.0/22 43.228.164.0/22
 43.228.172.0/22 43.229.16.0/22 43.231.130.0/23  43.240.52.0/22
 43.240.232.0/22 23.114.97.241 8.8.8.8 1.1.1.1
```

A second methode with fail2ban v0.8.8 and later:

```
fail2ban-client set JAILNAMEHERE unbanip IPADDRESSHERE
```

The hard part is finding the right jail:

Use `iptables -L -n` to find the rule name… then use

```
fail2ban-client status | grep "Jail list" | sed -E 's/^[^:]+:[ \t]+//' | sed 's/,//g'
```

to get the actual jail names. The rule name and jail name may not be the same but it should be clear which one is related to which.

A third method is interactive. Type in

```
fail2ban-client -i
```

then in interactive mode type:

```
status sshd
```

you'll get:

```
Status for the jail: ssh
|- Filter
|  |- Currently failed: 0
|  |- Total failed: 6
|  `- File list:    /var/log/auth.log
`- Actions
   |- Currently banned: 1
   |- Total banned: 2
   `- Banned IP list:   203.113.167.162
```

then type in fail2ban interactive mode:

```
set sshd unbanip 203.113.167.162
```

you'll get:

```
203.113.167.162
```

it means no longer 203.113.167.162 in ban list.

Type `exit` to get out of the interactive fail2ban mode