BUILDING A SECURE WORDPRESS SERVER (LAMP) WITH CENTOS 7

Ray Heffer
Table of Contents

Introduction 5
  Assumptions ........................................................................................................... 5

Security Primer 6
  Do I really have to use SELinux? ........................................................................ 6

Managing Services with systemd 7
  systemd ................................................................................................................... 8
  Let’s Try It ............................................................................................................. 11

Understanding IPtables 12
  firewall ................................................................................................................... 13
  Nftables ................................................................................................................ 14
  Getting Started with IPtables .............................................................................. 14
  What are Chains? .................................................................................................. 15
  Flushing the Rules ............................................................................................... 16
  Creating Our First IPtables Rules ..................................................................... 17
  Logging .................................................................................................................. 18
  Set policy to drop all other traffic ............................................................... 19
  Recap What You Have Learned So Far .......................................................... 19
  Saving and Restarting IPtables ........................................................................ 20
  Advanced SSH Security .................................................................................. 20
  Limiting Other Attack Vectors ......................................................................... 25
  Host Access (TCP_WRAPPERS) ........................................................................ 26
  Blocking a IP Addresses with IPtables .......................................................... 27
  Common Firewall Rules for Web Hosts .......................................................... 27
  Final IPtables Rules ......................................................................................... 29
  Monitoring Logs .................................................................................................. 32

Stage 1: Deploying a new virtual private server (VPS) 33
  Securing Access .................................................................................................. 33
  Configuring SSH Key Based Authentication .................................................. 34
  Installing Core Packages .................................................................................. 36
  Basic Server Configuration .............................................................................. 38

Stage 2: IPtables Web Server Configuration 40
  Installing Fail2Ban ............................................................................................. 40

Stage 3: MariaDB (MySQL) 41

Stage 4: Migrating from Another VPS Host (Optional) 43
  Transferring files from another VPS host ....................................................... 43

Stage 5: Configuring LAMP (Linux, Apache, MariaDB/MySQL and PHP) 44
  Directory Structures and Permissions ............................................................. 44
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFTP (SSH File Transfer)</td>
<td>46</td>
</tr>
<tr>
<td>Apache Configuration</td>
<td>48</td>
</tr>
<tr>
<td>Adding The First Site (VirtualHost) for CloudWire.info</td>
<td>49</td>
</tr>
<tr>
<td>Configuring SSL</td>
<td>50</td>
</tr>
<tr>
<td><strong>Stage 6: Installing WordPress</strong></td>
<td>52</td>
</tr>
<tr>
<td><strong>Stage 7: Securing WordPress</strong></td>
<td>54</td>
</tr>
<tr>
<td>Redirecting HTTP to HTTPS using the rewrite module</td>
<td>55</td>
</tr>
<tr>
<td>Protecting Wordpress Admin (wp-admin) with .htaccess</td>
<td>56</td>
</tr>
<tr>
<td>Protecting wp-login.php with .htaccess</td>
<td>57</td>
</tr>
<tr>
<td>Configuring SFTP for WordPress Updates</td>
<td>58</td>
</tr>
<tr>
<td><strong>Stage 8: Configuring SELinux</strong></td>
<td>59</td>
</tr>
<tr>
<td>Setting Permissive Mode</td>
<td>59</td>
</tr>
<tr>
<td>How SELinux Works</td>
<td>60</td>
</tr>
<tr>
<td>Booleans</td>
<td>63</td>
</tr>
<tr>
<td>Configuring SELinux to Play Nicely with Apache and WordPress</td>
<td>65</td>
</tr>
<tr>
<td>Troubleshooting SELinux</td>
<td>67</td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
<td>70</td>
</tr>
<tr>
<td>Key Takeaways</td>
<td>70</td>
</tr>
<tr>
<td><strong>About the Author</strong></td>
<td>71</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>72</td>
</tr>
</tbody>
</table>
Introduction

I have been maintaining my own web server for several years now, dating back to 2004 when I first starting using CentOS to run my website. Those were the good old days when I switched from authoring websites using Dreamweaver and FTP, to using WordPress and ditching those antiquated tools altogether. Talking of antiquated, I've been working with Unix since 1992 and was a Linux systems administrator for an ISP for several years after that. I've also been learning along the way with each release of CentOS/RHEL, and I have taken much more notice of security hardening including the use of SELinux.

As an experiment, I posted a tweet merely mentioning SELinux which resulted in some predictable responses including:

“hate selinux!
...doesn’t everyone disable selinux at install?

Usability vs security wins every time.”

I really don't blame them for disliking SELinux, as it seems that is a majority opinion. But I hope to change that! If I can get it working and playing nicely with my WordPress site, then so can you. The reason I use SELinux isn't to make my life any more difficult (though that could be true at times!), but it helps me better understand the inner workings of CentOS 7 better, while providing significant levels of security.

Assumptions

For the remainder of this guide, I will assume that you know how to use Vi, Vim or Nano, and you have a basic understanding of the Linux operating system.
Security Primer

Here are the primary security requirements that I will address with this guide:

- SELinux will be enforcing security policies
- IPtables will provide firewall functionality
- TCP Wrapper will be configured for added security
- 2-Factor authentication (Google Authenticator) for WordPress
- SSL certificates must be used and HTTP will redirect to HTTPS
- FTP is not allowed, and all file transfers must be encrypted during transport (SSL / SSH file transfer)
- Multiple websites (virtual hosts) with SFTP users chroot (jailed) to their own directory.
- SSH key based authentication (disable root access via SSH)
- MySQL will be managed from the command line (no phpMyAdmin please!)

Now just a few words on TCP Wrapper before someone freaks out. It's old. Really old. It doesn't protect all services running on the server and it uses the libwrap library which was created in 1990. However, it takes the blink of an eye to configure so think of it like one of these security latches your granny uses on her front door. There is no harm in using it.

Do I really have to use SELinux?

I get it. SELinux is a pain, especially if you tried it back in the days of CentOS/RHEL 4. Oh my, I thought about throwing it all out of the window back then. But it's not really that bad, so ignore the negative comments and learn how to implement it properly.

The NSA (National Security Agency) created SELinux along with Red Hat who continue to be a major contributor. It's a Mandatory Access Control (MAC) system that sets a fixed (Targeted) policy for access. For example, if a policy prevents a user or process from accessing a directory then it will be prevented by SELinux. Period.

For the purposes of this guide I will focus on the targeted policy, since the other option to use MLS (Multi-Level Security) is not in the scope of this guide.
A targeted policy includes a list of processes that will be protected (or confined) by SELinux, and anything not targeted will be unconfined and will use the Discretionary Access Control (DAC) model. Almost all processes listening on the network (such as httpd, sshd) are confined by SELinux. By confining the process, if it is compromised by an attacker then it greatly reduces what they can do.

Before SELinux is enabled, packages will need to be installed and LAMP and WordPress will be configured. SELinux will not be configured until the very end. Reasons for that will become clear later.

Managing Services with systemd

For many years I had become used to using `service` and `chkconfig` commands to manage services with RHEL (RedHat Enterprise Linux) and CentOS. In fact, I first got my hands on a Unix system back in 1992, then got my first ever job as a Unix admin in 1996. I learned about SystemV run levels, and then became used to using `/etc/init.d/` to manage services. It takes a while to shake bad old habits, but CentOS 7 now uses `systemd` as the default init system.

Init (short for initialization) was the first process to start and the last to stop on a SysV (System V Unix) Linux system. This is the basic concept of run levels, each run level representing each state of the system, with run level 0 being shutdown (halt), 3 being multiuser mode (server has booted), and run level 5 is running the desktop environment if you use one (X Server starts and you have a desktop). Run level 6 restarts the system.
SysV is still present in CentOS 7, but you’ll not find much there. If you run the following command, you can see which services are enabled at boot (run level 3):

```bash
# chkconfig --list | grep on
```

Note: This output shows SysV services only and does not include native systemd services. SysV configuration data might be overridden by native systemd configuration.

If you want to list systemd services use 'systemctl list-unit-files'. To see services enabled on particular target use 'systemctl list-dependencies [target]'.

<table>
<thead>
<tr>
<th>Service</th>
<th>0:off</th>
<th>1:off</th>
<th>2:off</th>
<th>3:off</th>
<th>4:off</th>
<th>5:off</th>
<th>6:off</th>
</tr>
</thead>
<tbody>
<tr>
<td>netconsole</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>network</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This will only show SysV services (for example, `netconsole` and `network`). You can also check the `/etc/init.d/` directory.

**systemd**

Instead of `/etc/init.d/ systemd now uses `/etc/systemd/system/`. That directory contains systemd targets and the symbolic link for default.target.

```bash
basic.target.wants
dbus-org.freedesktop.Avahi.service -> /usr/lib/systemd/system/avahi-daemon.service
dbus-org.freedesktop.NetworkManager.service -> /usr/lib/systemd/system/NetworkManager.service
dbus-org.freedesktop.nm-dispatcher.service -> /usr/lib/systemd/system/NetworkManager-dispatcher.service
default.target -> /lib/systemd/system/multi-user.target
default.target.wants
getty.target.wants
multi-user.target.wants
sockets.target.wants
selinuxfsrelabel.service
system-update.target.wants
```

The two targets to focus on are `multi-user.target` and `graphical.target` which are essentially run level 5 (oops I said it again, sorry!). Installing a desktop environment is not required for a web server, so `multi-user.target` (the default target) will be the focus.
List all the currently loaded targets using the following command:

```
# systemctl list-units --target
```

Each service has 3 states; **enabled**, **disabled** or **static**. If a service is **enabled** it will start at boot. Static means it is typically dependent of other services and are controlled automatically.

To list enabled services use the following:

```
# systemctl list-unit-files --type=service | grep enabled
auditd.service enabled
avahi-daemon.service enabled
chronyd.service enabled
crond.service enabled
dbus-org.freedesktop.Avahi.service enabled
dbus-org.freedesktop.NetworkManager.service enabled
dbus-org.freedesktop.nm-dispatcher.service enabled
getty@.service enabled
httpd.service enabled
irqbalance.service enabled
microcode.service enabled
NetworkManager-dispatcher.service enabled
NetworkManager.service enabled
rsyslog.service enabled
sshd.service enabled
sysstat.service enabled
systemd-readahead-collect.service enabled
systemd-readahead-drop.service enabled
systemd-readahead-replay.service enabled
tuned.service enabled
```

When a service is enabled it creates a symbolic link, usually at our multi-user target: `/etc/systemd/system/multi-user.target.wants`. Take a look at this directory and look at the symbolic links:
To start/stop/restart services you can still use service, but the new command is:

```bash
# systemctl status <servicename>
# systemctl stop <servicename>
# systemctl start <servicename>
# systemctl restart <servicename>
```
Let's Try It
The best way to become familiar with this is to experiment with installing NTP with yum, checking the service status, starting it then enabling to start at boot (multi-user target).

```
# yum install ntp
# systemctl list-units --type service --all | grep ntp
ntpd.service loaded inactive dead Network Time Service
ntpdate.service loaded inactive dead Set time via NTP
sntp.service not-found inactive dead sntp.service

# systemctl start ntpd
# systemctl enable ntpd

# systemctl list-units --type service --all | grep ntp
ntpd.service loaded active dead Network Time Service
ntpdate.service loaded inactive dead Set time via NTP
sntp.service not-found inactive dead sntp.service
```

You can see that NTP was loaded but not active, so using start and enable, let’s see if it’s running now:

```
# systemctl enable ntpd
# systemctl list-units --type service --all | grep ntp
```

It is not necessary to include .service (ntpd.service) as systemctl will assume it’s a service you are specifying anyway.

Set the service to start a boot:

```
# systemctl enable ntpd
```

Check again to see which services are enabled, and ntpd should now be listed:

```
# systemctl list-unit-files --type=service | grep enabled | grep ntpd
ntpd.service enabled
```
Understanding IPtables

I consider the firewall as the most important topic of all. Securing your Linux host is, in my opinion, is the first thing you should be doing before hosting any web services. You have already learned all about systemd in the previous section, and you should now be comfortable with the switch from SysV init, otherwise, make sure you have fully understood this before proceeding to this section on IPtables.

If you are responsible for building Linux hosts for web applications, then this will be an especially important topic for you. This might get a little technical, but hang in there.

RHEL (RedHat Enterprise Linux) and CentOS 7 introduces firewalld which is now installed by default instead of IPtables. Another newcomer, but not yet loaded by default with CentOS 7 is Nftables. What’s the difference? Well firewalld is new to the user-space, but it doesn’t replace IPtables.

Confused? I don’t blame you, so let me explain the IPtables architecture. It’s important to understand how IPtables works in order to understand the changes that firewalld and what Nftables brings to the table (pun intended).

We’ll start with this basic architecture diagram for netfilter:

![Netfilter Architecture Overview]

*Figure 1 Netfilter Architecture Overview*
**User-space:** IPtables resides in what the user-space, this is your interface to the firewall for setting up your firewall rules. The same applies to `ip6tables`, `arptables`, `ebtables` and so on. As a firewall administrator, this is where you get stuff done!

**Kernel:** Netfilter is the framework which IPtables is based on. Netfilter implements a series of ‘hooks’ that inspect packets in the protocol stack, such as IPv4. These hooks allow for kernel modules to interact with them. IPtables has a huge list of kernel modules used for its firewalling capabilities. To see a list of IPtables kernel modules, type: `cat /proc/net/ip_tables_matches`.

**Hardware:** This is our network adapters; `eth0`, `eth1` and so on. We’re talking about the data link layer (OSI layer 2). Netfilter uses prerouting and postrouting to and from the network stack to inspect packets sent and received on each interface.

So the packet inspection is done at the kernel layer with the netfilter, and all the firewall rules and tools to manage the firewall reside in the user-space. Got it.

**firewalld**

As mentioned previously, firewalld is now the default with RHEL and CentOS 7. The main difference is that firewalld provides dynamic rule management (in place changes) and the concept of network ‘zones’, as opposed to IPtables which has a static ruleset.

If I want to make a rule change with IPtables then I have to flush out all the existing rules (`iptables -F`) then save the new rules to `/etc/sysconfig/iptables` (`iptables-save`). That is not necessary firewalld.

Both firewalld and IPtables use the IPtables service to talk to the netfilter. The change is at the user-space layer (`firewall-cmd`).

**Note** Whether or not to use firewalld or IPtables is up to you. I am not recommending one over the other, but I thinking having a solid understanding of IPtables will strengthen your understanding of firewalling with Linux.
Nftables

There have been some interesting developments since the 3.13 kernel was released in 2014. Nftables is the successor to IPtables, and introduces a completely new packet filtering framework on the kernel (3.13 or higher). While Nftables is the successor to IPtables, it is out of scope for the current version of this guide.

Getting Started with IPtables

Let’s get started with some IPtables basics. First, replace firewalld with IPtables. To do that, stop and mask (hide) the firewalld service, install iptables-services and enable IPtables. It is recommended to mask, rather than disable the service, since disabling a service removes it from what was previously known as the run level, the .wants file (e.g. basic.target.wants). When using the mask option, it creates a symbolic link to /dev/null so it cannot be started.

1. Stop and disable firewalld

```
# systemctl stop firewalld
# systemctl mask firewalld
```

2. Install iptables-services

```
# yum -y install iptables-services
```

3. Enable IPtables

```
# systemctl enable iptables
# systemctl enable ip6tables
# systemctl start iptables
# systemctl start ip6tables
```

Note Remember, the options for IPtables are case-sensitive.
At this stage you have no firewall rules. To check your existing rules, use:

```
# iptables -L
```

There are heaps of tutorials out there for IPtables, but I want to give you enough information to understand the basics and adopt some security best practices. Before proceeding any further I should explain what chains are all about.

**Note** Using verbose output (`iptables -L -v`) is useful to show stats on each rule. If there are no matches on a given rule, then you can mark it for deletion.

**What are Chains?**

A chain is a set of rules, checked one by one until it is matched. There are 3 chains: INPUT, FORWARD and OUTPUT (processed in that order).

Notice in the example above that the default policy for each chain is set to ACCEPT. In other words, it will expect you to create your rules on which traffic to deny, leaving everything else to get through. It’s far more secure to do the opposite, so we’ll set the default policies to DROP.

⚠️ *Wait.* If the default policy is set to drop, then you will lose connectivity to your Linux host, assuming of course you are connected via SSH. To avoid this, the drop policies will be configured in a later step.
Order of Chains
1. Flush all chains
2. Add rules for connections you want to drop
3. Add rules for connections you want to accept
4. Add logging rules (optional)
5. Set default policy to drop for each chain
6. Save and restart

Flushing the Rules
Remember that IPtables contains a static configuration, unlike firewalld, so if you implement a set of rules and want to make a change you’ll need to flush the ruleset, save and restart. Let’s look at this now.

1. Stop the IPtables service:
   
   ```
   # systemctl stop iptables
   ```

2. Flush out all of the rules (start over)
   
   ```
   # iptables -F
   ```
Creating Our First IPtables Rules
It would make sense to start with the connections to drop, but that will be covered in the Limited Other Attack Vectors section.

Let's create the first basic ruleset (set of chains) to allow incoming SSH connections:

```bash
# iptables -A INPUT -i eth0 -p tcp --dport 22 --state NEW,ESTABLISHED -j ACCEPT
# iptables -A OUTPUT -o eth0 -p tcp --sport 22 --state ESTABLISHED -j ACCEPT
```

I'll break this down into chunks that are easier to understand. First, look at the first line:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iptables -A INPUT</code></td>
<td>Append the rule to the end of the selected chain, in this case INPUT.</td>
</tr>
<tr>
<td><code>-i eth0</code></td>
<td>Specify the network interface, in this case it's eth0</td>
</tr>
<tr>
<td><code>-p tcp</code></td>
<td>The protocol is TCP</td>
</tr>
<tr>
<td><code>--dport 22</code></td>
<td>The destination port is 22 (default for SSH)</td>
</tr>
<tr>
<td><code>-m state</code></td>
<td>Specifies what to match, in this case it needs to check a given state</td>
</tr>
<tr>
<td><code>--state</code></td>
<td>States are defined as NEW and/or ESTABLISHED, in addition to RELATED or INVALID. In this example, it will match NEW and ESTABLISHED</td>
</tr>
<tr>
<td><code>-j ACCEPT</code></td>
<td>Since anything else will be dropped, accept this chain (rule) to allow SSH</td>
</tr>
</tbody>
</table>

Later in this guide you will learn how to set the default policy to deny all other traffic for all 3 chains. SSH traffic will also need to be allowed back out of eth0 (notice the use of -o for out interface). Only ESTABLISHED is required as it is the response back to the SSH client (OUTPUT) for a connection that has already been established.
Logging

I find it really useful to have a log for our **INPUT** and **OUTPUT** chains for anything that is dropped. Remember that IPtables processes each rule sequentially, so once you have finished adding rules to **ACCEPT** a connection, you will add logging rules.

```bash
iptables -N LOGINPUT
iptables -N LOGOUTPUT
iptables -A INPUT -j LOGINPUT
iptables -A OUTPUT -j LOGOUTPUT
iptables -A LOGINPUT -m limit --limit 4/min --log-prefix "DROP INPUT: " --log-level 4
iptables -A LOGOUTPUT -m limit --limit 4/min --log-prefix "DROP OUTPUT: " --log-level 4
```

Let's break this down into 3 parts.

**Part 1**

```bash
iptables -N LOGINPUT
iptables -N LOGOUTPUT
```

Create new chains for logging; **LOGINPUT** and **LOGOUTPUT**

**Part 2**

```bash
iptables -A INPUT -j LOGINPUT
iptables -A OUTPUT -j LOGOUTPUT
```

Use `iptables -A` to append a new rule for our **INPUT** and **OUTPUT** chain and send it to a target with `-j`. In this case our target is our new chain **LOGINPUT** or **LOGOUTPUT**.

**Part 3**

```bash
iptables -A LOGINPUT -m limit --limit 4/min --log-prefix "DROP INPUT: " --log-level 4
iptables -A LOGOUTPUT -m limit --limit 4/min --log-prefix "DROP OUTPUT: " --log-level 4
```

Using the limit matching module (`-m limit`) you can ensure that no more than 4 per minute is logged, which will stop the logs from filling up too fast. The logs will be sent to `/var/log/messages` (`--log`) and classified as warning entries (`--log-level 4`). I'll cover monitoring for attacks later in the article.
Set policy to drop all other traffic

The final three lines change the policy (-P) for the built in chains (INPUT, FORWARD and OUTPUT) to DROP. It is possible to leave the last OUTPUT chain to ACCEPT, given this is for traffic coming from the host (which is trusted), but I personally think it’s good practice to be as secure as possible. It also means you specify exactly what you allow in, and out.

```bash
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```

Recap What You Have Learned So Far

So there you have your first set of basic IPtables rules. This is what you have learned so far:

- Stopping IPtables and flushing out any existing rules
- Created the first rule to allow incoming SSH connections
- Added a logging chain for incoming and outgoing connections (INPUT and OUTPUT)
- We’ve set the default policy to drop everything else

Here is the first basic firewall ruleset with IPtables. You will allow ssh connections (into eth0) and allow any established ssh connections out of eth0. Anything else being dropped is logged. Go ahead and copy the following into your console (remember to flush your configuration first with `iptables -F`).

```bash
```
Saving and Restarting IPtables

Now you've copied your first few IPtables rules into the console, you need to save it and restart IPtables.

```
# iptables-save > /etc/sysconfig/iptables
# systemctl restart iptables
```

**Note**  If you lose access to your SSH console at this point, then you've blocked your own connection. Make sure the interface name is correct (`-i eth0`) as you may have something different. Once you've found the issue, stop the IPtables service, flush the rules and add your rules again.

Advanced SSH Security

Within minutes of booting a Linux virtual machine with my hosting provider, my host was under a brute force attack for root. Great, let's have some fun. Don't worry, we'll dive back into IPtables again after this :)

It's strongly recommended to disable root login with SSH. By default, your SSH configuration will allow root to login. Another trick is to change the maximum number of SSH authentication attempts, change the SSH listening port from 22 to something else (e.g. 9292), and tweak the firewall rules to rate-limit the number of SSH connections within a given period of time.

Disable root SSH login

There should be no reason to allow root login from SSH. To make sure this is disabled, edit `/etc/ssh/sshd_config` and set `PermitRootLogin` to `no`. 

```
iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT
iptables -N LOGINPUT
iptables -N LOGOUTPUT
iptables -A INPUT -j LOGINPUT
iptables -A OUTPUT -j LOGOUTPUT
iptables -A LOGINPUT -m limit --limit 4/min -j LOG --log-prefix "DROP INPUT: " --log-level 4
iptables -A LOGOUTPUT -m limit --limit 4/min -j LOG --log-prefix "DROP OUTPUT: " --log-level 4
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```
# vi /etc/ssh/sshd_config

```
PermitRootLogin no
```

Restart SSH for these changes to take effect:

```
# systemctl restart sshd
```
Changing the Maximum Authentication Attempts

While editing `/etc/ssh/sshd_config` it is also possible to change the maximum number of connection attempts per connection before it disconnects that connection. Most modern brute force techniques mitigate this by establishing a new connection attempt each time, but you can also protect against that using a combination of rate-limiting and Fail2ban.

```
MaxAuthTries 3
```

Changing the SSH Port

Another effective, but not infallible technique, is changing the default port for ssh from 22 to something else. Most port scans will sweep a range of IP addresses to find common open ports that can be targeted, such as 21 (FTP), 22 (ssh), 23 (telnet) and so on. Some port scans will even try similar port numbers (222, 2222) to see if ssh is listening.

Another advantage to changing the listening port for ssh is a reduction in failed ssh connection attempts. For this example, the port has been changed to 9922.

```
Port 9922
```

**Note**  Remember to change your firewall rules accordingly. This example uses 9922, but you can choose anything not used by other services.

Rate-Limiting SSH Connections

Finally, my favorite trick is rate-limiting, to drop more than 3 connections per minute from an IP address. In fact, you can adjust the time frame accordingly to be more aggressive than that. In this example for IPtables, rate-limiting is set to no more than 3 connections every 60 seconds.

```
iptables -A INPUT -i eth0 -p tcp --dport 9922 -m state --state NEW -m recent --set --name SSH
iptables -A INPUT -i eth0 -p tcp --dport 9922 -m state -- NEW -m recent --update --seconds 60 --hitcount 4 --rttl --name SSH -j DROP
iptables -A INPUT -i eth0 -p tcp --dport 9922 -m state --state NEW, ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 9922 -m state --state ESTABLISHED -j ACCEPT
```
Notice that the last two lines are exactly the same as before, but port 9922 is used for ssh as this was changed in the last section. Adding `--set --name SSH` makes it easier to read later on. There are two new rules that use the `--recent` extension. This creates a list of IP addresses that match a certain criterion.

Let's take a closer look at each one:

**Line 1**

```
iptables -A INPUT -i eth0 -p tcp --dport 9922 --state --state NEW --m recent --set --name SSH
```

This command is the same as before, appending `--A` to the `INPUT` chain on interface `eth0` on TCP destination port 9922, and matching the `NEW` state.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--m recent</code></td>
<td>Match the rule using the <code>recent</code> extension.</td>
</tr>
<tr>
<td><code>--set</code></td>
<td>This will add the source address of the packet to the list</td>
</tr>
<tr>
<td><code>--name SSH</code></td>
<td>Sets the list name to SSH (case sensitive). This can be changed to your own naming preference. The list is stored in <code>/proc/net/xt_recent/SSH</code></td>
</tr>
</tbody>
</table>
Line 2

```
iptables -A INPUT -i eth0 -p tcp --dport 9922 -m state --state NEW -m recent --update --seconds 60 --hitcount 4 --rttl --name SSH -j DROP
```

The second line then checks if the connection was last seen in the last 60 seconds and has received at least 4 connections (hitcount => 4).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--update</td>
<td>Update the match when used with --seconds and --hitcount</td>
</tr>
<tr>
<td>--seconds</td>
<td>Will match if last seen in the specified number of seconds</td>
</tr>
<tr>
<td>--hitcount</td>
<td>Will match if the hit equals or is higher than the hitcount</td>
</tr>
</tbody>
</table>
Limiting Other Attack Vectors

Christmas trees, floods and empty packages. Sounds like someone is having a really bad day!

IPtables can also be used to help protect against these attacks. A Christmas tree packet has all options set (SYN, FIN, URG and PSH). If you are receiving a large number of these then someone may be trying to conduct a denial-of-service (DoS) attack.

Null packets usually indicate that your host is being scanned, as a single packet with no flags set. This is never legitimate, so it should be dropped.

Syn-floods are also a possible threat from denial-of-service (DoS) attacks, as the connection is started with SYN (part of the three-way handshake) but then it doesn’t respond with an ACK (acknowledge). A bit like ringing someone's doorbell, waiting for them to open the door and then running away.

Drop XMAS packets

```bash
iptables -A INPUT -p tcp --tcp-flags ALL -j DROP
```

Drop NULL packets

```bash
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP
```

Drop Syn-flood

```bash
iptables -A INPUT -p tcp ! --syn ! --state --state NEW -j DROP
```
Host Access (TCP_WRAPPERS)
This has nothing to do with IPtables, but worth a mention. There are two host access files (/etc/hosts.allow and /etc/hosts.deny), that are part of the TCP_WRAPPER package. This makes it possible to allow or deny access to certain services based on the IP address.

This particular configuration has been around since the late 1990s, and it provides an additional layer of security, albeit a basic one. Not all services support TCP wrappers, but sshd does support it. You can check whether a service supports TCP wrappers by getting the path to the service using `whereis sshd`, and then searching for the `libwrap.so` library.

```bash
# whereis sshd
sshd: /usr/sbin/sshd /usr/share/man/man8/sshd.8.gz
# ldd /usr/sbin/sshd | grep libwrap.so
libwrap.so.0 => /lib64/libwrap.so.0 (0x00017ffceg8f1100)
```

If you try it with httpd (which is typically located in /usr/sbin/httpd) you'll see it produces no result, since httpd doesn't support TCP wrappers.

Here are some examples of its usage:

```bash
# vi /etc/hosts.allow
sshd:XX.XX.XX.XX
# vi /etc/hosts.deny
ALL:ALL
```

Using this example, a single IP address (XX.XX.XX.XX) is allowed to access the host with SSH (sshd) in `hosts.allow`. Then anything else is denied in `hosts.deny`. Be careful with this though, if your IP address changes then you will not be able to access your server without console access.

**Note** You can enter multiple IP address here (separated by spaces) or to allow SSH from any IP with `sshd:ALL`. 
Blocking a IP Addresses with IPtables

This is a simple example of how to block a specific IP address or entire subnet using the slash (/) notation:

```
# iptables -I INPUT -i eth0 -s XX.XX.XX.XX -j DROP
# iptables -I INPUT -i eth0 -s XX.XX.XX.0/24 -j DROP
```

Common Firewall Rules for Web Hosts

Taking stock on the current progress, IPtables has protected against some common attack vectors, but ports required to run a web host are still blocked. The best approach is to allow the required services and then deny anything else, therefore a minimal rule set is required.

- Set the default policies to drop ✓
- Drop common DDOS attack vectors ✓
- Accept incoming connections (rate-limited) on SSH ✓
- Allow incoming HTTP / HTTPS requests ✓
- Allow host to perform DNS queries ✓
- Allow host to perform NTP updates ✓
- Allow localhost inbound and outbound ✓
- Allow outbound connections to HTTP / HTTPS (web browsing, yum updates) ✓
- Allow outbound SMTP (so the host can send email) ✓
- Log denied requests ✓
Hopefully you now have an understanding on how IPtables firewall rules are configured. It’s time to address each of the remaining firewall rules:

**Allow incoming HTTP / HTTPS requests**

```
iptables -A INPUT -i eth0 -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 80 -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 443 -m state --state ESTABLISHED -j ACCEPT
```

**Allow host to perform DNS queries**

```
iptables -A INPUT -i eth0 -p udp --sport 53 -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p udp --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
```

**Allow host to perform NTP updates**

```
iptables -A OUTPUT -o eth0 -p udp --dport 123 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p udp --sport 123 -m state --state ESTABLISHED -j ACCEPT
```

**Allow localhost inbound and outbound**

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
```

**Allow outbound connections to HTTP / HTTPS**

```
iptables -A OUTPUT -o eth0 -p tcp --sport 80 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --sport 80 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 443 -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 443 -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
```
Allow outbound SMTP

```
iptables -A INPUT -i eth0 -p tcp --sport 25 -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --dport 25 -m state --state NEW,ESTABLISHED -j ACCEPT
```

Final IPtables Rules

Pulling all of this together should provide you with a complete IPtables ruleset for your web server. Remember you may need to change the network interface name, and you can check this on your Linux host by typing `ifconfig`, this will list the network interfaces with the corresponding names.

Before copy/pasting these rules, remember to stop the IPtables server (`systemctl stop iptables`) and then flush the existing ruleset (`iptables -F`). Once the rules have been added, you will need to save them, then restart the service:

```
# iptables-save > /etc/sysconfig/iptables
# systemctl restart iptables
```
Sample IPtables Ruleset

# Drop NULL packets
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP

# Block syn flood attack
iptables -A INPUT -p tcp ! --syn -m state --state NEW -j DROP

# Block XMAS packets
iptables -A INPUT -p tcp --tcp-flags ALL ALL -j DROP

# SSH Rate-limit new connections (drop if more than 3 attempts in 60 seconds) and allow only established SSH connections
iptables -A INPUT -i eth0 -p tcp --dport 9922 -m state --state NEW -m recent --set --name SSH
iptables -A INPUT -i eth0 -p tcp --dport 9922 -m state --state NEW -m recent --update --seconds 60 --hitcount 4 --rttl --name SSH -j DROP
iptables -A INPUT -i eth0 -p tcp --dport 9922 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 9922 -m state --state ESTABLISHED -j ACCEPT

# Allow SSH for outbound connections
iptables -A OUTPUT -o eth0 -p tcp --dport 9922 -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --sport 9922 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Perform DNS Queries
iptables -A INPUT -i eth0 -p udp --sport 53 -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p udp --sport 53 -m state --state NEW,ESTABLISHED -j ACCEPT

# Allow NTP
iptables -A OUTPUT -o eth0 -p udp --dport 123 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p udp --sport 123 -m state --state ESTABLISHED -j ACCEPT

# Web Server (HTTP/HTTPS)
iptables -A INPUT -i eth0 -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 80 -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 443 -m state --state ESTABLISHED -j ACCEPT
# Web Browsing
```bash
iptables -A INPUT -i eth0 -p tcp --sport 80 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --sport 443 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --dport 80 -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --dport 443 -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

# Allow Inbound/Outbound to Localhost
```bash
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
```

# Allow SMTP outbound (sendmail)
```bash
iptables -A INPUT -i eth0 -p tcp --sport 25 -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --dport 25 -m state --state NEW,ESTABLISHED -j ACCEPT
```

# Log all dropped packets
```bash
iptables -N LOGINPUT
iptables -N LOGOUTPUT
iptables -A INPUT -j LOGINPUT
iptables -A OUTPUT -j LOGOUTPUT
iptables -A LOGINPUT -m limit --limit 4/min --log-prefix "DROP INPUT: " --log-level 4
iptables -A LOGOUTPUT -m limit --limit 4/min --log-prefix "DROP OUTPUT: " --log-level 4
```

# Set policies to drop everything else
```bash
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```
Monitoring Logs

Logs are stored in `/var/log/`, and it is important to monitor both `/var/log/messages` and `/var/log/secure`. Use the `tail -f` command to follow data being appended to a log file as it is appended. This is particularly useful to monitor denies.

1. Monitor all security related logs, such as authentication failures, SSH logins, failed login attempts, and more.

   ```bash
   # tail -f /var/log/secure
   ```

2. Monitor the system log (send IPTables logging here)

   ```bash
   # tail -f /var/log/messages
   ```

   **Note**  Remember your SSH rate-limiting rules? This creates a file in `/proc/net/xt_recent/` (e.g. `/proc/net/xt_recent/SSH`). Check the contents of the file to see which IP addresses have been caught exceeding the rate-limit.

   Finally, as I mentioned earlier don’t forget to use `iptables -L -v` (verbose) from time to time. This is a trick I used back in the days when I was a maintaining and troubleshooting firewalls. Why? If you see a firewall rule has no hits after some time, then the likelihood is that it’s not required and it is time to tidy up your IPtables configuration.
Stage 1: Deploying a new virtual private server (VPS)

When you have a brand new virtual machine up and running with CentOS 7, first thing to do is make sure it's up to date. You will initially need access to your virtual machine using the console (not SSH).

First, login with the console as root and apply the latest updates:

```
# yum check-update
# yum update -y
```

Securing Access

Adding a new user

Root access via SSH will be disabled, and a standard user account will be used for administering the host. Whenever root privileges are required, `sudo` will be used. In this example, Fred the systems administrator will be added.

```
# useradd fred
# passwd fred
# visudo
```

Note   This will open the sudoers file `/etc/sudoers`

1. Look for the line `root ALL=(ALL) ALL` and add the new user account below it. This will allow the new user account to have sudo privileges.

```
fred ALL=(ALL) ALL
```

2. Save the file (`:wq`)

Disable Root SSH Access

Edit `/etc/ssh/sshd_config` and edit the following:

```
PermitRootLogin no
MaxAuthTries 3
```

Now logout, and log back in as the new user. Use `sudo -i` to login with root privileges if needed.
Configuring SSH Key Based Authentication

By this stage you should be logged in with your new user account (not root). PuTTYgen will be used to generate an RSA key pair and configure PuTTY to access the server using the private key.

Generate the public/private key-pair

1. Launch PuTTYgen make sure SSH-2 RSA is selected and 2048 bits.
2. Copy the public key in the text box, which starts with ssh-rsa AAAA and save it in a text file somewhere safe (For example, public.key).
3. Save the private key (For example, private.ppk) and don’t lose it!

Copy the public key to the server

In order for public/private key authentication to work, your server must have a copy of the public key in ~/.ssh/authorized_keys. ~/.ssh denotes your home directory (/home/bob), and you may need to create the .ssh directory first. Just make sure both .ssh and the authorized_keys file is owned by the user being authenticated (not root). The permissions need to be set as follows:

```
# chmod 700 .ssh
# chmod 644 authorized_keys
```

If the permissions are not set correctly, you may receive the error: Server refused our key

Note: If you still get this error, make sure authorized_keys is owned by correct user account (not root).

Paste the contents of the public key in to the file ~/.ssh/authorized_keys. If this directory or file doesn't exist, create it.

The authorized_keys file should look something like this:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEA5KiG06JFR8AtacUL1xd271So30pkUY3I2KYzGR+BuGjEJeCqgN66LYTUw2ygFs4VhOwZ5Pmiq9T2RskiK6/gMa0VVoFM18xUS9Emwdq3pxWDSQ0p2ZgUtb+7bix+h7xpu40iAa1LE7JSSWsvzaczKEgjgO61RQgq5WEAtV3E+Ks5tqyJ5/3PiEnLCQOcmA9OXPYb7s37/X1i5iSeZfqsF+8nOBf+yh72xwP2fIix8KL8cVak9MKyeKdBPMwM5RIaI0Ek87/idqinal4m9j+QoD7ajGnm4NvOoD//K7uozEXcBZGiMEFX17aAcY44hWM462zJQ/M1/y2sco41Q== rsa-key-20170906
```
Load the private key into PuTTY

1. Launch PuTTY and configure the hostname under the Session category.
2. Click on Connection > Data and type the Auto-login username
3. Click on Connection > SSH > Auth and Browse to the location of the private key.
4. Click on Session and save your configuration.

Now when you launch PuTTY it will present the private key, and this will match the corresponding public key on the server. Only you will hold a copy of the private key.

Disable SSH Password Authentication

This step is optional but strongly recommended. By disabling SSH password based authentication, then SSH can only be authenticated with the private key you configured in the last step. You still need passwords in order for sudo to work, but this step will only access certificate based authentication when you SSH to the host.

Edit `/etc/ssh/sshd_config` and set the following:

```
PasswordAuthentication no
```

Note You’ll need to configure Wordpress to use authentication keys if you disable SSH password authentication. This requires both the public and private key to be installed on the Wordpress server which I think is a terrible idea. One option is to temporarily enable password authentication for SSH for when Wordpress needs it.
Installing Core Packages

Now it is time to install the core packages needed to get the secure server up and running. Wget is later required to download the RPMs for the Remi repository, and it is a useful tool to have on the host anyway. Later in the guide NTP will be configured for time synchronization. Finally, `yum-cron` will be used to make sure the CentOS host is updated regularly, and the EPEL repository will be used.

Core Packages

```bash
# yum install -y wget ntp gcc net-tools iptables-services yum-cron
sshfs epel-release
```

SMTP (Optional)

Next install Sendmail or Postfix. This is required if you want WordPress to send email notifications, or if you are using a website contact form. Since `yum-cron` is installed already, it can also be configured to send email notifications.

```bash
# yum install -y postfix
```

LAMP Stack

Next is the rest of the LAMP stack, which includes Apache, OpenSSL, PHP, and other associated components.

```bash
# yum install -y httpd mariadb-server mariadb openssl mod_ssl
libstdc++ libssh2 libssh2-devel libssh2-docs libssh2 libssh2-devel
```

Installing the Remi Repository

CentOS 7 doesn't install the latest version of PHP from the CentOS/RHEL or EPEL repositories. CentOS 7.3 will install PHP version 5.4.16, not PHP 7.1 which is the latest version at the time of writing this.

The solution to this is to use the Remi repository.
First install the repo:

```bash
# wget https://rpms.remirepo.net/enterprise/remi-release-7.rpm
# rpm -Uvh remi-release-7.rpm
```

Next, configure the repository and set the first entry to enabled=1. If you want to use PHP 5.6 and not the very latest (PHP 7.1) then set enabled=1 in the [remi-php56] section.

```bash
# vi /etc/yum.repos.d/remi.repo

[remi]
name=Remi's RPM repository for Enterprise Linux 7 - $basearch
#baseurl=http://rpms.remirepo.net/enterprise/7/remi/$basearch/
#mirrorlist=https://rpms.remirepo.net/enterprise/7/remi/httpsmirror
mirrorlist=http://rpms.remirepo.net/enterprise/7/remi/mirror
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-remi

```

### Installing PHP 7.1

If you want to install PHP 7.1 then edit remi-php71repo and set enabled=1 (make sure 5.6 disabled in the remi.repo as described in the previous step):

```bash
# vi /etc/yum.repos.d/remi-php71.repo

[remi-php71]
name=Remi's PHP 7.1 RPM repository for Enterprise Linux 7 - $basearch
#baseurl=http://rpms.remirepo.net/enterprise/7/php71/$basearch/
#mirrorlist=https://rpms.remirepo.net/enterprise/7/php71/httpsmirror
mirrorlist=http://rpms.remirepo.net/enterprise/7/php71/mirror
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-remi

```

Install PHP and use Yum to update the server:

```bash
# yum install php php-pecl-ssh2 gcc php-devel php-pear php-gd php-mysql php-mcrypt php-mbstring
# yum update -y
```
Check which version of PHP is installed:

```
# php --version

PHP 7.1.9 (cli) (built: Aug 30 2017 20:06:08) ( NTS )
Copyright (c) 1997-2017 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2017 Zend Technologies
```

You should now have the latest version from the branch you selected.

**Basic Server Configuration**

**Configuring yum-cron for automatic updates**

Enable the yum-cron service to make sure it starts automatically.

```
# systemctl enable yum-cron
```

Edit the `/etc/yum/yum-cron.conf` file and edit the following lines:

```
# vi /etc/yum/yum-cron.conf

update_cmd = security
apply_updates = yes
```

This is what will happen daily. You can optionally configure `/etc/yum/yum-cron-hourly.conf`, which as the name suggests is run every hour. By default, the hourly configuration won't download or apply anything.

Finally, start yum-cron:

```
# systemctl start yum-cron
```

**Note**  `/var/log/yum.log` contains a list of all installed packages.
NTP

Edit vi /etc/ntp.conf and configure NTP servers, for example:

```
server 0.centos.pool.ntp.org iburst
server 1.centos.pool.ntp.org iburst
server 2.centos.pool.ntp.org iburst
server 3.centos.pool.ntp.org iburst
```

Next enable NTPD and start the service:

```
# systemctl enable ntpd
# systemctl start ntpd
```

Set the correct time zone for your region:

```
# timedatectl list-timezones
# timedatectl set-timezone 'America/New_York'
```

You can check the time and date using the `date` command.
Stage 2: IPtables Web Server Configuration

Earlier in this eBook I provided a sample IPtables configuration. At this stage, you should have already disabled root SSH access, change the SSH port to something other than port 22, and set a maximum number of authentication attempts in order to minimize the likelihood of a brute force attack. Finally you should have configured your IPtables rules.

Note  I like to keep a copy of my IPtables rules saved, without the comments so when I need to make changes I can simply flush the rules (iptables -F), copy, paste, save and restart.

Installing Fail2Ban

Fail2ban works by dynamically adding IP addresses to the firewall that have failed a given number of login attempts. It's very easy to install and configure.

Create a basic configuration:

```bash
# yum install -y fail2ban
# systemctl enable fail2ban

# vi /etc/fail2ban/jail.local
[DEFAULT]
# Set a 1 hour ban
bantime = 3600

# Override /etc/fail2ban/jail.d/00-firewalld.conf
banaction = iptables-multiport

[sshd]
enabled = true
```
Stage 3: MariaDB (MySQL)

MariaDB (which replaces MySQL) has already been installed as part of the core package installation. In order to start at boot time, the service needs to be enabled.

```bash
# systemctl enable mariadb
# systemctl start mariadb
```

Before proceeding, MySQL (MariaDB) needs to be secured. Using `mysql_secure_installation`, the implementation of MySQL is hardened for production use. You can do this later since it won't delete any existing databases, but I recommend you do this now.

```bash
# mysql_secure_installation
```

There won't be a password set since MariaDB is a fresh installation.

Type `mysql -u root -p` and hit enter.

You should see something similar to the following:

```
Welcome to the MariaDB monitor. Commands end with ; or \
g.
Your MariaDB connection id is 5
Server version: 5.5.52-MariaDB MariaDB Server

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

**Note**  You can list existing databases using the `SHOW DATABASES;` SQL query.
1. First set the root password for MySQL:

```sql
USE MYSQL;
UPDATE USER SET PASSWORD=PASSWORD("password") WHERE USER='root';
FLUSH PRIVILEGES;
```

Next create a database for your WordPress site. Change the database name to whatever you want, this will identify your WordPress database, (see following example):

```sql
CREATE DATABASE cloudwire;
```

2. Create a database user for the Wordpress site (Change username and password to your own!):

```sql
CREATE USER 'wireyuser'@'localhost' IDENTIFIED BY 'c0mpl3xP$$SSSw000rrrdddd';
```

3. Grant privileges to WordPress database

```sql
GRANT ALL PRIVILEGES ON cloudwire.* TO 'wireyuser'@'localhost';
FLUSH PRIVILEGES;
exit
```

**Note** If you have multiple databases, just repeat from step 2.
Stage 4: Migrating from Another VPS Host (Optional)

If you already have another WordPress installation you wish to migrate, follow these steps to move your files and MySQL database to the new server. If you are starting from scratch, then feel free to skip this section and move to Stage 5.

1. Backup your existing WordPress database on the origin server:

   ```
   # mysqldump -u root -p DB_NAME > /home/fred/wordpress.sql
   ```

2. Next, copy the exported database (.sql file) to your new server.

3. Import the WordPress database into MySQL on your new server:

   ```
   # mysql -u root -p DB_NAME < /home/fred/wordpress.sql
   ```

   **Note**  You can use WinSCP or Filezilla to copy files using SSH. Remember you may need to use your SSH private key in order to authenticate.

Transferring files from another VPS host

There are a few ways of doing this. I used to use sshfs to mount the SSH file transfer, but rsync is able to establish an SSH file transfer all in one hit. This is the fastest method in my opinion.

First, using WinSCP or Filezilla, make sure you can access the other VPS host and see all of the files required for the transfer. Using rsync everything will be copied to the new VPS.

Transfer the files using rsync:

```
# rsync --ignore-existing -prvh -e "ssh -p 22"
fred@ip_address:/data/oldsitedir/ /data/.sitedir/
```

**Note**  If you are unable to establish a connection, make sure you have allowed ssh connections outbound (not just inbound).
Stage 5: Configuring LAMP (Linux, Apache, MariaDB/MySQL and PHP)

By this stage you have either created an empty database, or have migrated a database from another host and have all of the files and directory structure in place.

If you are building from scratch then so far you have an empty database for WordPress, but Apache hasn't been configured yet or a new directory structure created. Since this is a multi-site virtual host, Apache will use HTTP headers to determine which website directory to serve.

SSL (HTTPS) will be used for the fictitious blog (cloudwire.info), and since host headers only work with HTTP (not HTTPS), a unique IP address is required for each site listening on 443. In this example, one site (cloudwire.info) will be configured on HTTPS (443), so a single IP address allocated by the VPS provider will be used. Your VPS provider will be able to tell you how to add additional IP addresses if they are needed.

Directory Structures and Permissions

It is really up to you how to create your directory structure. I prefer to create a new hidden directory, such as /data/.sitehome/ and then create a directory for each site. For example, /data/.sitehome/cloudwire.info/. Being a hidden directory (staring with a dot) doesn't add that much in terms of security, but it just a practice I have got used to over the years.
Obviously you should think of a better naming convention than I have. The key here is each website owner will have a user account that is jailed to their website directory. This user account will also be used with WordPress for installing updates, plugins and themes. You should have something similar to the following:

```
#/data/.sitehome/cloudwire.info/
--- web
--- private
--- logs
--- cgi-bin

#/data/.sitehome/wordpress_site2/
--- web
--- private
--- logs
--- cgi-bin
```

Create your own site directory structure:

```
# cd /data/.sitehome/
# mkdir cloudwire.info
# cd cloudwire.info
# mkdir logs private cgi-bin web
```
SFTP (SSH File Transfer)

This is the way to go. Forget FTP. SSH is already configured, so in order to use it effectively for file transfer (SFTP) chroot, or a jail, will be used to confine the user to their home directory.

For each site being hosted, add a site admin account for SFTP. SSH access will be disabled for each of these users. I like to use random usernames for each site that can't be guessed very easily.

```bash
# useradd username -d /data/.sitehome/wordpress001/
```

**Note** You will get a warning if you have created the directory, you can ignore this.

If you have already added the user, you can simply modify the home directory using:

```bash
# usermod -m -d /data/.sitehome/ username
```

1. First add a group called sshonly

```bash
# groupadd sshonly
```

2. Add the user to be chrooted to that group. A new username will be created for the first website 'cw001' for the cloudwire.info site.

```bash
# usermod -aG sshonly username
```

3. Edit `/etc/ssh/sshd_config` and change the sftp SubSystem line from `/usr/libexec/openssh/sftp-server` to:

```bash
Subsystem sftp internal-sftp
```

**Note** This doesn't prevent SSH access, it allows file transfer using SFTP.
4. Add the following to the end of `sshd_config`:

```bash
Match Group sshonly
ChrootDirectory %h
ForceCommand internal-sftp
X11Forwarding no
AllowTcpForwarding no
```

5. Restart SSHD

```bash
# systemctl restart sshd
```

6. Change the group of the root directory and all sub-directories to `sshonly`.

```bash
# chgrp sshonly /data/.sitehome/wordpress001/ -R
```

**Note** If you need to modify an existing user account with a new home directory, use the following command:

```bash
usermod -m -d /data/.sitehome/wordpress001/ username
```

7. Everything inside the website directory needs to be owned by the new user account (E.g. cw001). Don't give them ownership of the website directory itself, just everything inside.

```bash
# chown username * -R
```

**Note** Steps 2-4 above ensures that the user cannot login to the host using SSH, but they can use SFTP.
Apache Configuration

If you have stuck with me this far into the guide, then give yourself a pat on the back. No really, do it. Go and grab another coffee, change your music playlist and let's finish this beast of a web server!

1. Enable the http daemon

```bash
# systemctl enable httpd
```

2. Edit `httpd.conf` and change the following line. Specify your servers IP address or hostname here:

```bash
# vi /etc/httpd/conf/httpd.conf
ServerName X.X.X.X:80
```

3. Add the VirtualHost entry for the default site at the very bottom of `httpd.conf`. I call this the not-yet page. If anyone reaches the server with the IP address or it can't be matched by an HTTP header, then it will display DocumentRoot as a default page.

```bash
# Load Default VirtualHost
<VirtualHost *:80>
    DocumentRoot /data/.sitedir/not-yet/web
    <Directory /data/.sitedir/not-yet/web>
        AllowOverride None
        Require all granted
    </Directory>
</VirtualHost>
```

4. Save the file (`:wq`) and restart the httpd service:

```bash
# systemctl start httpd
```

**Note** Make sure all of these directories exist and the paths are correct, otherwise HTTPD will fail to start.
At this stage if everything is configured correctly, when you start HTTPD, and browse to the IP address of your server you should be served the contents of `/data/.sitehome/not-yet/webroot/`, you should have index.html in there. I've got a standard 'Not yet!' page I've been using since the 1990s, and as I'm nostalgic I've decided to keep it all these years. Let's try it out! If that works, then proceed to adding the first site.

**Adding The First Site (VirtualHost) for CloudWire.info**

Each site needs a VirtualHost entry. So far, only the default page that is displayed for HTTP (TCP port 80) has been configured. I like to keep things tidy, so rather than adding multiple VirtualHost entries in httpd.conf, 'Include' entries will be used that point to a file in `/etc/httpd/vhosts/`. This is where the VirtualHost configuration file for each site will be stored.

For this example, I've just registered cloudwire.info and configured DNS to point to the IP address of the new server.

1. To start with, at the very bottom of `httpd.conf` add an 'Include' pointing to the VirtualHost file. Give the file a meaningful name, this is just an example below:

   ```
   Include /etc/httpd/vhost/cloudwire.info.conf
   ```

2. Next, create the vhost file. The vhost directory won't exist so you'll need to create that.

   ```
   # vi /etc/httpd/vhost/cloudwire.info.conf
   <VirtualHost *>:80>
   ServerAdmin admin@cloudwire.info
   DocumentRoot /data/.sitehome/cloudwire.info/webroot
   ServerName www.cloudwire.info
   ServerAlias cloudwire.info
   ScriptAlias /cgi-bin/ /data/.sitehome/cloudwire.info/cgi-bin/
   <Directory /data/.sitehome/cloudwire.info/web>
   Options FollowSymLinks
   AllowOverride All
   Require all granted
   </Directory>
   </VirtualHost>
   ```
3. Save the file (:wq) and restart httpd

```
# systemctl restart httpd
```

Providing DNS is updated with the IP address of your server, you should now be able to browse to the new website and it will be displayed.

**Configuring SSL**

Configuring HTTPS for your website first requires an SSL certificate, and it really isn't very difficult at all. I used GoDaddy and they have several options for SSL certificates. If you are concerned about costs, then I recommend Let'sEncrypt which is free. Typically, 3 files are required:

- Certificate file (.crt)
- Certificate private key (.key)
- Intermediate certificates chain (bundle .crt)

You should place the certificate files outside of the web root directory, in this example they are stored in `/data/.certificates/cloudwire.info/`. 
Open the file you created in the previous step (`# vi /etc/httpd/vhost/cloudwire.info.conf`) and add another `VirtualHost` entry as below, but changing it with your own values:

```
<VirtualHost XX.XX.XX.XX:443>
  SSL engine On
  SSLCertificateFile /data/.certificates/cloudwire.info/cloudwire.crt
  SSLCertificateKeyFile /data/.certificates/cloudwire.info/cloudwire.key
  SSLCertificateChainFile /data/.certificates/cloudwire.info/gd_bundle-g2-g1.crt
  ServerAdmin admin@cloudwire.info
  DocumentRoot /data/.sitehome/cloudwire.info/webroot
  ServerName www.cloudwire.info
  ServerAlias cloudwire.info
  ScriptAlias /cgi-bin/ /data/.sitehome/cloudwire.info/cgi-bin/

  <Directory /data/.sitehome/cloudwire.info/web>
    Options FollowSymLinks
    AllowOverride All
    Require all granted
  </Directory>
</VirtualHost>
```

**Note** As you can see, this is why you will need an additional IP address for each site requiring SSL, since you need to specify the IP listening on port 443.
Stage 6: Installing WordPress

While WordPress can be installed using yum, it will assume WordPress is running on a single site web server using /var/www/html for the website location. Since one or more WordPress sites using VirtualHost configuration in Apache will run on the server, wget will be used to grab the latest version of WordPress and install directly into the site directory.

1. Navigate to your home directory

   ```
   # cd /home/fred
   ```

2. Download the latest WordPress binaries and extract the contents of the tarball:

   ```
   # wget https://wordpress.org/latest.tar.gz
   # tar -xzf latest.tar.gz
   ```

3. Copy the contents of the WordPress directory to the website directory (/data/.sitehome/clourwire.info/web/).

   ```
   # cp wordpress/* /data/.sitehome/clourwire.info/web/ -R
   ```

4. Move the sample configuration file up one level (/home/.sitehome/cloudwire.info/) and copy it new wp-config.php. Never place the config file in a publicly accessible location.

   ```
   # mv wp-config-sample.php ../
   # cd ..
   # cp wp-config-sample.php wp-config.php
   ```

5. Edit wp-config.php and configure the database name, username and password as created in Stage 3. If you migrated your WordPress database from another VPS, then enter the details of the WordPress database here.
6. Finally, you should add unique keys that WordPress will use for authorization and encryption. There is actually an online salt generator which generates this configuration for you.

```php
define('DB_NAME', 'database_name_here');
define('DB_USER', 'username_here');
define('DB_PASSWORD', 'password_here');
define('AUTH_KEY', 'put your unique phrase here');
define('SECURE_AUTH_KEY', 'put your unique phrase here');
define('LOGGED_IN_KEY', 'put your unique phrase here');
define('NONCE_KEY', 'put your unique phrase here');
define('AUTH_SALT', 'put your unique phrase here');
define('SECURE_AUTH_SALT', 'put your unique phrase here');
define('LOGGED_IN_SALT', 'put your unique phrase here');
define('NONCE_SALT', 'put your unique phrase here');
```

7. If you have DNS configured, navigate to your URL (E.g. http://cloudwire.info) and you will be presented with a welcome page. You'll need to enter some basic information and importantly create your WordPress username and password. When you are ready, select 'Install WordPress'.

**Note** If you haven't configured DNS to point to your host's IP address, you can edit your local hosts file on your client machine to test the site before going live. For example, in Windows open %windir%\System32\drivers\etc\hosts (make sure you open as an administrator), and add the hosts external IP address followed by the domain name.
Stage 7: Securing WordPress

The security of WordPress probably justifies a blog post all by itself, but I can provide some pointers to help you lock it down enough that you'll be safe from brute force attacks and limit the WordPress attack vector for vulnerabilities. Nothing is 100% secure. If there is a 0-day vulnerability in WordPress then fingers crossed IPtables, SELinux and keeping the server updated with yum-cron is enough.

I am no longer surprised when I take a look at other blogs and see how insecure they are. Here are my top 10 of security issues I would address immediately (in no particular order):

1. Excessive use of plug-ins
2. Out of date plug-ins
3. Theme vulnerabilities
4. WordPress is out of date
5. Weak admin password
6. Lack of 2-Factor authentication (Use Google Authenticator, it's free!)
7. Admin username revealed in blog posts (don't post blogs with your admin account!)
8. Using HTTP (insecure) instead of HTTPS
9. Access to wp-admin unprotected
10. Using FTP instead of more secure SSH (SFTP) for file transfers

It can be very tempting when you first start using WordPress to try out shiny new plug-ins and themes. These can be open to security vulnerabilities and provide an easy way in for somebody wanting to hack your site. A bit of vigilance can go a long way. Research known bugs and security flaws with any themes or plug-ins you download, and don't get too carried away. The more plug-ins you have installed the more chance you'll be bitten.

Most of these security best practices are just common sense and the others are well documented elsewhere. For the purposes of this guide, I will focus on the last three (8, 9 and 10).
Redirecting HTTP to HTTPS using the rewrite module

Since SSL certificates have already been installed, an SSL redirect needs to be created in .htaccess. If you go to http:// it will automatically redirect to https://.

1. Edit .htaccess in the web root directory. This file should already exist for WordPress, if not create it.

```
# vi /data/.sitehome/cloudwire.info/web/.htaccess
```

2. Add the following, replacing it with your own URL:

```
# SSL Redirect
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteCond %{SERVER_PORT} 80
RewriteRule ^(.*)$ https://www.cloudwire.info/$1 [R,L]
</IfModule>
```
Protecting Wordpress Admin (wp-admin) with .htaccess

Some might say that obfuscation doesn't provide any security benefits, so there isn't any real advantage to renaming `wp-login.php` to something else. Since I don't trust too many plug-ins, using a plug-in to rename it seems like a double-edged sword. Instead, I prefer to use `.htaccess` to prevent access from unauthorized sources to `wp-login.php` and the `wp-admin` directory.

1. Create `.htaccess` in your `wp-admin` directory

   ```
   # vi /data/.sitehome/cloudwire.info/web/wp-admin/.htaccess
   Order Deny,Allow
   Deny from all
   Allow from XX.XX.XX.XX
   Allow from XX.XX.XX.XX
   ```

   **Note** If your IP address changes and you get locked out, you will need to SSH into your server and update this.
Protecting wp-login.php with .htaccess

As an additional measure, do the same to restrict access to the wp-login.php file which resides in your web root directory.

1. Edit .htaccess in the web root directory. This file should already exist for WordPress, if not create it.

   ```
   # vi /data/.sitehome/cloudwire.info/web/.htaccess
   ```

2. Add the following, replacing X.X.X.X with your IP address.

   ```
   <Files wp-login.php>
   order deny,allow
   Deny from all
   Allow from X.X.X.X
   </Files>
   ```
Configuring SFTP for WordPress Updates

You have already learned how to use SSH for SFTP file transfers and use `chroot` to restrict access to the site directory. WordPress supports FTP, FTPS (SSL) and SSH2 connections. When you update WordPress or install a plug-in, you will be presented with the following:

![Connection Information]

**Figure 2:** Wordpress Updates using SSH

1. Change the hostname to the IP address or hostname of your server, and remember to specify the SSH port as this is not using the default (cloudwire.info:9922)

2. Choose SSH2 as the connection type

3. Enter the username and password for your site

4. Leave Authentication Keys blank (see note)

**Note**  As I mentioned in Stage 1, you’ll need to configure WordPress to use authentication keys if you disable SSH password authentication. This requires both the public and private key to be installed on the WordPress server which I think is a terrible idea. One option is to temporarily enable password authentication for SSH for when WordPress needs it.
Stage 8: Configuring SELinux

If you have made it this far, hopefully everything is working as it should. You can create multiple sites, host WordPress, transfer files using SFTP and the host is secure with IPtables.

There is a reason I have saved SELinux until last. Grab another coffee, and while you are up and about get a notepad, a pen, turn your phone to silent and get comfortable. No really, this next step is where most folk abandon SELinux and set it back to 'disabled'. You wouldn't do that now, would you? Stick with me!

If you use Linode, by default their kernel doesn't include SELinux. This is easy enough to fix, just follow their guide on using a distribution supplied kernel.

Setting Permissive Mode

Don't worry, SELinux won't enforce anything just yet!

1. First, install required packages for SELinux:

```
# yum install -y policycoreutils policycoreutils-python selinux-policy
   selinux-policy-targeted libselinux-utils setroubleshoot-server setools
   setools-console mcstrans
```

2. Next SELinux to 'permissive' mode and reboot:

Before actually enforcing SELinux policies, it is important to test SELinux with 'permissive' mode first. SELinux will log activity to /var/log/audit/audit.log, starting with "SELinux is preventing", which will be useful to troubleshoot any processes, files or directories that would otherwise be restricted.

```
# vi /etc/sysconfig/selinux
SELINUX=permissive
```

3. Reboot the server (reboot)
How SELinux Works
Before going any further, I am going to explain how SELinux works as simply as I can. It is notorious for being a complicated topic, but it really shouldn't be.

The configuration for SELinux is contained in `/etc/sysconfig/selinux`, and you can see whether it is running by using the `sestatus` or `getenforce` command. If you type `sestatus` you'll see it will either be set to disabled, enforced, or permissive. By setting SELinux to permissive, which is the first step, it won't actually enforce any policies but it will log them. This is really useful before switching it on, as it will allow you to see what it would otherwise restrict.

SELinux Context and Labels
To understand how SELinux actually works, it is important to understand the concept of SELinux context and labels. Labelling allows files, processes, sockets, directories, TCP and UDP ports and many others, to be labelled with a SELinux context. At the time of writing this, I checked and there are 4729 types listed. Don't worry, you won't be expected to memorize them all!

You can use `seinfo -t` to list all SELinux context types, but if you combine it with `grep` you can narrow the search. Give this a try:

```
seinfo -t | grep httpd_sys
```

This will show you all of the context types starting with `httpd_sys`. This will become useful later on when troubleshooting SELinux.
Figure 3: SELinux - None Shall Pass!

SELinux is analogous to a guardian. It expects files and directories to be labelled, helping classify what each file and directory is and isn’t allowed to do.

Forbidden
You don’t have permission to access / on this server.

The SELinux context for files and directories are labelled with extended attributes on the file system, and everything else is managed by the kernel. Since this is a web server running Apache, SELinux will expect a label that give Apache access to /var/www/ directory and all files within it. Take a look at /var/www/ using ls -aZ.

This will list all of the files in the working directory.

```bash
# cd /var/www
# ls -aZ
```

Note Labels use the following format: user:role:type:level, but for targeted mode, and the scope of this guide, I will ignore the others and just focus on type.

Using ls -aZ, you can see that the type for the html directory is set to httpd_sys_content_t:
In this example, `user:role:type:level` for the html directory is:

- **User:** `system_u`
- **Role:** `object_r`
- **Type:** `httpd_sys_content_t`
- **Level:** `s0`

SELinux recognizes the type label and allows Apache (httpd) to read the file.

But wait. This server is going to be configured with multiple websites and will not use the default `/var/www/html` location. Take a look at the SELinux context for the new location `/data/.sitedir/`:

```
# cd /data/.sitedir/
# ls -aZ
```

The type is set to `default_t`, and if Apache tries to read files from this location, SELinux will prevent it since it hasn’t been labelled correctly. You can see why SELinux can be a pain, but also very powerful. I will show you how to set `httpd_sys_content_t` on the new web directory location in a moment.
Booleans

A Boolean, named after George Boole (pictured left), is a true or false value. There are 301 Boolean settings at the time of writing this, and similar to using `seinfo -t` to list types, `getsebool -a` can be used to list them all. Boolean values can be set using the `setsebool -P` command (`-P` means persistent so it will be written to disk), and you can specify a 1 or 0 (on / off).

To help you understand SELinux Booleans, it may help list the ones relevant to an Apache server. Using `semanage boolean -l | grep httpd`, Boolean settings that contain the word 'httpd' and listed. This will list around 42, along with their description. Here is a sample:

**Figure 4:** George Boole, The Inventor of Boolean
There are three that stand out:

httpd_unified

httpd_unified is off by default with CentOS/RHEL 7. This means that SELinux will require the file/directory type label httpd_sys_rw_content_t if Apache is required to write to that directory. For WordPress this would include the wp-uploads directory. With previous versions of CentOS, httpd_unified is on by default, and Apache can read, write and execute on files/directories with the httpd_sys_content_t label.

httpd_can_sendmail

Another Boolean httpd_can_sendmail says that it allows Apache (httpd) to send mail. WordPress often requires this to send emails for new users, comments, and so on.

httpd_can_network_connect

By default, this is off. Setting it to 'on' will allow Apache (httpd) to connect to the network. This is required if scripts running under Apache need to connect to remote services (E.g. RSS or wget).

Note To see which httpd Booleans are on, use semanage boolean -l | grep httpd | grep -v off.grep -v inverts the results. Since the list contains the word ‘on’ for the default setting, I can’t use that so I’m saying anything that doesn’t contain the word ‘off’.
Configuring SELinux to Play Nicely with Apache and WordPress

Apache is running and a few things have changed so it's time to tell SELinux about these changes.

Step 1: Setting the Correct SELinux Context for the Web Directories
Remember I said that your website directories need to be labeled correctly? Let's configure that now on the directory containing all website directories:

```bash
# semanage fcontext -a -t httpd_sys_content_t "/*/data/.sitehome(.*)*?"
# restorecon -Rv /data/.sitehome
```

SEmanage is another useful command that can be used to read and configure settings on network ports, interfaces, SELinux modules, file context and Booleans. Using `restorecon -Rv` will set this security context recursively (\(-R\)) and will output any changes to the type (\(-v\)).

Step 2: Allow SSH on non-default port
Another change is the default SSH port from 22 to 9922. Again, using `semanage` the new port can be specified.

```bash
# semanage port -a -t ssh_port_t -p tcp 9922
```

Step 3: Allow Apache to Send Email
This time the `setsebool` command will be used to set the boolean for `httpd_can_sendmail` to 1. This will allow Apache (httpd) to send email using Sendmail or Postfix for example.

```bash
# setsebool -P httpd_can_sendmail 1
```
Step 4 (Optional): Allow Apache to Read & Write Directories with httpd_sys_content File Types

As mentioned previously, the httpd_unified Boolean is turned off by default with CentOS/RHEL 7. This does strengthen security, as any files and directories that need to be writable by Apache will require the httpd_sys_rw_content_t context. However, with WordPress this can prevent users from uploading files or installing plug-ins.

If you decide to enable this and allow read, write and execute, then use the following command:

```
# setsebool -P httpd_unified 1
```
Troubleshooting SELinux

First, check the status of SELinux and make sure it is set to permissive if it isn’t already.

```
# sestatus
```

Check that SELinux is enabled, the policy is 'targeted' and the mode is set to permissive:

```
SELinux status: enabled
Loaded policy name: targeted
Current mode: permissive
```

When troubleshooting SELinux it can be useful to clear the audit log and then rebooting. This will make it easier to read the logs and not get confused with older entries while you are troubleshooting in permissive mode.

1. Clear the audit.log and reboot

```
# > /var/log/audit/audit.log
# reboot
```

2. Use the `sealert` command to check for issues in audit.log

```
# sealert -a /var/log/audit/audit.log
```

**Note** You can run a summary of audit log reports using `aureport -a`. This will provide a summary of reports from the audit log (/var/log/audit/audit.log). You can check the logs for today using `aureport -a -ts today`.

When you have run the `sealert` command, at the top of the report you’ll see something similar to:

```
100% done
found 7 alerts in /var/log/audit/audit.log
```
The output may look very lengthy, but don't be too concerned. Once you start analyzing each report you'll see it is actually very useful. It will start with 'SElinux is preventing', and provide a confidence level for each suggested fix. I will list some common issues, along with the steps you will need to take to resolve the issue.

**Issue 1:** SElinux is preventing audispd from getattr access on the file /etc/ld.so.cache.

This provides an easy fix, detailed right there in the report:

```
If you want to fix the label. /etc/ld.so.cache default label should be ld_so_cache_t. Then you can run restorecon.
Do
# /sbin/restorecon -v /etc/ld.so.cache
```

**Issue 2:** SElinux is preventing audispd from open access on the file /etc/ld.so.cache.

This is tied to the first issue, and the same fix is suggested so will move on to the next one.

**Issue 3:** SElinux is preventing /usr/sbin/sedispatch from execute access on the file sedispatch.

This had a 100% confidence level, with the suggested fix below:

```
# ausearch -c '/usr/sbin/sedispatch' --raw | audit2allow -M my-sedispatch
# semodule -i my-sedispatch.pp
# setsebool httpd_can_network_connect=1
```

**Issue 4:** SElinux is preventing /usr/sbin/httpd from name_connect access on the tcp_socket port 9922.

This is actually required since FTP is not used to update WordPress or install plugins, and Apache needs to connect on SSH.

```
# setsebool httpd_can_network_connect=1
```
Rinse and repeat. You may need to do this a few times, implementing the fix, clearing the log, rebooting. Once you are satisfied that there are no more issues being reported by SELinux, you can now set it to 'enabled'!

```
# vi /etc/sysconfig/selinux
SELINUX=enforced
```

# reboot

Once the server reboots, log in and check `getenforce` which should report

Enforcing.
Conclusion

If you have made it this far then give yourself a pat on the back. One of the reasons I maintain my own web host is to keep my Linux skills fresh. I left my role as a Linux systems administrator back in 2006, but I assure you that running a secure web server for one or more websites is an art that needs to be mastered. Over the years I’ve encountered issues with VPS (Virtual Private Server) providers, having to deploy a custom kernel, and dealing with DDoS and other attack attempts.

My aim with this guide is to get you setup with a secure web server with CentOS 7, running WordPress for multiple websites. I don’t believe I have taken any shortcuts in regards to security, and while I agree that usability is important, I hope you can see that enabling SELinux doesn't mean 'unusable' any longer.

Key Takeaways

By following this guide, you now have:

- A basic understanding of SELinux and have it set to 'enforcing' on your web server.
- Learned how to use IPtables and use it to secure your server.
- Host your blog in WordPress using HTTPS (SSL)
- Host multiple websites using VirtualHost in Apache, and setup the new fictitious WordPress site, cloudwire.info!
- Provided website admins secure SFTP file transfer, restricting them to their own site directory.
- Learned how to use yum-cron to maintain the host with automatic updates.
About the Author

Ray Heffer is a Senior Cloud Solutions Architect and End-User Computing (EUC) specialist at VMware working within the Cloud Provider Program, a partner program consisting of over 4,200 cloud service providers worldwide. With over 20 years of experience in the IT industry, Ray is a veteran of Unix and Linux and has managed data center environments for service providers, government sector and private organizations. In 2016 he moved with his family from the United Kingdom to North Carolina, USA.

Ray is certified as a Double VCDX #122 (Data Center and Desktop) since 2013, AWS Solutions Architect Associate and holds over 15 VMware certifications in total. If you are one of the unlucky ones you will see Ray as a panelist on the VCDX program!

Follow Ray’s updates or contact him directly using Twitter @rayheffer.
Additional Resources

- YouTube: 2015 Red Hat Summit: SELinux For Mere Mortals - https://www.youtube.com/watch?v=cNoVgDqqj MM
- YouTube: Are you listening to what SELinux is telling you? - https://www.youtube.com/watch?v=Wv9kwlabdlo

Stay up to date with this guide and any updates by following me on Twitter @rayheffer