Ubuntu Server 22.04 - Hardened System

Table of Contents

- Starting Fail2ban

Prerequisites

- Ubuntu Server 22.04 LTS installation media (ISO file)
- Virtual machine with at least :
  - 4GB RAM
  - 60GB storage
  - Network connectivity

Installation Procedure

Initial Setup

> **ⓘ Boot Media**
>
> Start your VM the Ubuntu Server 22.04 ISO mounted

- Configure language and keyboard layout
- Set up network settings
- Configure proxy if needed

Storage Configuration with Encryption

> **⚠ Critical with Encryption**
>
> The encryption passphrase cannot be recovered if lost. Document it securely but separately from the system.

1. At storage configuration, select `“Custom storage layout”`
2. Create a boot partition :

- Select free space → "Add GPT Partition"

```
Size : 2G
Format : ext4
```

```
Mount : /boot
```

3. For remaining space :

- Select free space → "Add GPT Partition"

```
Size : Leave blank
Format : Leave unformatted
```

4. Create a volume group (LVM)" :

- Select `"Select Create volume group (LVM)"`
- Set up a name for the LVM volume group
- Select the partition used for encryption (not the boot partition)
- Select "Create encrypted volume" and create a strong encryption passphrase

5. Create logical volumes :

Partition	Size	Min Size	Mount Point	Purpose	Security Benefits
/	13.33%	8 GB	/	Root filesystem	Basis for system
/usr	16.67%	10 GB	/usr	System applications and libraries	Can be mounted read-only in production
/var	16.67%	10 GB	/var	Variable data (logs, spools, etc.)	Contains logs critical for security analysis
/home	33.33%	20 GB	/home	User home directories	User isolation and quota managment
swap	13.33%	8 GB	N/A	Virtual memory	Memory management support

> **Partition Rationale**
>
> This simplified partition scheme isolates critical system components while allowing more space for each partition.

6. Review and confirm

User and Authentication Setup

- Set servers name
- Create admin user :

```
Full name: Admininistrator
Username: localadmin
Password: [strong password]
```

- Skip SSH key import (configured later)

> **Role-based Administrator**
>
> This creates a dedicated system administrator account with proper user attributes.

Installation and First Boot

1. Wait for completion
2. Remove installation media → reboot
3. Enter encryption passphrase when prompted

Post-Installation Security Configuration

Update System and Install vim

```
sudo apt update
sudo apt upgrade -y
sudo apt install vim
```

Configure sudo for localadmin

> **Principle of Least Privilege**
>
> We're ensuring the localadmin account has appropriate administrative capabilities while documenting each action.

Edit the sudoers file :

```
sudo vim /etc/sudoers
```

Add the following line under `“User privilege specification”` :

```
# Grant localadmin full sudo privileges
localadmin ALL=(ALL:ALL) ALL
```

Configure log for sudo

Edit the sudoers file :

```
sudo vim /etc/sudoers
```

Add the following line :

```
Defaults logfile=/var/log/sudo.log
```

Disable root Account

> ⚠ **Critical Security Measure**
>
> Disable the root account prevents direct root login, enhancing system security.

Edit `/etc/passwd` :

```
sudo vim /etc/passwd
```

Find the `root` line and change the shell to `/usr/sbin/nologin` :

```
root:x:0:0:root:/root:/usr/sbin/nologin
```

Edit `/etc/shadow` :

```
sudo vim /etc/shadow
```

Lock the root password by replacing `*` by `!` :

```
root:!:19977:0:99999:7:::
```

Secure Mount Options

Edit `/etc/fstab` :

```
sudo vim /etc/fstab
```

Add security options :

Partition	Option	Dump	Pass	Justification
/ (root)	defaults,nodev	0	1	• `nodev` : Device files should only exist in /dev
/usr	defaults,nodev	0	2	• `nodev` : No device files needed
/var	defaults,nodev,nosuid	0	2	• `nodev` : No device files needed • `nosuid` : Prevents privilege escalation • Allows execution for services
/home	defaults,nosuid,nodev,noexec	0	2	• `nodev,nosuid,noexec` : No legitimate need for these privileges
/boot	defaults,nodev,nosuid,noexec,ro	0	2	• `ro` : Protects bootloader and kernel • `nodev,nosuid,noexec` : No legitimate need for these privileges
/tmp	rw,nodev,nosuid,noexec	0	0	• `nodev,nosuid,noexec` : Maximum restrictions for temporary files
/proc	defaults,hidepid=2	0	0	• `hidepid=2` : Processes of other users are invisible

And add the following lines :

```
# hidepid=2 : Processes of other users are invisible
proc /proc proc hidepid=2 0 0
```

```
# nodev,nosuid,noexec : Maximum restrictions for temporary files
tmpfs /tmp tmpfs nosuid,nodev,noexec 0 0
```

> **System Protection**
>
> These mount options significantly enhance system security by compartmentalizing privileges and execution permissions across the filesystem hierarchy.

SSH Server: Setup, Configuration and Hardening

Instead of using passwords to log in to your server, SSH keys provide enhanced security. This system works through a matched pair of keys:

- Your server holds the public key
- You keep the private key secure on your computer

When connecting, your server verifies your identity by checking if your private key matches the stored public key - no password needed. This prevents unauthorized access even if someone attempts password-guessing attacks.

Basic Setup

Install Required Packages

First, update your system and install the necessary packages :

```
sudo apt install openssh-server libpam-google-authenticator qrencode
```

Configure SSH Service

Start and enable the SSH service to ensure it runs at system boot :

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

Hardening SSH Configuration

Modify SSH Port

Changing the default SSH port helps reduce automated attack attempts.

Edit the SSH configuration file :

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

Find the `Port` line and modify it (uncomment if necessary) :

```
Port <port>
```

> **Security Note :**
>
> Choose a port number between `1024` and `65535` that isn't used by other services.

Disable Root Login

Preventing direct root login via SSH is a critical security measure.

In `/etc/ssh/sshd_config` , find and modify :

```
PermitRootLogin no
```

Apply Changes

Restart the SSH service to apply all changes :

```
sudo systemctl restart ssh
```

Connect to SSH

On client machine :

```
ssh -p <port> <username>@<server-ip>
```

RSA Key Authentication

Generate SSH Key Pair on Server

On your server machine (not the client) generate an RSA key pair :

```
ssh-keygen -t rsa -b 4096
```

Then, add the public key to `authorized_keys` :

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Transfer Private Key to Client

Copy your private key to the client :

Windows (with PowerShell):

```
scp -P <port> <username>@<server-ip>:~/.ssh/id_rsa ./.ssh
```

Enable Authorized Keys File

In `/etc/ssh/sshd_config` , find and uncheck :

```
AuthorizedKeysFile .ssh/authorized_keys
```

Apply Changes

Restart the SSH service to apply all changes :

```
sudo systemctl restart ssh
```

> ⚠ **Important :**
>
> Do not close your current SSH session until you've verified the new configuration works.

Two-Factor Authentication (2FA)

Install and Configure Google Authenticator

Set up correct time zone

```
sudo timedatectl set-timezone <continent>/<capital>
```

1. On the server, run the configuration tool :

```
google-authenticator
```

2. A QR code will be displayed in the terminal. Scan it with an authenticator app like Google Authenticator, Authy, or Microsoft Authenticator.
3. Answer the questions when prompted :
   - `"Do you want authentication tokens to be time-based?"` → `y` (Uses time-based OTP tokens)
   - `"Do you want to disallow multiple uses...?"` → `y` (Prevents replay attacks)
   - `"Do you want to increase the original window...?"` → `n` (Maintains tight time synchronization requirements)
   - `"Do you want to enable rate-limiting?"` → `y` (Prevents brute force attacks)
4. Save the emergency scratch codes in a secure location.

Configure PAM for SSH 2FA

1. Edit the PAM SSH configuration file :

```
sudo vim /etc/pam.d/sshd
```

2. Add the following line at the beginning :

```
auth required pam_google_authenticator.so
```

Configure SSH to Use PAM

Edit `/etc/ssh/sshd_config` to enable PAM, challenge-response authentication and keyboard interactive authentication:

```
PasswordAuthentication no
KbdInteractiveAuthentication yes
UsePAM yes
ChallengeResponseAuthentication yes
AuthenticationMethods publickey,keyboard-interactive
```

Apply Changes

Restart the SSH service to apply all changes :

```
sudo systemctl restart ssh
```

> ⚠ **Important :**
>
> Do not close your current SSH session until you've verified the new configuration works.

Access Control

`LoginGraceTime` determines the authentication window for users.

`MaxAuthTries` controls how many authentication attempts are permitted before blocking a connection.

`MaxSessions` defines how many simultaneous SSH connections a single user can maintain.

These are the recommended values for these three variables:

```
LoginGraceTime 20
MaxAuthTries 3
MaxSessions 5
```

> ⓘ **These 3 lines are commented by default, make sure to uncomment them!**

Prevent Empty Passwords

By default, SSH prevents users with empty passwords from accessing the server, but it's always good to double-check.

Look for the `PermitEmptyPasswords` variable and make sure it's set to `no` like this:

```
#PermitEmptyPasswords no
```

It's commented out by default. If it's already set to `no`, you can leave it as is.

Idle Timeout Interval

Unattended SSH sessions present a significant security vulnerability. When a user maintains an open terminal without activity, this creates an idle connection that could potentially be exploited.

You can automatically disconnect inactive sessions by configuring two key parameters:

`ClientAliveInterval` sets how frequently (in seconds) the server sends a verification signal to the client.
`ClientAliveCountMax` establishes how many consecutive verification attempts can fail before the server terminates the connection.

```
ClientAliveInterval 60
ClientAliveCountMax 3
```

X11 Forwarding

By default, SSH can forward X11 sessions, which can be useful for certain graphical applications.

Unless necessary, disable X11 forwarding in SSH, as the X11 protocol isn't security-oriented.

Find the `X11Forwarding` variable and set it to `no` like this:

```
X11Forwarding no
```

Disabling non-essential features like X11 forwarding in SSH can greatly strengthen your server's security.

Agent and TCP Forwarding

SSH agent forwarding lets you use your SSH keys to move from one server to another without keeping keys on the first server.

SSH TCP forwarding, also known as port forwarding, lets you redirect data between your local and remote machine.

Disable these two features if you don't need them to minimize the attack surface.

Find the `AllowAgentForwarding` and `AllowTcpForwarding` variables and set them to `no` like this:

```
AllowAgentForwarding no
AllowTcpForwarding no
```

Password Authentication

Accessing your server usually involves using a password with SSH. But a more secure method is SSH key authentication.

This uses two keys: a public key on your server and a private key you keep. You get access when the server verifies you have the private key.

This method is safer than just using a password, as it ensures only someone with the private key can enter the server.

Find the `PasswordAuthentication` variable and set its value to `no` like this:

```
PasswordAuthentication no
```

> ⓘ **By doing this, you ensure that the server can only be accessed using SSH keys.**

> ⚠ **Always remember to keep your private key secure.**

Reloading SSH

After you've made your changes, be sure to reload the SSH service to apply them.

Before that, you should verify the configuration for errors. You can do this using the following command:

```
sudo sshd -t
```

Use this command to restart SSH:

```
sudo systemctl restart ssh
```

Once the service is reloaded, your changes will be in effect.

> ⓘ **You can also use the `sudo sshd -T` command to list all variables with their values, allowing you to check whether the changes have taken effect or not.**

Automating Security Updates on Linux Servers

Installing Unattended Upgrades

```
sudo apt install unattended-upgrades
```

Now, run :

```
sudo dpkg-reconfigure unattended-upgrades
```

A pop-up window will appear, asking you if you want to automatically download and install stable updates.

Choose `Yes` and press the `ENTER` key.

Changes made

When you do this, Unattended Upgrades changed the value from `0` to `1` in the `/etc/apt/apt.conf.d/20auto-upgrades` file:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";
```

Configuration

Open `/etc/apt/apt.conf.d/50unattended-upgrades` :

```
sudo vim /etc/apt/apt.conf.d/50unattended-upgrades
```

Information

If you want it to handle non-security updates and update other installed packages, you can uncomment the `${distro_id}:${distro_codename}-updates` line.

To control if Unattended Upgrades should reboot your server automatically, look for the line `Unattended-Upgrade::Automatic-Reboot` in the configuration file.

Set this to `"false"` to prevent automatic reboots after updates.

If you prefer automatic reboots, change it to `"true"` .

Additionally, you can schedule a specific time for these reboots.

For this, find the line `Unattended-Upgrade::Automatic-Reboot-Time` and set it to your desired time, like `"04:00"` for a reboot at 4 AM.

Testing the Setup

```
sudo unattended-upgrade -d
```

Harden network with sysctl settings

Prevent source routing of incoming packets and log malformed IP's.

Open and edit `/etc/sysctl.conf` :

```
sudo vim /etc/sysctl.conf
```

Add or uncomment the following lines :

```
# Reverse Path Filtering (Prevent Spoofing Attacks)
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1

# Block SYN attacks
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.tcp_synack_retries = 3

# Ignore ICMP redirects
net.ipv4.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
net.ipv4.conf.all.secure_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0

# Disable Source Packet Routing
net.ipv4.conf.all.accept_source_route = 0
net.ipv6.conf.all.accept_source_route = 0
```

```
net.ipv4.conf.default.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0

# Disable Packet Forwarding (unless server is functionning as router or VPN)
net.ipv4.ip_forward = 0
net.ipv4.conf.all.forwarding = 0
net.ipv6.conf.all.forwarding = 0
net.ipv4.conf.default.forwarding = 0
net.ipv6.conf.default.forwarding = 0

# Protect TCP Connections (TIME-WAIT State)
net.ipv4.tcp_rfc1337 = 1

# Harden the BPF JIT Compiler
net.core.bpf_jit_harden = 2
kernel.unprivileged_bpf_disabled = 1

# Restrict Core Dumps
kernel.core_pattern = |/bin/false
fs.suid_dumpable = 0

# Disable Magic Keys
kernel.sysrq = 0

# Restrict Access to Kernel Logs
kernel.dmesg_restrict = 1

# Restrict ptrace Access
kernel.yama.ptrace_scope = 3

# Restrict User Namespaces
kernel.unprivileged_userns_clone = 0

# Control Swapping
vm.swappiness = 1

# File Creation Restrictions
fs.protected_regular = 2
fs.protected_fifos = 2
fs.protected_hardlinks = 1
fs.protected_symlinks = 1

# Address Space Layout Randomization (ASLR)
kernel.randomize_va_space = 2
```

Reload `sysctl` :

```
sudo sysctl -p
```

Setting Up a Firewall using UFW

Setting up a firewall is essential for securing your server, and UFW makes this process straightforward and user-friendly.

Installing UFW:

On Debian-based distribution, like Ubuntu, UFW often comes pre-packaged, but you can check and install it using theses commands:

```
# To check if UFW is installed
sudo ufw status

# If it's already installed disable it
sudo ufw disable
sudo ufw reset

# To install it
sudo apt update && sudo apt install ufw
```

```
--You can re-enable it once you have added all the rules and finished configuring it.
```

UFW's Default Policy

By default, UFW takes a secure approach by blocking all incoming traffic while allowing outgoing traffic from our server. This means our server can communicate externally, but it remains inaccessible to others.

Since there is no issue with our server reaching the outside world, there is no need to make any changes to that aspect.

However, to enable incoming traffic, it’s essential to selectively open only the required ports and authorize traffic through them.

You can find the default policy defined in the `/etc/default/ufw` file:

```
DEFAULT_INPUT_POLICY="DROP"
DEFAULT_OUTPUT_POLICY="ACCEPT"
```

ⓘ **As you can see, the default policy for incoming traffic is set to `DROP`, while the default policy for outgoing traffic is set to `ACCEPT`.**

If UFW is enabled, you can also review the default policy using the following command:

```
sudo ufw status verbose
```

You can modify this default behavior of UFW either by directly editing the file or by using these two commands:

```
sudo ufw default <policy> incoming
sudo ufw default <policy> outgoing
```

⚠ **Replace `<policy>` with either `deny`, `allow` or `reject`**

`deny` corresponds to DROP
`allow` corresponds to ACCEPT
`reject` corresponds to REJECT.

ⓘ **What are these options? :**

Both DROP and REJECT policies prevent traffic from passing through the firewall, but they differ in their response messages.

With DROP, the traffic is silently discarded without any acknowledgment sent to the source. It neither forwards the packet nor responds to it.

On the other hand, REJECT sends an error message back to the source, signaling a connection failure.

UFW's Configuration Files

You can find the UFW configuration file in `/etc/default/ufw` and examined the default policy of UFW, but there are other settings you might need to know about.

In this file, you will need to change only one option, the `'IPT_SYSCTL file path'`:

```
# Since we harder directly the linux kernel, change the IPT_SYSCTL file path
IPT_SYSCTL=/etc/sysctl.conf
```

This is for our configuration only

The only change we make in this file is to the `IPT_SYSCTL` variable. There's another file we'll discuss next, which is `/etc/ufw/sysctl.conf`. UFW uses this file to tweak certain kernel parameters.
However, the original file for changing kernel parameters is `/etc/sysctl.conf`.

Why this option? :

While UFW uses its own version for this purpose, we prefer not to do that. When we harden the Harden network with sysctl settings we make our changes to kernel parameters directly in the `/etc/sysctl.conf` file.
That's why we need to change that line, if you did not harder the kernel, do not change it. This way, I avoid dealing with two files and can maintain a clearer overview of the changes in a single file.

To incorporate the default changes that UFW makes, I can simply add these to the end of the `/etc/sysctl.conf` file:

```
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_ignore_bogus_error_responses = 1
net.ipv4.icmp_echo_ignore_all = 0
net.ipv4.conf.all.log_martians = 0
net.ipv4.conf.default.log_martians = 0
```

Now, simply reboot the server for the changes to take effect. There's no need to enable UFW for the changes to apply, since we are using the original `sysctl.conf` file.

The last file we need to review is the `/etc/ufw/ufw.conf` file, which contains just two variables:

```
ENABLED=no
LOGLEVEL=low
```

> **Why this option? :**
>
> The first variable controls whether UFW is enabled or disabled. There's no need to change it manually, as enabling or disabling UFW from the command line will automatically update this value.
> The second variable controls the log level of UFW. It can be set to `off`, `low`, `medium`, `high`, or `full`, depending on how much logging detail you want.

You can also change the logging level directly from the command line using the following command:

```
sudo ufw logging <logging_level>
```

```
->This will automatically update the value of the `LOGLEVEL` variable.
```

UFW's Rules Files

In the `/etc/ufw/` directory, you'll find files with the `.rules` extension(some example):

```
after6.rules  after.rules  before6.rules  before.rules  user6.rules
user.rules
```

These files control how UFW manages incoming, outgoing, and forwarded traffic. Files with the number `6` handle IPv6 traffic, while files without it handle IPv4 traffic.

> **Important**
>
> It's important to note that you should not modify the `user.rules` or `user6.rules` files directly, as any changes could be overwritten by UFW. Rules added by the user from the command line are saved to these files, which is why they are called `user.rules` files.
> You are free to add custom rules only to the `before.rules` or `after.rules` files.

> **Important**

The order in which UFW processes firewall rules is as follows: `before.rules` first, `user.rules` next, and `after.rules` last.
Understanding the order in which UFW processes these files and their respective roles is essential for effectively managing your firewall.

Basic User-Defined Rules

In the following, we'll cover the basic firewall rules that can be added from the command line.

UFW offers a set of commands for managing firewall rules directly, allowing you to quickly specify which services or ports are allowed or denied.

While these rules are designed for basic network access control and aren't intended for advanced use cases, they are perfect for setting up a firewall swiftly and efficiently.

Tips

To review the rules you've added when the firewall is disabled, use the command `sudo ufw show added`, as `sudo ufw status` won't display the rules in that case.

Remember

Every rule you add from the command line is saved to the `user.rules` file, so feel free to check it as you add rules to understand how UFW translates the commands into the file.

Allowing and Denying Traffic

The core functionality of UFW is to allow or deny network traffic.

To allow or deny traffic for specific ports, you use the `allow` or `deny` rules, respectively.

To allow incoming traffic on port 22 (SSH):

```
sudo ufw allow 22
```

You can also specify a protocol (TCP or UDP):

```
sudo ufw allow 22/tcp
```

To deny traffic on port 80(HTPP):

```
sudo ufw deny 80
```

If a range of ports is required, such as 5000-6000, use the following:

```
sudo ufw allow 5000:6000/tcp
```

Similarly, to deny traffic for the same range:

```
sudo ufw deny 5000:6000/tcp
```

You can also allow or deny traffic based on a service's name instead of specifying a port number. For example, to allow SSH traffic by using the service name, you can use:

```
sudo ufw allow ssh
```

UFW will then automatically determine the correct port (port 22 for SSH) and the associated protocol (TCP) to apply the rule. This approach simplifies rule management, especially when dealing with well-known services.

> ⚠ **Caution**
>
> In our case, we have changed the port of our ssh, so this command do not work.

Application Profiles

Applications (software or services installed) can register their profiles with UFW upon installation, enabling UFW to manage them by name.

To view the available profiles, you can use the following command:

```
sudo ufw app list
```

For instance, to allow traffic on port 443 (HTTPS), you can use the following commands:

```
sudo ufw allow "NGINX HTTPS"
sudo ufw allow "Apache Secure"
```

> **Tips**
>
> If you're curious about the origins of these profiles, check the `/etc/ufw/applications.d/` directory.

The `Limit` Rule

The `limit` rule in UFW helps protect against brute-force attacks by restricting the number of connection attempts an IP can make in a short time.

For example, when securing SSH, this rule lets legitimate users connect but temporarily blocks any IP that makes too many failed attempts.

By default, the rule allows only 6 connections from the same IP within 30 seconds. If the limit is exceeded, the IP is blocked temporarily, which reduces the risk of brute-force attacks.

To apply this rule for SSH traffic:

```
sudo ufw limit 22
```

```
->This command enables SSH access while protecting the server from excessive connection attempts.
```

Access Control by IP or Subnet

UFW lets you control access based on IP addresses or subnets, which is useful for limiting access to specific clients or denying access from specific sources.

To allow all traffic from a specific IP to any port:

```
sudo ufw allow from 192.168.1.100
```

To allow SSH traffic only from a specific IP:

```
sudo ufw allow from 192.168.1.100 to any port 22
```

To allow traffic from a subnet:

```
sudo ufw allow from 192.168.1.0/24
```

To deny traffic from a specific IP address:

```
sudo ufw deny from 203.0.113.50
```

In some cases, you might want to specify not just the IP or subnet but also the protocol (TCP or UDP).

For example, to allow only TCP traffic from an IP address to port 80, you can specify the protocol as follows:

```
sudo ufw allow from 192.168.1.100 to any port 80 proto tcp
```

By specifying protocols in your access control rules, you gain more control over the traffic flow, ensuring that only the desired traffic (TCP or UDP) is allowed from specific sources or networks.

Enabling and Checking Status

Before activating our firewall, it’s crucial to review the rules we’ve added so far to prevent any unexpected behavior.

> **Tips**
>
> As I mentioned earlier, if the firewall is disabled, we can't use the `sudo ufw status` command to view our rules.
> Instead, we use the `sudo ufw show added` command. This command will list all the rules we have added.
> *Always add the rules, review them, and then proceed to enable the firewall.*

To enable UFW and apply the rules you’ve configured, use the following command:

```
sudo ufw enable
```

Now, you can check the status of UFW and your current ruleset using the `sudo ufw status` command.

For a more detailed view:

```
sudo ufw status verbose
```

> **Tips**

If you experience any issues, disable UFW using `sudo ufw disable` and review your rules again.
If you need to reset UFW to its default state (removing all rules), you can use the `sudo ufw reset` command.

Deleting Rules

If, for some reason, you want to delete a rule you have added, you can use the `sudo ufw delete` command followed by the rule itself like this:

```
sudo ufw delete deny from 111.111.111.111 to any port 80 proto tcp
sudo ufw delete allow 80
```

There is another easier way to delete rules, but it requires the firewall to be enabled. This method involves using the rule number.

Once the firewall is enabled, you can use the `sudo ufw status numbered` command to obtain a list of your rules and their corresponding numbers, like this:

```
To                         Action      From
--                         ------      ----
[ 1] 22/tcp                ALLOW IN    Anywhere
[ 2] 22/tcp (v6)           ALLOW IN    Anywhere (v6)
```

Now, to delete a rule, you can simply use the rule number:

```
sudo ufw delete 1
```

```
->This is a much simpler method.
```

Best Practices

The first step before enabling a firewall is to allow SSH traffic to ensure access to the server. If you enable the firewall before adding this rule, you risk losing access to your server.

We showed you how to use the `allow` or `limit` rules. Using these rules will permit any IP address to access port 22, which means our SSH port is open to everyone. This is something we avoid on a production server.

If we have a static IP from which we can access the server, we restrict the SSH port to that IP. This provides an extra layer of security and reduces the risk of unauthorized access.

Even if you've generated an SSH key pair, implemented key authentication, and created a non-root user, hackers could still attempt to breach your server.

> **Tips**
>
> Only restrict the SSH port to your IP if it's static – such as when using a VPN service or if your ISP has assigned you a static IP.

If you have a static IP, use the following command:

```
sudo ufw allow from <YOUR_IP> proto tcp to any port 22
```

Now, the IP specified in the command is the only one that can access the server.

> **Caution**
>
> When you restrict SSH access to a single IP, Fail2Ban becomes irrelevant since there are no IPs to block. However, I still recommend keeping Fail2ban installed and enabled.

Advanced Firewall Rules

Up until now, we've focused on basic user-defined rules that you can easily manage from the command line.

Now, we'll introduce the idea of advanced rules, which allow you to control traffic at a deeper level by configuring UFW's `/etc/ufw/before.rules` files.

These advanced rules let you filter traffic before it reaches your server's services and before the firewall applies its standard rules.

Advanced rules are incredibly powerful and can be tailored to specific use cases, offering finer control over your network's security and performance.

And as we mentioned earlier, all of UFW's rule files primarily use the iptables syntax.

Structure of `before.rules` Files

The `before.rules` file begins with a declaration of the `*filter` table and defines several custom chains:

```
*filter
:ufw-before-input - [0:0]
:ufw-before-output - [0:0]
:ufw-before-forward - [0:0]
:ufw-not-local - [0:0]
```

ⓘ Why theses option? :

- `:ufw-before-input` : Processes incoming packets.
- `:ufw-before-forward` : Handles packets forwarded through the server.
- `:ufw-not-local` : Deals with packets that are not addressed to or from the local system.

It includes several default rules to handle fundamental network operations and improve security. These rules cover a variety of scenarios, such as allowing all traffic on loopback interfaces or dropping invalid packets. **They are applied by default once you enable the firewall**.
The file ends with the `COMMIT` line, signaling the completion of the rules.

The structure in `before6.rules` for IPv6 is nearly identical to that of `before.rules` for IPv4.

The main difference is that the chains in `before6.rules` all have the number `6` added to their names, like this:

```
*filter
:ufw6-before-input - [0:0]
:ufw6-before-output - [0:0]
:ufw6-before-forward - [0:0]
```

However, unlike `before.rules`, `before6.rules` does not include a `ufw6-not-local` chain.

It includes some of the default rules that `before.rules` has, but it also contains additional rules that are applied specifically to IPv6 traffic.

The file also ends with the `COMMIT` line, signaling the completion of the rules.

Use Case

As we mentioned earlier, these files take priority, meaning they are executed first.

This allows us to define rules that will take effect before traffic reaches anything running on the server and before it travels further through the firewall.

You can, for example, implement a solution to block SYN flood attacks by rate-limiting the number of SYN packets allowed through ports 80 and 443, and block IPs exceeding these limits. This is something that cannot be done using basic user-defined rules from the command line.

While you can use the `limit` rule for ports 80 and 443, it's not ideal for a web server since the limits may not be well-suited to handle the typical traffic patterns of a web server.

Using more advanced rules in `before.rules` allows you to fine-tune the firewall for specific use cases like this.

> ⓘ **What's SYN flood attacks?:**
>
> SYN flood attacks are a common and dangerous type of attack that can overwhelm a server by sending an excessive number of connection requests, ultimately disrupting legitimate traffic and potentially causing the server to go down.

Example of Advanced Rules

If you examine the contents of the `before.rules` file, you will notice these two rules:

```
-A ufw-before-input -m conntrack --ctstate INVALID -j ufw-logging-deny
-A ufw-before-input -m conntrack --ctstate INVALID -j DROP
```

> ⓘ **Why this option? :**
>
> These two rules are designed to log and block any invalid packets, and they are added by default by UFW to the `ufw-before-input` chain, which filters incoming traffic before it reaches the server, ensuring that only legitimate connections are allowed.

> ⓘ **What is Invalid Packet?:**
>
> Hackers use tools to create TCP packets with unusual, weird flag combinations, known as **Invalid Packets**, capable of causing significant harm.
> UFW blocks these invalid packets by default, but there are still instances where it may overlook and fail to block.

UFW uses the `conntrack` module (short for connection tracking) to monitor connections and identify those with `INVALID` connection states. While these rules are effective, we can make

them even better.

To further enhance the security of our server, we could add two additional rules to log and block any new connections that don't have only the SYN flag set.

Add the following two rules below the ones added by default by UFW:

```
-A ufw-before-input -p tcp -m tcp ! --tcp-flags FIN,SYN,RST,ACK SYN -m
conntrack --ctstate NEW -j ufw-logging-deny
-A ufw-before-input -p tcp -m tcp ! --tcp-flags FIN,SYN,RST,ACK SYN -m
conntrack --ctstate NEW -j DROP
```

Don't forget to add them to the `before6.rules` file as well:

```
-A ufw6-before-input -p tcp -m tcp ! --tcp-flags FIN,SYN,RST,ACK SYN -m
conntrack --ctstate NEW -j ufw6-logging-deny
-A ufw6-before-input -p tcp -m tcp ! --tcp-flags FIN,SYN,RST,ACK SYN -m
conntrack --ctstate NEW -j DROP
```

> ⓘ **Why this option? :**
>
> -The first rule drops any packet that's considered `INVALID` by the `conntrack` module.
> -The second rule blocks TCP packets that are flagged as `NEW` (indicating new connection attempts) but don't have the SYN flag set alone.

Now, reload UFW if it is already enabled:

```
sudo ufw reload
```

These additional rules further enhance the firewall's ability to filter out potentially malicious packets and protect your server from unwanted connection attempts.

Installing and configuring Fail2Ban

Fail2ban is available in Ubuntu's repositories.

To install it, use this command:

```
sudo apt install fail2ban
```

Pre-Configuration

Fail2ban's configuration files are located in the `/etc/fail2ban/` directory.

If you list the contents of this directory, you will find two important configuration files:

- The `fail2ban.conf` file contains Fail2Ban's global settings, which I don't recommend modifying.
- The `jail.conf` file contains jails, filters with actions.

Fail2Ban recommends creating two local copies of these configuration files for us to modify.

Use the following commands to create a local copy of these two files:

```
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
sudo cp /etc/fail2ban/fail2ban.conf /etc/fail2ban/fail2ban.local
```

✓ **Now you can safely modify Fail2Ban's configuration.**

Configuration

Open the `jail.local` file with your preferred editor and examine its settings.

Under the `[DEFAULT]` section, there are some variables that you may want to modify.

```
bantime
```

The `bantime` variable sets the duration for which an IP will be blocked from accessing the server after failing to authenticate correctly.

ⓘ **By default, this is set to 10 minutes.**

```
findtime
maxretry
```

The `maxretry` variable defines the number of authentication attempts an IP is allowed to make within a time period defined by `findtime` before being blocked.

With the default settings, Fail2Ban will block an IP that unsuccessfully attempts to access the server more than 5 times within a 10-minute interval.

```
#ignoreip = 127.0.0.1/8 ::1
```

The `ignoreip` variable contains a list of IP addresses, CIDR masks, or DNS hosts that Fail2ban won't block.

> ⓘ **By default, this variable is commented out.**

Email Alerts

If you want to receive email alerts whenever Fail2ban blocks an IP, you should adjust these two variables inside the `jail.local` file:

```
destemail
sender
```

The `destemail` variable defines the email address to which the alerts should be sent.

The `sender` variable defines the email address from which the alerts will be sent.

The sender variable should look like this:

```
sender = root@example.com
```

Lastly, there is the `mta` variable, which specifies the mail agent that will be used to send the emails.

> ⓘ **By default, Fail2ban uses Sendmail as its mail agent. You can change this with the `mta` variable.**

Default Action

If you scroll down a bit inside the `jail.local` file, you'll see the `action` variable:

```
action = %(action_)s
```

This variable dictates the action Fail2ban should take when blocking an IP address.

The default action is to add a firewall rule that rejects traffic from the IP address, removing it after the specified `bantime` elapses.

Above the action variable, you'll find various actions you can switch between :

- `action_mw` sends an email when taking action
- `action_mwl` sends an email and includes logging
- `action_cf_mwl` does all of the above, plus sends an update to the Cloudflare API associated with your account to ban the attacker there as well

> **Pick the one that suits you the most.**

Individual Jails

Now, it is time to examine the service-specific sections, also known as individual jails, such as the `[sshd]` jail, which protects our server from unauthorized access attempts.

Each of these jails needs to be individually enabled by adding an `enabled = true` line under the header, along with their other settings.

By default, only the `[sshd]` jail is enabled, and all others are disabled.

Scroll down the `jail.conf` file until you find the `[sshd]` jail, which should look similar to this:

```
port    = ssh
logpath = %(sshd_log)s
backend = %(sshd_backend)s
```

If you've changed the SSH port, ensure to update the value of the `port` variable accordingly.

You can include variables defined in the `[DEFAULT]` section, such as the `bantime`, `maxretry`, and `findtime` variables, which will only apply to this jail.

If you scroll down further, you'll find other jails that are disabled, such as the `[nginx-http-auth]` or `[apache-auth]` jails.

Starting Fail2ban

Since the `[sshd]` jail, which protects SSH, is enabled, we can proceed to start and enable the `fail2ban` service if it is disabled, depending on the version of Ubuntu you are using.

Use the following commands to start and enable Fail2ban:

```
sudo systemctl start fail2ban
sudo systemctl enable fail2ban
```

You can use the `fail2ban-client` command to check the active jails:

```
sudo fail2ban-client status
```

Output:

```
Status
|- Number of jail:  1
`- Jail list:   sshd
```

To view the status and information regarding a specific jail like the `sshd` jail, you can use the following command:

```
sudo fail2ban-client status sshd
```

Output:

```
Status for the jail: sshd
|- Filter
|  |- Currently failed: 5
|  |- Total failed: 21
|  `- File list:    /var/log/auth.log
`- Actions
   |- Currently banned: 1
   |- Total banned: 2
   `- Banned IP list:   218.92.0.29
```

> ⚠ **If you have disabled password authentication for SSH, you may notice zero failed attempts.**

> ⚠ **Caution**
>
> For Ubuntu-Server 22.4 you need to download r6log to make Fail2Ban work!
> Then you need to add "`authpriv.* /var/log/auth.log`" in the `/etc/rsyslog.conf`