Technical Report

DomainTools

Building a Secure VPS Server Under Debian 11 ("Bullseye"): Some Notes

Joe St Sauver, Ph.D.
Distinguished Scientist

Executive Summary

Unmanaged Virtual Private Servers (VPS) are a popular way of getting what feels like "your own server" from a remote service provider. However, while you can get a Un*x virtual private server cheaply and easily today, there are still many details associated with bringing up a functional and secure system. This document is meant to help users bring up a secure-yet-usable system for purposes including personal email, serving static web pages, and miscellaneous purposes.

This document assumes technical users are working from a local Mac laptop, and are spending $10-$20/month for a remote Virtual Private System (VPS) running Debian 11 ("Bullseye"). Why Bullseye and not Bookworm? At the $10-$20/month price point, the VPS may have quite modest resources, including perhaps as little as 512MB of RAM. "Bullseye" recommends 512MB (see https://www.debian.org/releases/bullseye/amd64/ch03s04.en.html), and thus would fit in that configuration, while Bookworm's absolute minimum for a standalone system is 1024MB of RAM.
(see https://wiki.debian.org/DebianEdu/Documentation/Bookworm/Requirements).

Despite our quite-limited VPS hardware, we'll demonstrate...

- Basic authoritative DNS records that should be created in one's DNS provider's control panel
- Bringing up sshd for encrypted remote access with public key authentication and Yubikey MFA support
- Getting automatic patching set up
- Setting up ufw (with ipsets) for firewall service
- Using NTP for time synchronization
- Configuring and running a DNSSEC-enabled recursive resolver service
- Installing Postfix for email, complete with SPF/DKIM/DMARC and opportunistic TLS
- Setting up an NGINX web server to deliver static web pages with Let's Encrypt free TLS certificates
- While malware's not the problem for Un*x systems that it is for Windows, we do also install two anti-rootkit products and a system auditing tool as part of the build
- Finally, we address the reality that having only 512MB forces us to forgo many classic security tools and services we really wish we could have shoe-horned in, including staples such as ClamAV and SpamAssassin.

No security document can guarantee a "bulletproof" system, but we hope this document will provide a solid and usable foundation for those setting up a basic VPS like the one described above.

Table Of Contents

I. Introduction

> ***"If we can hit that bullseye, the rest of the dominoes will fall like a house of cards. Checkmate."***
>
> ***-- Futurama, an American animated sci-fi sitcom***

1. What We're Doing Today

Self-managed Virtual Private Servers ("VPS") have made new systems cheap for technically knowledgeable people to deploy, whether for hosting a personal ("vanity") domain, to create a remote distributed node from which to collect data, for use in working with DNSDB API or the Security Information Exchange data, or for other purposes. Whilst the process of setting up those new systems has gotten easier over time, there are still some "minor details" to attend to, particularly if you're doing a "one-off" build and won't be leveraging system automation tools in deploying that new node. This document will outline some of the considerations you may want to keep in mind while securely setting up such a new system. Of course, the precise setup you need to perform will depend on the exact capabilities you need and the services you want to run.

For the purposes of this article, we're going to assume:

Table 1: System Requirements

Attribute	Value
System Type	Small self-managed Virtual Private Server (VPS)
Network Attached?	Both IPv4 and IPv6 connectivity
X Windows (GUI) support?	Command line interface only
Firewalled or reverse-proxied?	Firewalled
Internet Accessible Applications?	ssh (with public key auth and MFA), email (smtp with SPF+DKIM+DMARC); static web pages over https (with nginx and Let's Encrypt certificates)
Internal-Only Applications?	Recursive DNS (with DNSSEC), NTP, chkrootkit & rkhunter, Lynis, backups, and DNSDB API and SIE Remote Access (not installed as part of this document)

2. "Why Is *DomainTools* Writing About Helping Users Secure New Servers of All Things?"

Some customers may want to set up and use a virtual private server to run DNSDB API or to access the Security Information Exchange via SIE Remote Access. We're also going over this content today because one of our explicit company objectives is to "help make the Internet a safer place." Part of that includes helping people have secure systems -- particularly if the system in question is used by one of our customers!

That said, let us be clear: even if a reader diligently follows *ALL* of the advice in this document, no one can warranty that you or your system will always be "perfectly safe and secure." There may be things we've inadvertently overlooked (or intentionally excluded), latent undiscovered vulnerabilities, novel new attacks, typos on our part (or yours), untrustworthy staff at the hosting provider, or many other issues that may keep a relatively-well-configured systems from being fully secure. After all, as notable cybersecurity expert Gene Spafford (https://en.wikipedia.org/wiki/Gene_Spafford) once said, *"The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards -- and even then, I have my doubts."*

We can't insist on a "Spafford-like" level of security. We need systems that authorized users can use to get their work done. Nonetheless, we believe that the steps outlined in this document will at least hopefully help you be *somewhat* more secure (and we always appreciate feedback on areas where further attention would be helpful).

3. "Is This Document Right For ME?"

Everyone comes to working with systems and networks from a different level of technical expertise (and with different objectives).

To be comfortable with this document, you'll NEED to be technically savvy. If you're NOT a technical person who likes to run his or her own system, this is not the right document for you. We're explicitly assuming that you do NOT need a "control panel" environment, for example.

Similarly, if you're not interested in a Un*x-based approach using Debian Linux, perhaps preferring MS Windows-based systems instead, this is also NOT the right document for you. Our discussion will be framed around bringing up a Debian Virtual Private Server, and we're also going to assume that you're working from a Mac laptop running the latest version of macOS (and not a Microsoft Windows laptop).

4. "Which <u>*Specific*</u> Hosting Service Is This Writeup Keyed To?"

There are many different hosting options out there, from tiny mom-and-pop independent shops to huge corporate providers. Some may be excellent (but come at a premium prices). Others may work hard to offer an acceptable service at a competitive price point. Still others might be frustrating to use, with availability issues or oversubscription problems. Everyone's needs will be somewhat different, which is why the market offers so many options. We want to respect that range of options and remain "vendor neutral" except where we have no choice but to be more specific. Factors we encourage you to review and consider when picking a VPS service:

- <u>What operating system do we want to run? Does the provider in question OFFER that option?</u> For example, some vendors may not offer Debian at all, or they may offer Debian 10 or 12, but not Debian 11. Ubuntu may be "close enough" (except when it isn't).

- <u>What applications do we want to support?</u>
    - Email? If so, will our provider allow us to send and receive port 25/TCP traffic? Has their address space been widely abused? If so, will my IP start out heavily blocklisted? Hosting email in such address space may not work very well.
    - Do we want to serve web pages? If so, just static pages, or do we need to be able to dynamic pages using a framework such as WordPress?
    - What about other applications? For example, will the provider handle authoritative DNS for our domains, or can I backup my system easily with a provider-supported backup provider?
    - Be sure to also consider any potential system overhead for any security measures we anticipate running -- substantive applications (such as simple email and web) may not require much in the way of resources, but some more intensive security technologies may be a different matter.

- <u>Given those applications, how big of a system do we need?</u> In particular, think about hardware requirements such as
    - CPU cores: do we need four, or are two enough?
    - RAM: is 2GB enough, or do we need 4GB or even more? We're going to strive to have a configuration that will work on the smallest of virtual private servers, needing no more than 512MB of RAM.
    - Disk space -- most providers use solid state disk these days, but some may offer NVMe storage (see https://en.wikipedia.org/wiki/NVM_Express) while others may still have rotating disk, perhaps at bargain prices. Do we want RAID, or are plain old disks "good enough?" How much disk is enough? Again, we'll try to minimize needs and work in just 20GB of SSD.

    - Data transfer limits per month (if any). Some providers may offer unlimited network transfers; others may have a monthly cap. Some may even apply data transfer limits to only one direction (typically outbound from the virtual private server, with inbound traffic remaining unmetered).

- What will that package cost? Are costs constant over time, or does the provider offer an initial "introductory rate" but then increase that price once you're "locked in?" Are there discounts for prepaying multiple years in advance?

- Where will my server be physically located? Location will impact network performance (since throughput's typically a function of latency and bandwidth-delay products), and choice of location may also have implications for applicable laws and civil procedures. Therefore, while it might sound "cool" to host your site somewhere "exotic" like Panama or the Seychelles, somewhere in the United States or Canada might be a more practical choice for most North Americans.

- How's the provider's network? Do they support both IPv4 and IPv6? Is their transit and peering connectivity good, or are their links underprovisioned and congested during peak load periods?

- How reliable is the provider? Do they have a well-established history? Do they seem well positioned to continue, or are there worries about their viability as a going concern? How do their uptimes look? Do they have a formal SLA (service level agreement), and what (if anything) happens if they breach it?

- What do the company's terms of service (ToS), acceptable use policy (AUP), and privacy policies look like?

- How's customer service? Do tickets get handled quickly and professionally?

You should be able to get unmanaged VPS service with full root access for **between $10-$20/month ($120-$240/year)**. The provider needs to offer Debian 11 (or at least Debian 10 or Ubuntu). We encourage you to do your own due diligence before picking a VPS provider. (We'all also note that many providers have special promotional sale pricing from time-to-time, if you're able to "time" your purchase). Some factors to consider in evaluating a potential VPS provider:

- **Providers with TOO-HIGH pricing:** You should be able to get a usable unmanaged VPS for $20/month or less (in our opinion). Some providers may attempt to avoid a race-to-the-bottom (and not market to "bargain hunters") by maintaining a minimum price slightly ABOVE $20/month (such as perhaps $25/month). That's certainly their prerogative (and you may decide you're fine with that approach), but we don't think you need to pay an extra $60/year to get adequate unmanaged VPS service.

WAY-too-high pricing for unmanaged VPS service (such as 3-5X normal unmanaged VPS service pricing) can be a sign you may have encountered a "bulletproof" ("complaints ignored") hosting outfit. (Don't conflate "bullet proof unmanaged VPS hosting" at 3-5X normal prices with legitimate MANAGED VPS service at perhaps 2X unmanaged VPS prices.)

- **Providers with TOO-LOW pricing:** This may seem counter-intuitive, but free (or nearly free) providers still need to cover their costs *somehow*, right? So, what's their business model? Some may badly oversubscribe their infrastructure; others may nickel-and-dime you for required basic services after you purchase what turns out to be a stripped-down intro package. Ultra-low cost or free providers may also lack the resources to fund a responsive anti-abuse team. Look for *reasonable* prices.

- **Providers with "non-transparent" pricing:** Some sites may provide "pricing by quotation" only. We'd exclude them because (in our opinion) they introduce too much "friction" into what should be a simple price discovery and comparison process. VPS hosting is neither high cost nor something so specialized that it makes sense for pricing to be done one-off.

- **"Shadowy" providers:** You need to be able to know and trust your provider, just as your provider should insist on knowing and trusting you, as their customer. Unfortunately, some VPS providers may intentionally attempt to conceal or obfuscate their identity. Other providers may attempt to "take a mulligan" by repeatedly renaming their service in an effort to put troubling prior behaviors behind them (we've got no problem with sites that have had legitimate merger and acquisition activity, our concern lies with companies where NOTHING has changed EXCEPT the company's name).

- **Providers with "Sketchy" customers:** If a provider accommodates sketchy customers, those customers can taint *your* site's reputation or make it hard for you to use your server to send email. Indicators of providers with sketchy customers may include:
  - Lax terms of service (including comments that "DMCA complaints will be ignored," or a willingness to host undesirable behaviors such as web comment spamming, even if email spamming is (theoretically) strictly forbidden.
  - Blocklisting by Spamhaus (one VPS provider we considered and examined had 176 SBL listings at the time we checked!)
  - Anonymous sign up options and anonymous payment channels (often featuring crypto currencies)
  - An ability for customers to temporarily get multiple IPv4 addresses from multiple disjoint netblocks -- this may often be a sign that the customer is playing games with IP reputation.
  - Data centers located in regions where corruption is endemic (or governments may be hostile to American interests).

5. *"Why Debian Linux? Why not Ubuntu, Rocky/Alma Linux or … Instead? Or FreeBSD, NetBSD, OpenBSD? Or…"*

We had to pick *some* operating system, because file system layouts vary from operating system to operating system, ditto package management tools and firewall options -- there's simply no way to avoid it. We had to select SOMETHING.

As to why *Debian* -- we know that choice of distro (like selection of a religious denomination or selection of a political party), can be an intensely personal thing, and good people can make radically different decisions on this topic. ALL distros will have advantages and disadvantages, but Debian is quite popular, community supported, stable, and known for remarkable quality control (if perhaps having a reputation for being a bit "stodgy" and "conservative").

Debian's also the primary Un*x server operating system Farsight Security (now part of DomainTools) has historically used. Then there's the fact that our focus today is on securing a remote *server* environment, not a *desktop workstation* environment, so window manager and desktop-related application considerations don't pertain. If our focus was different, we might have made a different choice. We encourage you to review some of the many web pages devoted to comparing and contrasting the various options.

Then there's the question of which *version* within a given distro. We've elected to use Version 11 (aka "Bullseye") of Debian for this document because it's the "old stable" version (released August 14th, 2021 and supported for five years thereafter).

It's minimum recommended configuration is just 512MB of RAM (see https://www.debian.org/releases/bullseye/amd64/ch03s04.en.html), and thus our modest target system configuration would be compatible. "Bookworm," the current stable vesion of Debian, needs an absolute minimum of 1024MB of RAM. (see https://wiki.debian.org/DebianEdu/Documentation/Bookworm/Requirements).

We could also have selected a really old legacy version, or a bleeding-edge-still-under-development version, but neither of those would have felt right for this project.

The Linux kernel (see `https://www.kernel.org/category/releases.html)` we're using for this example is:

```
     $ uname -a
     Linux  stsauver.com  5.10.0-23-amd64  #1  SMP  Debian  5.10.179-2
(2023-07-14) x86_64 GNU/Linux
```

This is as we want since **5.10.179-2** was listed as the most recent security release for Bullseye as of the time this document was prepared per `https://security-tracker.debian.org/tracker/source-package/linux`

Nonetheless, if you're a long-time proponent of some other operating system/version/kernel we totally get and respect that -- your system, your choice. Many of the steps described below may also apply *generally* to your environment, albeit with system-specific differences you'll need to sort out yourself.

6. Assumptions: We're Securing a System WITH Network Connectivity

Your system likely won't exist in isolation, and probably will be more-or-less directly connected to the public Internet. We're going to assume that:

- Your provider has supplied you with a new VM with a fresh install of Debian 11, probably with `cloud-init` (see `https://cloudinit.readthedocs.io/en/latest/)`
- You have a regular (non-root) account and can remotely login to that account using ssh (this might be with either a username and password initially, or with public key authentication).
- You have the root password for the system and can "su" to root, or your non-root account is set up to be able to use "sudo" for privileged administrative work.
- Your provider will supply an IPv4 address and an IPv6 prefix for your use, and network access to your system will be up at the time the server is delivered to you
- You have (or will newly register) a domain for use on your system.
- Either your registrar or your hosting provider will handle authoritative DNS for your domain; you've listed their name servers when you registered your domain; you've at least got an "A" record pointing at your server; and you can add additional DNS records as may be required.
- You've got a full backup so that if something goes catastrophically wrong with the following, you can always roll back to that. Let's get now get started with everything else on our "to-do list."

II. AUTHORITATIVE DNS

> ***"The Domain Name Server (DNS) is the Achilles heel of the Web."***
>
> ***-- Tim Berners-Lee, Inventor of the World Wide Web***

7. Verify Your System's Authoritative DNS Entries Are Present and Correctly Configured

You and other visitors will rely on authoritative DNS entries to reach your site. We're going to assume that you've already set up at least the basics for your domain's zone, and you're ready to check that setup. We like and recommend the IIS.SE Zonemaster site for checking your authoritative DNS:

```
    https://zonemaster.iis.se/en/
```

Since we're planning to use both IPv4 and IPv6, be sure to check the option to review BOTH protocols while at that site.

If your domain is ***example.com***, your IPv4 address is ***198.51.100.48***, and your IPv6 address is ***2001:DB8::90***, you might create the following records for your domain:

example.com	A	***198.51.100.48***
example.com	AAAA	***2001:DB8::90***
example.com	MX	10 mail.***example.com***
mail.***example.com***	A	***198.51.100.48***
mail.***example.com***	AAAA	***2001:DB8::9*****0**
www.***example.com***	A	***198.51.100.48***
www.***example.com***	AAAA	***2001:DB8::90***

A Note About Data Specific to Your Site and Placeholder Examples Show in This Document:
From time to time during this writeup, we'll need to refer to things specific to your new server such as your domain name or your IPv4 or IPv6 address. Those values will obviously be unique to each installation. To emphasize the fact that the values we're showing are JUST EXAMPLES and will need to be replaced with ACTUAL VALUES FOR YOUR SITE, we're bolding, italicizing and underlining them as shown above. DO NOT ACTUALLY USE THOSE bolded, italicized and underlined values as-is! Replace them with the ACTUAL VALUES FOR YOUR OWN VPS!

8. DNSSEC Signing Your Zone?

Most top level domains (TLDs) now support DNSSEC. Assuming your domain's TLD supports DNSSEC signing, you may want to use DNSSEC to protect your domain against cache poisoning attacks. (The potential downside? If you mess up signing your zone or let your keys expire, your domain may become unreachable by those who validate DNSSEC signatures.) Having DNSSEC enabled for your zone would also let you use DANE and other technologies that depend on the availability of a secure zone. However, we are NOT going to show DNSSEC signing for your own zone in this document because at least some VPS providers may not readily support DNSSEC signing for customer zones. If you **have** DNSSEC-signed your zone, you should check that your DNSSEC signing status is correct with:

```
    https://dnsviz.net/
```

We'll cover DNSSEC validation of other people's DNSSEC-signed zones in the recursive resolver section of this guide.

A Note Before Proceeding Around Command Prompts: Virtually all of the work discussed in this paper requires administrative ("root") privileges. This level of access can either be requested on a **command-by-command basis** (by prefixing commands with `sudo`), or you can become **root for the session** (by using `su -`).

Whichever approach you employ, when it's appropriate, we'll prefix the commands being run with a **hash mark (#)**. The hash mark will normally be shown unbolded below because your system prints it out, it's NOT part of what you type in. It's just a sign that you're working with root privileges, and should be careful!

III. SSHD: REMOTE ACCESS TO YOUR SERVER

> *"It's worse than you think."*
>
> *-- Mike Rowe, Host of the TV series "Dirty Jobs"*

9. ssh with ed25519-sk keys (FIDO/U2F)

In the bad old days, remote logins happened using `telnet` plus a username and an often-quite-short password -- very insecure!

- Anyone on the network path between your local terminal and your server could have sniffed your unencrypted username and password as it passed over the wire. "Game over" (from a security point of view) if/when that happened!
- If sniffing wasn't a big enough problem, short passwords were another significant vulnerability: short passwords could be attacked via dictionary attacks or via "brute force" attempts to login (e.g., an attacker could try logging in with all possible passwords).

ssh is a vast improvement. One way it helps is by encrypting all your network traffic. ssh can also help harden your auth process -- some of you may still use SSH with password authentication, but ssh supports far-more-secure public key authentication, too. On the other hand, since ssh is what virtually everybody uses for remote access to servers, it's also a popular focus for attacks, so we'll want to do whatever we can to harden it against remote attack attempts.

As part of that, we're going to use **ed25519-sk format ssh public/private keys**. OpenSSH announced support for this key format in OpenSSH 8.2 around February 2020 (see https://www.openssh.com/txt/release-8.2 at "FIDO/U2F Support"; see also "OpenSSH now supports FIDO U2F security keys for 2-factor authentication" at https://thehackernews.com/2020/02/openssh-fido-security-keys.html). ed25519-sk keys combine the cryptographic advantages of ed25519 (see https://ed25519.cr.yp.to/) with support for one-touch hardware tokens. They make it easy for us to protect our OpenSSH logins using MFA.

We're going to use Yubikeys (https://www.yubico.com/) for our "smart keys." They're secure, convenient, robust, proven and relatively affordable. [Yes, other security key alternatives may also work, but we didn't want to get distracted by buying and testing a ton of different hardware tokens. If you're using a **non**-Yubikey hard token for ssh ed25519-sk keys, we'd love to hear about your experiences, good or bad.]

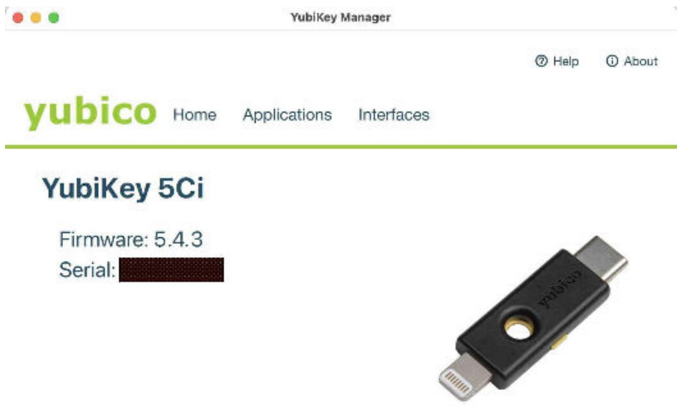
"Did You Say YubikeyS, Plural, As in MORE THAN ONE?" YES. Despite their legendary physical toughness, you MUST purchase and configure at least TWO. If you set up your system to REQUIRE multifactor authentication (but do so only for ONE Yubikey), and then something bad happens and you accidentally lose or destroy your one-and-only "magic" authenticator, you may be PERMANENTLY LOCKED OUT (unless you've retained the ability to remotely access your host console out-of-band or you have a friend or colleague who also has root access who can help you install a new key). You REALLY want to buy AT LEAST TWO!

"Which <u>Sort</u> of Yubikeys Should I Get?" Simplest rule: buy a current generation Yubikey 5 that fits the ports on the laptop you use -- USB-A, USB-C, Lightning, etc. (Do NOT purchase a FIPS 140-2 Yubikey for this application.)

Confirm That Your Yubikeys Run Supported Firmware

- Install the Yubikey Manager app on your Mac: `https://www.yubico.com/support/download/yubikey-manager/`
- Check each of your Yubikeys to ensure that they're running firmware 5.2.3 or later (as required for ed25519-sk key format per `https://developers.yubico.com/SSH/Securing_SSH_with_FIDO2.html`)
- Older Yubikeys (such as the Yubikey 4 shown below) **won't** be able to use ed25519-sk but ***can*** at least use ecdsa-sk. REALLY old Yubikeys (such as the Yubikey Standard show below) can't even do ecdsa-sk, and should be replaced.

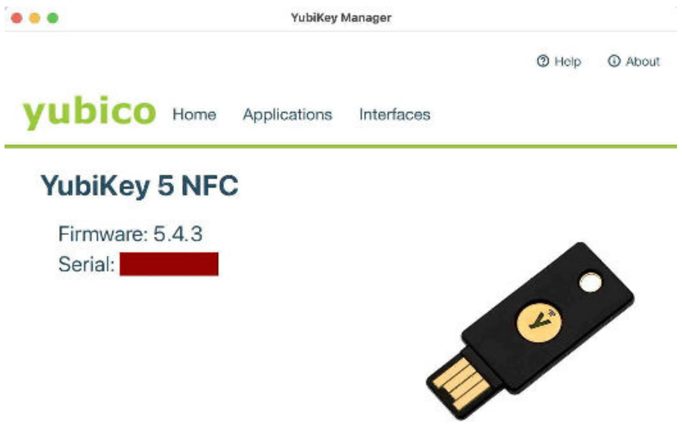
We'll now show some screenshots from the Yubikey Manager. Two are for different Yubikey 5's running firmware 5.4.3 (fine for ed25519-sk). One's from a Yubikey 4 (which can at least handle ecdsa-sk). Finally, one's from a really old Yubikey Standard that needs replacing. Note that different generations of Yubikey may superficially look quite similar, so be sure to **explicitly check** your devices.



YubiKey 5Ci

Firmware: 5.4.3

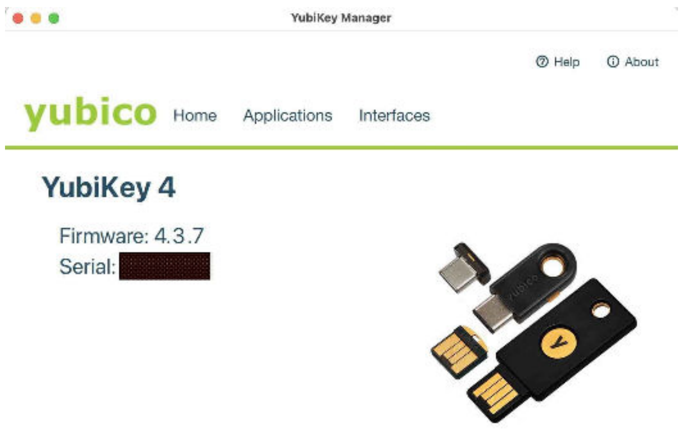
Serial: [redacted]



YubiKey 5 NFC

Firmware: 5.4.3

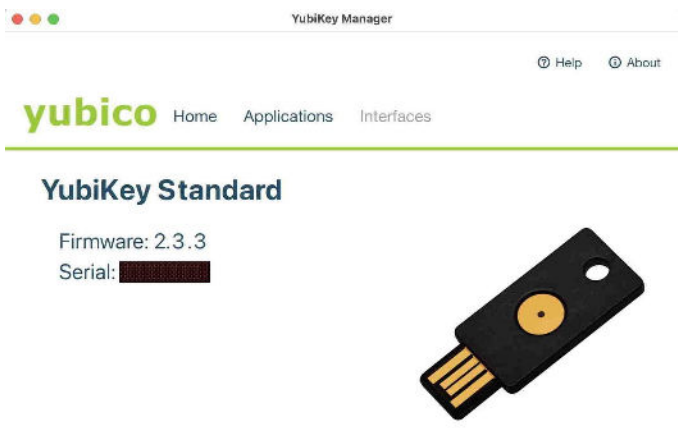
Serial: [redacted]



YubiKey 4

Firmware: 4.3.7

Serial: [redacted]



YubiKey Standard

Firmware: 2.3.3

Serial: [redacted]

Picking the Right Yubikey "Recipe:" If you attempt to research integration of Yubikeys with ssh yourself, be prepared to run into multiple "radically-different-looking recipes." All will talk about using OpenSSH with Yubikeys, but beyond that you may see little commonality between those recipes.

This complexity arises in large measure because of Yubikeys' flexibility (see https://developers.yubico.com/SSH/). In THIS document we're explicitly **NOT** going to use "yubico-pam" mode (see https://developers.yubico.com/yubico-pam/) **NOR** will we use "yubico-piv" mode (see https://developers.yubico.com/PIV/Guides/).

We **WILL** use FIDO2 Ed25519-sk public keys in "non-discoverable" form (see https://developers.yubico.com/SSH/Securing_SSH_with_FIDO2.html).

10. Creating ed25519-sk Keys

Ensure you have the latest OpenSSH and Fido2 libraries on both your workstation and your server (we'll assume you have some sort of initial remote login access to it, even before installing ssh (or improving) your server's initial default ssh installation):

- Mac:
```
$ brew update
$ brew upgrade
$ brew install openssh
$ brew install libfido2
```
- Debian:
```
# apt update
# apt upgrade
# apt install openssh-server
# apt install libfido2-1 libfido2-dev libfido2-doc fido2-tools
```

Now insert your Yubikey into your Mac, open a local Terminal window, and enter:

```
     $ ssh-keygen -a 500 -t ed25519-sk -f ~/.ssh/id_ed25519_sk -C
"ed25519_sk for keyring"
     Enter passphrase (empty for no passphrase):
     Enter same passphrase again:
     Your identification has been saved in
/Users/jsmith/.ssh/id_ed25519_sk
     Your public key has been saved in
/Users/jsmith/.ssh/id_ed25519_sk.pub
     The key fingerprint is:
     [etc]
```

Repeat this for any other Yubikeys (obviously use different filenames, such as `id_ed25519_sk_2, id_ed25519_sk_3, etc)`

"Decoding" the arguments to the above command (you can "play along at home" by looking at `$ man ssh-keygen` on your Mac):

Argument	**Purpose**
`-a 500`	"[…] KDF (key derivation function, currently bcrypt_pbkdf(3)) rounds used. Higher numbers result in slower passphrase verification and increased resistance to brute-force password cracking (should the keys be stolen). The default is 16 rounds."
`-t ed25519-sk`	"Specifies the type of key to create. […]"
`-f ~/.ssh/id_ed25519_sk`	"Specifies the filename of the key file."
`-C "ed25519-sk for keyring"`	"Provides a new comment."

NOTE: Some people may decide to skip using a password for their key if they're confident that their Yubikey gives them adequate protection.

CAUTION: If you DO decide to use a passphrase on your key, DON'T forget it, because it **CAN'T** be recovered if lost or forgotten.

CAUTION: If you supply a filename for your new key and that filename is ALREADY being used, you may be asked to confirm that you want to overwrite those files. Generally, **DO NOT** do that -- destruction of your old keys may lock you out of other systems that expect you to still have those keys available! Pick a unique NEW filename instead!

CAUTION: If you're using something OTHER THAN a Debian 11 ("Bullseye") server, confirm that ed25519-sk format keys are supported on the target server environment (some server or cloud environments may not yet support them, see for example https://learn.microsoft.com/en-us/troubleshoot/azure/virtual-machines/ed25519-ssh-keys)

11. Protecting Your ssh Private Key on Your Local Workstation

Because of the sensitivity of your ssh private key, double check that your workstation's directory's protections are appropriate (there should be no "group" or "other" access):

```
$ ls -ld ~/.ssh
drwx------ 32 jsmith staff 1024 Jan 30 14:55 .ssh
```

If your `~/.ssh` directory's permissions don't look as shown above in the bold green text, fix them with:

```
$ chmod 0700 ~/.ssh
```

Recheck the permissions to ensure that they are correctly set.

Now check the file permissions on the `id_ed25519_sk` private key file we created. The associated **.pub** file is basically fine as long as it isn't world or group WRITEABLE, but the associated **private key** file MUST **NOT** BE WRITEABLE ***OR*** **READABLE** BY ANYONE EXCEPT THE USER HIM OR HERSELF:

```
$ cd ~/.ssh

$ ls -l id_ed25519_sk*
-rw------- 1 jsmith staff 561 Jan 30 14:48 id_ed25519_sk
-rw-r--r-- 1 jsmith staff 145 Jan 30 14:48 id_ed25519_sk.pub
```

If those files have incorrect permissions, fix them with:

```
$ chmod 0600 ~/.ssh/id_ed25519_sk
$ chmod 0644 ~/.ssh/id_ed25519_sk.pub
```

Repeat for any additional keys you created. If you had to tweak permissions, recheck them for correctness after doing so.

12. Getting Your new ed25519-sk Public Key(s) up Onto Your New Server

Your initial public key may get uploaded and installed on your new VPS as part of the provider-specific server setup process, or you may start off with conventional password authentication, meaning you'll then need to upgrade to public key authentication yourself.

- The *"upload and install as part of the provider-specific setup process"* will vary from provider to provider. See your provider's documentation for details of if/how you can upload ssh keys during provisioning for your VPS.
- The "start with password authentication and then upgrade to public key authentication" process typically relies on use of the `ssh-copy-id` command:

  ```
  $ ssh-copy-id -i ~/.ssh/id_ed25519_sk.pub jsmith@example.com
  ```

  If you created key pairs for multiple Yubikeys, repeat this step for any backup keys.

- Sometimes the `ssh-copy-id` command may fail to work. If that happens, ssh into your server using your regular (non-root) username and password and copy-and-paste the contents of `~/.ssh/my_id_ed25519_key.pub` from your Mac into `~/.ssh/authorized_keys` on the new server.

  If you're using multiple Yubikeys, also cut and paste the other public keys (for your backup Yubikeys) into the `~/.ssh/authorized_keys` file on your new server, one per line.

  Check to make sure this file is not writable by anyone except the user him- or herself.

13. Configuring Your Mac's ssh Client to Use the New ed25519-sk Key Pair for the New Remote Server

Using your favorite editor on your local workstation edit `~/.ssh/config` (add the following lines at the top, then save and exit):

```
host example.com                          ← substitute your new server's name
  User jsmith                             ← substitute your username on the new server
  IdentityFile ~/.ssh/id_ed25519_sk       ← primary key
# IdentityFile ~/.ssh/id_ed25519_sk_2     ← backup key
# IdentityFile ~/.ssh/id_ed25519_sk_3     ← tertiary key
```

If you eventually want to use one of your backup keys, comment out the primary key and uncomment a backup key.

14. Test and Confirm That Your New ed25519-sk ssh Keys Actually Work

CAUTION: IT IS ABSOLUTELY CRITICAL THAT YOU CONFIRM YOUR NEW KEYS WORK BEFORE REMOVING ANY OLD KEYS FROM THE AUTHORIZED KEYFILE ON THE SERVER!

ALSO NOTE: When using your Yubikey, the Yubikey MUST be already in the computer, you WON'T be prompted to insert it!

When using ed25519-sk keys, you WILL be prompted to "confirm user presence" when logging in. That's geekspeak for "touch your Yubikey to authenticate." For example:

```
$ ssh jsmith@sample.com
Confirm user presence for key ED25519-SK SHA256: [etc]
[touch the Yubikey in the port]
User presence confirmed
[etc]
```

You should now be logged in. Yay! Log out.

Now test your **backup** Yubikey. Change `~/.ssh/config` to point at your backup Yubikey. Remove the primary Yubikey from you Mac and replace it with your backup Yubikey. Confirm that you can use it to login with that one, too. Log out.

If you've got **another backup** Yubikey, tweak `~/.ssh/config`, swap in the next Yubikey, and confirm you can login with it, too. If everything's good, tweak your `~/.ssh/config` back to your primary key and store your backup keys somewhere safe, such as with your passport or other important documents.

After a few days, once you've confirmed that all your Yubikey-enabled keypair's are fully functional and working smoothly, you can remove any previous non-Yubikey public key from `~/.ssh/authorized_keys` on the server. Just be sure you remove the RIGHT (non ed25519-sk) KEYS!

15. Troubleshooting

"I Can't Connect at All!" Have you configured the sshd configuration with a port OTHER than the default port 22? We haven't talked about that yet, but we will do so later. If you've already done so, and you've got a firewall (such as ufw) running, that firewall may need to have a hole opened for the OTHER port. In the root window you've still got open (you DID leave a root window open, right?), and assuming you were going to use port 754 for sshd, try:

```
# ufw allow 754/tcp
```

Then go to a new terminal window on your local system and try sshing in to the remote server again. [We'll talk more about changing the port for sshd and using ufw later in this document]

"I Can Connect via ssh Fine When I'm On My Regular User Account, But Not When I'm Root on my Local Workstation!"

Confirm who you're currently "running as:"

```
# whoami
root
```

If you are indeed running locally as root, remember: direct remote root logins may already be disabled on your VPS (and if not, we'll disable them later). So, if you tried just logging in by saying **`ssh`** ***`example.com`***, ssh will assume you want to login to the same account as you're currently using, so you'd be (implicitly) trying to login to the remote host as root (which is (or shortly will be) disabled).

However, even if you ARE explicitly trying to login to a regular user's remote account while root, and you've correctly specified a user account by saying ***`ssh jsmith@example.com`***, if you're running as root, ssh will still try to use the root user's **`.ssh/config` file, not the regular user account's `.ssh/config` file.** (and do you have the right alternative ssh port number specified in the root user's `.ssh/config` file just as you do in the regular user's account, if you're using a non-standard ssh port number?)

Note, too, that the root user's `.ssh/` directory also likely won't have the **private key** that's normally present in the regular user's `.ssh/` directory for that system.

"Something Else Isn't Working Right…" You may want to look for clues on the workstation or on the server or both.

- On the client side, try running `ssh` in verbose mode (for example `$` **`ssh -vvv jsmith@sample.com)`**.
- On the server side (in your still-logged-in root window), check `/var/log/auth.log` for any `sshd`-related entries that may give you a "lead."

"When Logging In, I See:

```
     'sign_and_send_pubkey: signing failed for ED25519-SK
"/Users/jsmith/.ssh/id_ed25519_sk":
     device not found
     jsmith@sample.com: Permission denied (publickey).
```

That usually means "Your Yubikey isn't plugged in." Plug it in and try it again.

"My Old Yubikey Can't Use ed25519-sk!"

You could try using **ecdsa-sk** instead of **ed25519-sk**, but a better idea would be to replace the old key with a pair of current keys that can support ed25519-sk. Alternatively, you may decide that **password protected** ed25519 keys (completely without security key MFA protection) are strong enough on their own WITHOUT using Yubikey FIDO/U2F.

We're now going to harden our sshd configuration, beginning with editing our /etc/ssh/sshd.conf file.

16. *An Aside:* Fixing Default vi/vim Options in /etc/vim/vimrc

Before editing our sshd config files, we might want to adjust some vi/vim default settings, particularly settings that may prevent normal "Mac-style" cut-and-paste operations. If you like how vi/vim works by default in Debian, that's fine, you can skip the following. However, if vi/vim ISN'T working the way "plain vi/plain vim" classically "should work," you might want to try the following settings for a "more traditional" vi/vim environment. Become root, then edit `/etc/vim/vimrc`

```
" make vim more vi-compatible
set compatible

" disable mouse support (vi should NOT try to be a GUI editor!)
set mouse=
set ttymouse=

" disable filetype-specific behaviors
filetype off
filetype plugin off
filetype indent off
syntax off

" use basic (rather than "fancy") search
set noincsearch
set nohlsearch
set noignorecase
set nosmartcase

" look-and-feel items
colorscheme default
set title
set titlestring=%t
set showmode
set noruler
set nospell
set nowrap
set noautoindent
set nonumber
set nocursorline
set nocursorcolumn

" prevent $VIMRUNTIME/defaults.vim from being loaded.
let skip_defaults_vim = 1
```

Save the configuration file and try editing with vi or vim -- do things feel a "little more normal" now? If so, you're ready to tackle tweaking `/etc/ssh/sshd_conf`

17. Tweaking `/etc/ssh/sshd_config` to Harden the Server's sshd Configuration

Virtually all "securing ssh" tutorials include recommendations around tweaking the server's `/etc/ssh/sshd_config` file, so we'll discuss it, too. **BE CAREFUL.** Tweaking `/etc/ssh/sshd_config` MAY increase sshd's security (OR incorrect tweaks may end up **locking you out** or leaving you **insecure** rather than more secure). Before implementing anything in this particular section, please note the following key points:

- The following tweaks are just **suggestions** for your consideration. Ultimately, **YOU** must take responsibility for any changes you make to the default settings. We urge you to PERSONALLY RESEARCH ALL CHANGES before implementing them. One description of settings can be seen at `https://www.ssh.com/academy/ssh/config`
- Keep a pristine copy of your original configuration file as a backup. You may also want to save a copy of the output from:

```
# sshd -T
```

- Keep a root window open until you've confirmed that everything's working the way you expect it to.
- After making changes (but before restarting sshd), syntax check `/etc/ssh/sshd_config` for "sanity:"

```
# sshd -t
```

Resolve any errors or warnings from the above command, THEN and ***ONLY*** THEN proceed to...

```
# systemctl restart sshd
```

Table 3. /etc/ssh/sshd_config Settings

Setting	Explanation or reference
`PubkeyAuthentication` **`yes`**	**Public key authentication MUST be allowed!**
`PasswordAuthentication` **`no`**	Disable password-based authentication. **Important: Only do this after public key auth is verified to be running 100% OK!**
`Protocol` **`2`**	Only use sshV2. See https://www.kb.cert.org/vuls/id/684820
`Port [pick a port other than 22]`	Network scanners may still discover (and try to attack) your new non-standard port, but it may still lower the noise level by an order of magnitude or two. If you DO choose to use a different port, keep a root window open until you're sure everything's working the way it should! Other things to remember: (a) Make a note of the server port number you pick! (b) On the server, as root, **be sure** to let the ssh traffic in past the firewall: ***ufw allow*** **my_ssh_port_number*/tcp*** (b) On the client, in ~/.ssh/config, set ***Port*** **my_port_number** for the correct Host (c) Test and check that you can login to the new server on the new port number (d) If you're using fail2ban (we won't be), its port number info would also need to be adjusted (if you were surgically targeting just the default port 22 only)
`PermitRootLogin` **`no`**	Require regular user login THEN su or sudo (instead of direct remote root login)
`PermitEmptyPasswords` **`no`**	Disallow ssh login to any accounts that lacks a password
`StrictModes` **`yes`**	Require correct permissions on public/private key files
`LogLevel` **`VERBOSE`**	Enhance the amount of logging to allow auditing of ssh key usage by users
`ChallengeResponseAuthentication` **`no`**	Disable unused auth/access method
`KerberosAuthentication` **`no`**	"
`GSSAPIAuthentication` **`no`**	"
`HostbasedAuthentication` **`no`**	"
`X11Forwarding` **`no`**	Disable unneeded forwarding/tunneling
`AllowAgentForwarding` **`no`**	"
`AllowTcpForwarding` **`no`**	"
`PermitTunnel` **`no`**	"
`GatewayPorts` **`no`**	"
`IgnoreRhosts` **`yes`**	Keep .rhosts and .shosts files from being used for HostbasedAuthentication
`IgnoreUserKnownHosts` **`yes`**	Keep ~/.ssh/known_hosts from being used for HostbasedAuthentication
`PermitUserEnvironment` **`no`**	Avoid untrustworthy user env settings (also comment out any AcceptEnv lines)
`Compression` **`no`**	https://cisofy.com/lynis/controls/SSH-7408/
`MaxAuthTries` **`2`**	"

`TCPKeepAlive` **`no`**	"

`sshd Setting We're` <u>`Not`</u> `Suggesting You Tweak`	**Explanation/Discussion**
`AllowUsers [config]`	Not relevant to this configuration (we already have a limited numbers of users)
`LoginGraceTime 20`	Prevent attackers from leaving connections open in resource exhaustion attacks
`HashKnownHosts Yes`	Only relevant if the server is compromised and attackers are trying to pivot to other hosts by leveraging trust relationships.
`ClientAliveCountMax 3`	If this threshold is reached while client alive messages are being sent, sshd will disconnect the client
`ClientAliveInterval 300`	Idle timeout: <N> seconds.
`Banner [filename]`	Some jurisdictions may require banner notification to be provided

Reminder:

- Keep a root ssh window open on your server until you've confirmed that everything's working the way you expect it to.
- After making changes (but before restarting sshd), check the `/etc/ssh/sshd_conf` syntax for superificial "sanity:"

```
# sshd -t
```

Resolve any errors or warnings from the above command, THEN and ***<u>ONLY</u>*** THEN proceed to...

```
# systemctl restart sshd
```

18. Hardening sshd's Technical Cryptography with `ssh-audit`

In addition to major options, sshd cryptography can use weaker (or stronger) ciphers. `ssh-audit` does a nice job of identifying subtle sshd cryptographic weaknesses.

- Install `ssh-audit` on your Mac workstation with:

```
$ brew install ssh-audit
```

- Now run `ssh-audit` against your VPS (obviously substitute your actual host name for `example.com` and substitute the port number you're actually using for `sshd` for the illustrative value of `12345`):

```
$ ssh-audit --verbose example.com -p 12345
```

- Cryptographic weaknesses may be highlighted in the output. Consider fixing those issues on your server as described at `https://www.sshaudit.com/hardening_guides.html#debian_11`
- Rerun `ssh-audit` from the workstation against the server to confirm that the previous weaknesses have been fixed.

IV. PATCHING

> ***"The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at and repair."***
>
> ***-- Douglas Adams, Author of The Hitchhikers Guide to the Galaxy***

19. We Now Have Secure Access to Our VPS via ssh -- Now What? ***PATCH!***

Now that we have secure ssh access to our VPS, we're ready to do more. Let's begin by ensuring that our system is patched up to date. Let's begin by ensuring we have the right source repositories configured in `/etc/apt/sources.list`

```
### Options defined in https://wiki.debian.org/SourcesList
### https://wiki.debian.org/DebianStable
deb http://deb.debian.org/debian bullseye main non-free
deb-src http://deb.debian.org/debian bullseye main non-free
### https://www.debian.org/security/
deb http://security.debian.org/debian-security bullseye-security main
contrib non-free
deb-src http://security.debian.org/debian-security bullseye-security
main contrib non-free
### https://wiki.debian.org/StableUpdates
deb http://deb.debian.org/debian bullseye-updates main contrib non-free
deb-src http://deb.debian.org/debian bullseye-updates main contrib
non-free
### https://backports.debian.org/
deb http://deb.debian.org/debian bullseye-backports main contrib
non-free
deb-src http://deb.debian.org/debian bullseye-backports main contrib
non-free
### https://wiki.debian.org/StableProposedUpdates
deb http://deb.debian.org/debian/ bullseye-proposed-updates main
contrib non-free
deb-src http://deb.debian.org/debian/ bullseye-proposed-updates main
contrib non-free
```

Eventually (if we keep going through this guide) we'll also want to have in `/etc/apt/sources.list.d/`:

```
# cat nginx.list
### https://nginx.org/en/linux_packages.html#Debian
deb https://nginx.org/packages/debian bullseye nginx
deb-src https://nginx.org/packages/debian bullseye nginx

# cat cisofy-lynis.list
### https://packages.cisofy.com/community/#debian-ubuntu
deb https://packages.cisofy.com/community/lynis/deb/ stable main
```

Note that:

- In addition to the primary bullseye repository, we've also configured the `-security`, `-updates`, `-backports` and `-proposed-updates` repositories
- We've also asked for the `main`, `contrib` and `non-free` components `(see https://wiki.debian.org/SourcesList)`
- Depending on the system's uses, you may have additional vendor repositories listed. We show ones for `nginx` and `lynis`. We are NOT performing vendor pinning (for more on pinning, see https://douglasrumbaugh.com/post/apt-pinning/)
- Speaking of `lynis`, if you don't need translation support (you don't need non-English language support), please add a line to `/etc/apt/apt.conf.d/99disable-translations` reading:

```
Acquire::Languages "none";
```

Now, ensure the required software packages are available:

```
# apt install curl gnupg2 ca-certificates lsb-release
debian-archive-keyring
```

And you may need to install signing keys (beyond those in the debian-archive-keyring), at least if downloading updates from nginx or cisofy vendor repositories:

```
### apt-key is deprecated, but see
https://packages.cisofy.com/community/#debian-ubuntu
# apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
013baa07180c50a7101097ef9de922f1c2fde6c4
### as shown at https://nginx.org/en/linux_packages.html#Debian
# curl https://nginx.org/keys/nginx_signing.key | gpg --dearmor \
| sudo tee /usr/share/keyrings/nginx-archive-keyring.gpg
>/dev/null
```

Once the repositories and keys have been configured, update and upgrade your server's software with:

```
# apt update
# apt upgrade
```

20. Enable Automatic Updates

Let's also enable **automatic updates.** Some may prefer NOT to do this, but we prefer not having to remember to check and update:

```
# apt install unattended-upgrades apt-listchanges
```

The configuration file `/etc/apt/apt.conf.d/50unattended-upgrades` should have at least the following lines uncommented:

```
Unattended-Upgrade::Origins-Pattern {
"origin=Debian,codename=${distro_codename},label=Debian";

"origin=Debian,codename=${distro_codename},label=Debian-Securit
y";

"origin=Debian,codename=${distro_codename}-security,label=Debia
n-Security";
}
Unattended-Upgrade::Mail "jsmith@example.com";
Unattended-Upgrade::MailReport "on-change";
Unattended-Upgrade::Remove-New-Unused-Dependencies "true";
Unattended-Upgrade::Remove-Unused-Dependencies "true";
Unattended-Upgrade::SyslogEnable "false";
Unattended-Upgrade::SyslogFacility "daemon";
Unattended-Upgrade::Verbose "false";
```

And the configuration file `/etc/apt/apt.conf.d/20auto-upgrades should have:`

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

Sometimes after automatic updates, you may be prompted in the auto-update email receipt to reboot. Be sure to do so.

V. FIREWALL USING UFW AND IPSET

> *"We don't trust you."*
>
> *-- David Koh, Cyber Security Agency of Singapore*

21. Enable ufw for Firewall Protection

Debian 11 has a number of firewall options. The default (and most common) firewall option is ufw ("Uncomplicated Firewall"). There are some subtleties to this "uncomplicated" firewall option that are sometimes overlooked, including:

- Ufw handles IPv4 and IPv6 traffic independently. On dual-homed systems with connectivity via both IPv4 and IPv6, administrators may sometimes forget to ensure that both address families have consistent firewall policies.
- When blocking traffic, traffic can be REJECTed or DROPed. Both are legitimate choices, but it can be important (from a security point of view) that administrators pick the right option (one discussion of the difference between the two can be seen at `https://www.chiark.greenend.org.uk/~peterb/network/drop-vs-reject)`
- Rule persistence: ufw is built on top of iptables, but there are important differences between the two. For example, by default, iptable rules are NOT persistent ("will NOT survive a reboot"), but users can install and use `iptables-persistent` to change that if desired (or administrators can manually save and restore iptable rulesets). Ufw rules, on the other hand, when added via the command line interface, will be persistent by default.
- Rule ordering and processing: ufw evaluates rules sequentially, and stops after the first match. For that reason, rule order matters, and specific exception rules should appear before default/general-case rules. Many introductions to ufw fail to adequately stress this point, or omit explaining how to use things like `ufw insert 1` to prepend rules to the top of the ruleset rather than just having them be postpended to the bottom of the ruleset by default.
- Rule Scalability: Some sites may be interested in performing fine-grained filtering involving a large number of IPs or CIDR netblocks. Attempting to do that in ufw via a large number of "`ufw deny from`" rules will not go well. Fortunately, with the use of `ipset` (and `ipset-persistent)`, ufw can handle thousands of specific CIDR filters with ease, as we'll show below.
- Ufw logging: Logging can be enabled or disabled. Some may find firewall logging information to be just irritating noise (or even a vector for exhausting a system's disk space its own right), while others couldn't imagine living without that intelligence. Fortunately, ufw logging can be configured on or off to suit the admin's taste.

With that for context, here's what we'd suggest you consider trying for an initial ufw configuration for your server. We'll reiterate here for the record that, as always, YOU are ultimately responsible for whatever configuration you choose to deploy (or not deploy).

If necessary, install the required packages:

```
# apt install iptables
# apt install ipset
# apt install ufw
# apt install iptables-persistent
# apt install netfilter-persistent
# apt install ipset-persistent
```

Confirm that IPv6 is enabled in /etc/default/ufw:

```
IPV6=yes
```

Important: DO NOT reload or restart ufw yet!

Ensure that you've allowed ssh traffic inbound through the firewall to port 22/tcp with:

```
# ufw allow ssh
```

OR **if you're using a non-standard port for ssh** (such as, hypothetically, port 754/tcp), **allow inbound traffic to that port**:

```
# ufw allow 754/tcp
```

Important: Keep a root window open to your server until you've confirmed that the server is responding as expected.

If you're also going to run other services, such as an email server (normally SMTP listens on port 25/tcp) or a web server (web servers normally offer http on 80/tcp and https on 443/tcp), you may also want to take this opportunity to add rules for those services, too:

```
# ufw allow 25/tcp
# ufw allow 80/tcp
# ufw allow 443/tcp
```

Confirm everything looks good (note: if you're using a non-standard ssh port, you should see it listed in the following table, too):

```
# ufw status
[…]
To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
25/tcp                     ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
443/tcp                    ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
25/tcp (v6)                ALLOW       Anywhere (v6)
80/tcp (v6)                ALLOW       Anywhere (v6)
443/tcp (v6)               ALLOW       Anywhere (v6)
```

If you've accidentally added a firewall rule you didn't intend to add, remove it. The easiest way to remove a rule is by its number. Get rule line numbers with the `ufw status numbered` command:

```
# ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 25/tcp                     ALLOW IN    Anywhere
[ 2] 80/tcp                     ALLOW IN    Anywhere
[ 3] 443/tcp                    ALLOW IN    Anywhere
[ 4] 22/tcp                     ALLOW IN    Anywhere
[ 5] 25/tcp (v6)                ALLOW IN    Anywhere (v6)
[ 6] 80/tcp (v6)                ALLOW IN    Anywhere (v6)
[ 7] 443/tcp (v6)               ALLOW IN    Anywhere (v6)
[ 8] 22/tcp (v6)                ALLOW IN    Anywhere (v6)
```

Then, to delete a single rule, enter:

```
# ufw delete unwanted_rule_number
```

If you need to delete MULTIPLE rules, either "work upward from the bottom" (deleting the bottommost rule that's to be deleted first, then deleting the next lowest one, etc., so that rule shifts don't change the numbering of rules you may still be planning to prune) OR recheck `ufw status numbered` to see the new rule numbering after every deletion gets made.

When everything finally looks good, reload ufw:

```
# ufw reload
Firewall reloaded
```

Confirm that you're able to create a new ssh session from your laptop to the VPS.

Also confirm that you're seeing ufw log entries in `/var/log/ufw.log` -- if not (and you need log data), enter:

```
# ufw logging on
```

22. Configure ipset and ipset-persistent

Ufw is now up and running. Assume, however, that we've got a list of CIDR netblock sources you're interested in blocking. If we try adding those as rules to ufw one-rule-at-a-time directly, you'll find that process is (a) tedious and (b) doesn't work very well for a signicant number of CIDRs (as in, "your system may become catatonic or impossible to login-to"). Let's use `ipset` to ***scalably*** handle this, instead. We've already added the **`ipset`** and **`ipset-persistent`** packages as part of getting set up in the previous section.

We'll need two ipset nethashes, one called **`cidr-blocklist`** (for IPv4 CIDRs), and a second one called **`cidr-blocklist-ipv6`** (for IPv6 CIDRs):

```
# ipset create cidr-blocklist nethash
# ipset create cidr-blocklist-ipv6 nethash family inet6
```

When created this way (with default values for `hashsize` and `maxelem`), those ipsets should be able to accommodate up to 65536 rules. If you need to be able to handle more rules than that, check out **`man ipset`** for details on how to tweak **`hashsize`** & **`maxelem`**

Once you've created those ipsets, populate them. For example (purely hypothetically), perhaps you want to add rules blocking all connection attempts from China Telecom (AS4134). They originate 704 IPv4 prefixes (as listed at `https://bgp.he.net/AS4134#_prefixes)`, and 606 IPv6 prefixes (as listed at `https://bgp.he.net/AS4134#_prefixes6)`

The first part of doing this is purely about building and loading the ipset rules…

- Using Firefox, copy and paste the IPv4 prefixes from the bgp.he.net IPv4 prefix tab table into a file called **`temp.txt`**
- Use the Un*x **`awk`** command to keep only the first column from that file, `sort` and uniquify (deduplicate) those results. Save the remains to a new file called **temp2.txt**:

  ```
  $ awk '{print $1}' < temp.txt | sort -u > temp2.txt
  ```

  Most of the results in `temp2.txt` will be CIDR prefixes, but there may also be some country names or other miscellaneous text junk. Using your favorite text editor, remove any lines (typically near the top or bottom) that aren't CIDR prefixes.
- Because the data we just collected is based on BGP routing data, we may be able to minimize the number of rules we need for ipset by combining adjacent CIDR netblocks, or by eliminating overlapping prefixes (for example, there may be a large covering route plus a set of more specific prefixes -- we don't need both). We'll condense our list of IPv4 CIDRs with `cidrmerge (see https://github.com/kfeldmann/cidrmerge):`

  ```
  $ cidrmerge < temp2.txt > temp3.txt
  ```

That takes us down from 704 CIDR netblocks to just 406 -- a nice savings!

```
$ wc -l temp2.txt temp3.txt
  704 temp2.txt
  406 temp3.txt
```

- Once again using our favorite text editor, add the following (including a literal space at the end of the string shown) to each line in temp3.txt:

  ```
  ipset add cidr-blocklist 
  ```

  temp3.txt will then look like:

  ```
  ipset add cidr-blocklist 1.48.0.0/15
  ipset add cidr-blocklist 1.50.0.0/16
  ipset add cidr-blocklist 1.68.0.0/14
  ipset add cidr-blocklist 1.80.0.0/13
  ipset add cidr-blocklist 1.180.0.0/14
  ipset add cidr-blocklist 1.192.0.0/13
  ipset add cidr-blocklist 1.204.0.0/14
  ipset add cidr-blocklist 14.16.0.0/12
  ipset add cidr-blocklist 14.104.0.0/13
  ipset add cidr-blocklist 14.112.0.0/12
  ipset add cidr-blocklist 14.134.0.0/15
  ipset add cidr-blocklist 14.144.0.0/12
  ipset add cidr-blocklist 14.208.0.0/12
  ipset add cidr-blocklist 27.16.0.0/12
  ipset add cidr-blocklist 27.54.224.0/19
  ipset add cidr-blocklist 27.128.0.0/15
  ipset add cidr-blocklist 27.148.0.0/14
  ipset add cidr-blocklist 27.152.0.0/13
  ipset add cidr-blocklist 27.184.0.0/13
  ipset add cidr-blocklist 27.224.0.0/14
  ipset add cidr-blocklist 36.1.0.0/16
  ipset add cidr-blocklist 36.4.0.0/14
  ipset add cidr-blocklist 36.16.0.0/12
  [etc]
  ```

  Save that file. Run it by saying:

  ```
  # sh -x temp3.txt
  ```

  This should run quickly, and add all the IPv4 prefixes to the ipset. Now run

  ```
  # netfilter-persistent save
  ```

- If you're dual homed, repeat the above process for the IPv6 prefixes. Things to note:

- The `cidrmerge` command we used for the IPv4 prefixes isn't "IPv6 aware", but there are other similar packages that can be used, perhaps https://github.com/zhanhb/cidr-merger (or you can just leave the IPv6 prefixes unaggregated
  -- there tends to be fewer of them, so merging/deduplicating them may not be as big of a deal).

- Instead of prefixing these lines with `ipset add cidr-blocklist` you'll want to prefix the IPv6 lines with `ipset add cidr-blocklist-ipv6`

● If you want to confirm the rules you've just loaded into ipset, try:

```
# ipset list cidr-blocklist
# ipset list cidr-blocklist-ipv6
```

● To keep an eye on the total size of your rulesets (remembering your 65K rule default limit):

```
# ipset list cidr-blocklist | wc -l
# ipset list cidr-blocklist-ipv6 | wc -l
```

The second part of this process is about connecting the ipset(s) with iptables (running "underneath" ufw), making the rulesets persistent, and ensuring that `netfilter-persistent` is routinely started (it reloads the ipsets at reboot);

```
    # iptables -I INPUT -m set --match-set cidr-blocklist src -j DROP
    # iptables -I FORWARD -m set --match-set cidr-blocklist src -j
DROP

    # ip6tables -I INPUT -m set --match-set cidr-blocklist-ipv6 src
-j DROP
    # ip6tables -I FORWARD -m set --match-set cidr-blocklist-ipv6 src
-j DROP

    # netfilter-persistent save

    # dpkg-reconfigure ipset-persistent
    # dpkg-reconfigure iptables-persistent

    # systemctl enable netfilter-persistent
    # systemctl start netfilter-persistent
    # systemctl status netfilter-persistent
```

23. Verify What's Open and Closed

Let's now confirm our firewall configuration using nmap. Begin by ensuring we have `nmap` installed on our Mac workstation:

```
      $ brew install nmap
```

NOTE: Please do NOT nmap or otherwise scan servers owned by others without the other server-owner's permission. Scanning often serves as a reconnaissance step prior to an attack and potential compromise, and will normally be viewed as hostile. As such, it may result in the scanning IP (or even a larger network range) getting blocked, complaints to your provider, or potentially even criminal "hacking" charges in some jurisdictions. You're normally going to be fine scanning your own VPS running on a dedicated IP address, but please refrain from scanning other people's systems or shared IP addresses.

We should now do a basic scan of our VPS to ensure the world sees our VPS the way we expect. From our Mac laptop:

```
# nmap -4 -sV example.com
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-11 14:02 PST
Nmap scan report for example.com (198.51.100.48)
Host is up (0.074s latency).
Other addresses for example.com (not scanned): 2001:DB8::90
Not shown: 100 filtered tcp ports (no-response)

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 68.42 seconds
```

In this case, using just a basic service discovery scan, the **sshd server** (which we'll assume we're running on a non-standard port) wasn't seen. A more exhaustive search would likely have detected it. Note, too, that we only scanned the **IPv4** address for the server by default (we can't scan both IPv4 and IPv6 at once).

Repeating the process with **`-6`** to scan the IPv6 interface to the server:

```
$ nmap -6 example.com
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-11 12:30 PST
Nmap scan report for example.com (2001:DB8::90)
Host is up (0.076s latency).
Other addresses for example.com (not scanned): 198.51.100.48
Not shown: 100 filtered tcp ports (no-response)

Nmap done: 1 IP address (1 host up) scanned in 53.17 seconds
```

Let's move on.

VI. NTP

"Time is an illusion."

-- Albert Einstein, Physicist

24. NTP (Network Time Protocol)

Accurate time is important for logging, multifactor authentication, and DNSSEC among other things. Ideally, if we had roof access for a GPS antenna, we'd synchronize our server to GPS/GNSS satellite time with a commercial hardware GPS time server or something like https://github.com/domschl/RaspberryNtpServer [However, it's really not very realistic to expect roof access for a $20/month VPS!]

Staying realistic, we're going to use NTP (Network Time Protocol) instead, see https://support.ntp.org/Main/WebHome

Begin by confirming your time zone is set to UTC, like all good servers:

```
# cat /etc/timezone
Etc/UTC
```

Install the ntp server:

```
# apt install ntp
# apt install ntp-doc
```

The ntp configuration file is at `/etc/ntp.conf`

Per `https://www.ntppool.org/zone/north-america` we're going to tweak it to use these servers:

```
server 0.us.pool.ntp.org
server 1.us.pool.ntp.org
server 2.us.pool.ntp.org
server 3.us.pool.ntp.org
```

Restart the ntp service and ensure it's successfully running:

```
# dpkg-reconfigure ntp
# systemctl restart ntp
# systemctl status ntp
```

Check the servers that are being used. Ours looked like:

```
# ntpq -pn
     remote           refid      st t when poll reach   delay
offset  jitter
==============================================================
===============
 0.debian.pool.n .POOL.          16 p    -   64    0    0.000
+0.000   0.000
 1.debian.pool.n .POOL.          16 p    -   64    0    0.000
+0.000   0.000
 2.debian.pool.n .POOL.          16 p    -   64    0    0.000
+0.000   0.000
 3.debian.pool.n .POOL.          16 p    -   64    0    0.000
+0.000   0.000
 45.79.111.167   .INIT.          16 u    -   64    0    0.000
+0.000   0.000
 159.203.82.102  .INIT.          16 u    -   64    0    0.000
+0.000   0.000
 2607:7c80:54:3: 17.253.16.125    2 u   33   64   37   38.786
+0.281   0.363
 45.55.58.103    .INIT.          16 u    -   64    0    0.000
+0.000   0.000
-38.229.57.9     172.18.54.10     2 u   22   64   37  177.712
+16.303   0.549
-66.85.78.80     172.16.23.153    2 u   19   64   37   10.723
-3.097   1.648
-45.83.235.160   192.168.178.1    3 u   17   64   37  126.577
-2.389  16.239
-2602:fe2e:3:d:f 129.6.15.28      2 u   25   64   37   37.990
-1.897   0.760
-2603:c020:0:836 132.163.97.4     2 u   20   64   37   21.704
-0.011   1.762
+2620:2d:4000:1: 145.238.203.14   2 u   23   64   37  107.124
-1.574   1.843
-204.93.207.12   206.55.64.77     3 u   18   64   37   20.582
-11.085   1.879
+2001:4998:58:18 98.139.133.62    2 u   18   64   37   32.381
-0.252   1.439
*217.180.209.214 .GPS.            1 u   17   64   37   36.120
-1.425   1.476
```

Time should be relatively well sync'd now.

VII. Recursive Resolver Service ("DNS") Using unbound

> ***"The world is full of bad ideas that turn into wonderful things."***
>
> ***-- Neal Shusterman, author of the book <u>Unbound</u>***

25. unbound (Recursive DNS Service with DNSSEC Support)

"Everything begins with DNS." This importance of that simple zen-like "koan" explains why we're going to install the `unbound` recursive resolver service on the server, and do DNSSEC, rather than just using a 3rd-party free recursive resolver. Official installation instructions are available online at `https://nlnetlabs.nl/documentation/unbound/howto-setup/`

Begin by installing the unbound and the dnsutils packages:

```
    # apt install unbound
    # apt install dnsutils
```

Create the unbound group:

```
    # groupadd unbound
```

Create the unbound user (with a disabled shell):

```
    # useradd --shell=/usr/sbin/nologin --home-dir=/var/lib/unbound
--group=unbound unbound
```

You should see an entry in **`/etc/password`** that looks somewhat like this (your userid and group id numbers may vary):

```
    # cat /etc/passwd | grep unbound
    unbound:x:111:119::/var/lib/unbound:/usr/sbin/nologin
```

Ensure `unbound.conf` is configured as shown:

```
    # cat /etc/unbound/unbound.conf
    server:
       interface: 127.0.0.1
       interface: ::1
       access-control: 127.0.0.0/8 allow
       access-control: ::1/128 allow
       do-ip4: yes
       do-udp: yes
       do-tcp: yes
       do-ip6: yes
       prefetch: no
       verbosity: 1
       remote-control:
          control-enable: yes

    # The following line includes additional configuration files
    from the
```

```
# /etc/unbound/unbound.conf.d directory.
include-toplevel: "/etc/unbound/unbound.conf.d/*.conf"
```

Confirm the configuration file is syntactically clean with:

```
# unbound-checkconf
unbound-checkconf: no errors in /etc/unbound/unbound.conf
```

Start (or restart) the server and confirm it's running:

```
# systemctl start unbound
# systemctl status unbound
```

Now try a test query, explicitly forcing use of the locally-running server:

```
# dig www.domaintools.com @127.0.0.1
[…]
;; ANSWER SECTION:
www.domaintools.com.  300   IN    CNAME dhqm1qu93sq6.wpeproxy.com.
dhqm1qu93sq6.wpeproxy.com. 300  IN    A     141.193.213.21
dhqm1qu93sq6.wpeproxy.com. 300  IN    A     141.193.213.20
[…]
```

Looks good! If everything's running well, set your server to use the local `unbound` for all queries. You'll normally do this in `/etc/network/interfaces.d/50-cloud-init` (or a similarly named file in that directory).

Typically, you'll see a single `dns-nameservers` line listed in the `"inet"` (e.g., IPv4) stanza. Duplicate that line in your text editor, commenting out the original (in case you ever want to revert to it). Change the copy of that line to read:

```
dns-nameservers 127.0.0.1
```

Now look for an `"inet6"` stanza. It may also have a `dns-nameservers` line. If so, make sure it looks like:

```
dns-nameservers ::1
```

Save the file and exit the editor.

Now let's enable DNSSEC. Begin by running the `unbound-anchor` command to retrieve a copy of the DNSSEC root key.

```
# unbound-anchor
# chown unbound:unbound /var/lib/unbound/root.key
# ls -l /var/lib/unbound/root.key
```

```
     -rw-r--r--    1    unbound    unbound    758    Feb       9    03:06
/var/lib/unbound/root.key
```

Verify that the value in the local key file matches the hash at
`https://data.iana.org/root-anchors/root-anchors.xml`

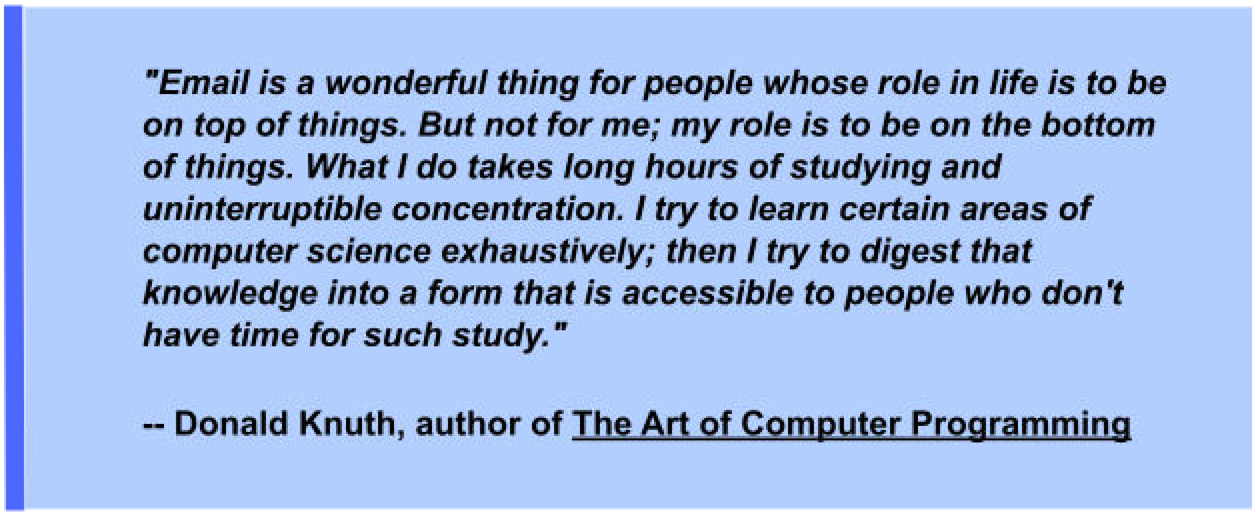
Then do:

```
     # systemctl restart unbound
```

Finally, test `unbound` to ensure it is doing DNSSEC. You should see something like the following:

```
     $ dig com @127.0.0.1 +dnssec

     ; <<>> DiG 9.16.37-Debian <<>> com @127.0.0.1 +dnssec
     ;; global options: +cmd
     ;; Got answer:
     ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59299
     ;; flags: qr rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY: 4,
     ADDITIONAL: 1

     ;; OPT PSEUDOSECTION:
     ; EDNS: version: 0, flags: do; udp: 1232
     ;; QUESTION SECTION:
     ;com.                IN    A

     ;; AUTHORITY SECTION:
     com.       900   IN    SOA   a.gtld-servers.net.
     nstld.verisign-grs.com. 1675914830 1800 900 604800 86400
     com.       900   IN    RRSIG SOA 8 1 900 20230216035350
     20230209024350 36739 com.
     fGrQCvR4XgJCjwsA/VknfsOsygrvaGacAvmdFCW9nsfUED1yYSYeQh1DYmNnsVe
     s4gyyPnJL2jRL1oGBUBQiQEg9uJHVpt6KBT/SjFDvpocG8zIIF4Q2mfF0m4Pcln
     NlXTDEW38CzLZm3izZR6PW+Kj6x80piytBgEggdI3L
     5P4yH6ZHOLYTUjrebTxRiQHkETPPoZqwA62zGdfteJqVFg==
     CK0POJMG874LJREF7EFN8430QVIT8BSM.com. 84011 IN NSEC3 1 1 0 -
     CK0Q2D6NI4I7EQH8NA30NS61O48UL8G5 NS SOA RRSIG DNSKEY NSEC3PARAM
     CK0POJMG874LJREF7EFN8430QVIT8BSM.com. 84011 IN RRSIG NSEC3 8 2
     86400 20230215052257 20230208041257 36739 com.
     tgh4dkHygg7rc4ENKPZxA9YBrsJedft9hKzYXY7hjjVm5MnMVGt9LU+885nkUqQ
     Be4mEle1XMNX86VrYjXxJblo5YKo/lQlXqt7T0wGiGFX1oV9XpUz+xwMCGGLheC
     o9G8loK7H1SYMS2GTr6ue5IypQ1hUd26/jP2Cx60Zh
     USZyWD+i0PirxvNiLQp1nVsn4Q0b60K4egu5nS3WtE9gCQ==

     ;; Query time: 43 msec
     ;; SERVER: 127.0.0.1#53(127.0.0.1)
     ;; WHEN: Thu Feb 09 03:54:12 UTC 2023
     ;; MSG SIZE  rcvd: 575
```

VIII. SMTP (Email) using Postfix (plus SPF/DKIM/DMARC)



"Email is a wonderful thing for people whose role in life is to be on top of things. But not for me; my role is to be on the bottom of things. What I do takes long hours of studying and uninterruptible concentration. I try to learn certain areas of computer science exhaustively; then I try to digest that knowledge into a form that is accessible to people who don't have time for such study."

-- Donald Knuth, author of The Art of Computer Programming

26. Installing the postfix MTA (SMTP) and alpine (a local command line email client)

Email is the classic Internet application. There are many different mail server options, but we're going to select `postfix` (instead of `exim or sendmail or …`) for our mail server.

Before starting, check `https://multirbl.valli.org/` to see if your server's IPv4 or IPv6 addresses, or your domain name, have a history of abuse issues. If either the IP or domain name are widely blocklisted, email from your system will likely run into deliverability problems. You may want to see if your provider can move your system to more pristine IPs (and while you're visiting with them, confirm that they're not routinely blocking all inbound or outbound traffic on port 25/TCP).

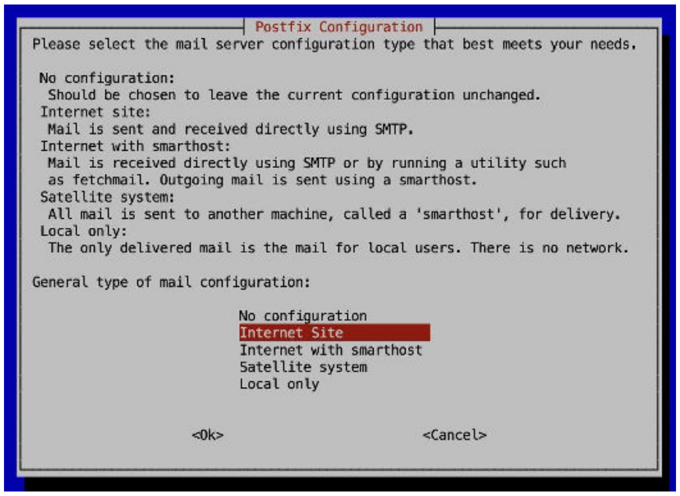
To install postfix:

```
# apt install postfix
# apt install postfix-pcre
```

`Postfix` has a helpful setup script that gets us most of the way to where we want to be:

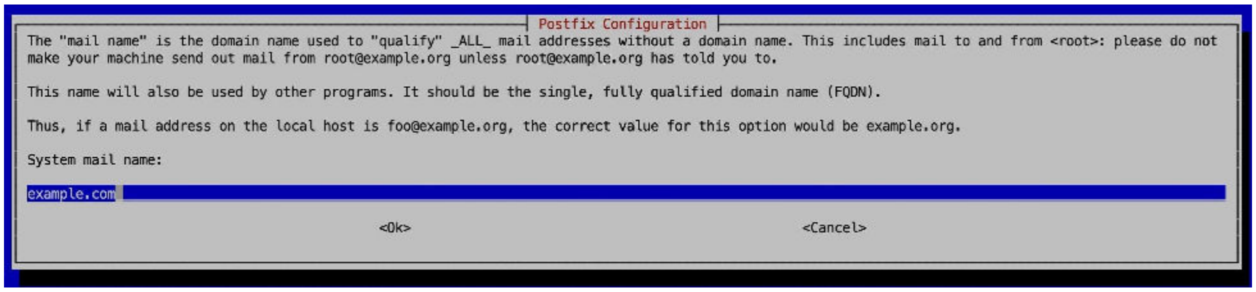
```
# dpkg-reconfigure postfix
```

Select "Internet Site:"



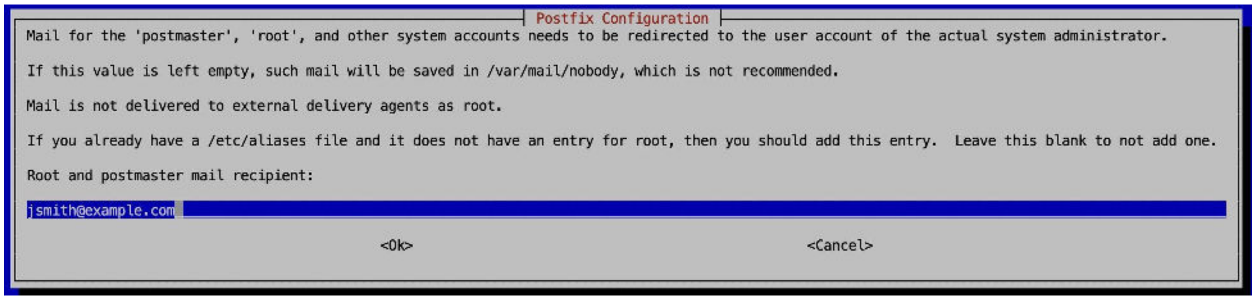
```
Postfix Configuration
Please select the mail server configuration type that best meets your needs.

 No configuration:
  Should be chosen to leave the current configuration unchanged.
 Internet site:
  Mail is sent and received directly using SMTP.
 Internet with smarthost:
  Mail is received directly using SMTP or by running a utility such
  as fetchmail. Outgoing mail is sent using a smarthost.
 Satellite system:
  All mail is sent to another machine, called a 'smarthost', for delivery.
 Local only:
  The only delivered mail is the mail for local users. There is no network.

General type of mail configuration:

                          No configuration
                          Internet Site
                          Internet with smarthost
                          Satellite system
                          Local only

                  <Ok>                        <Cancel>
```

Set the real name for your server:

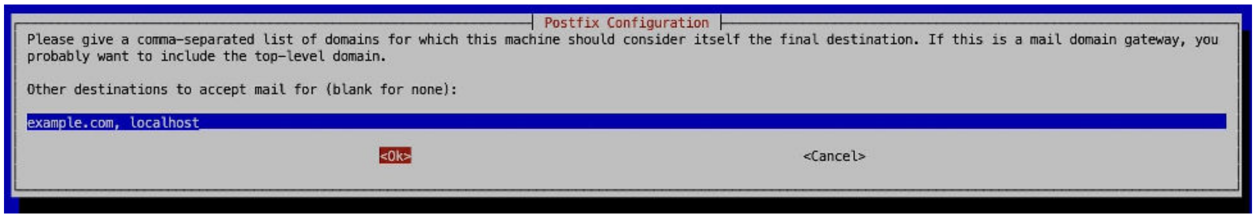


```
Postfix Configuration
The "mail name" is the domain name used to "qualify" _ALL_ mail addresses without a domain name. This includes mail to and from <root>: please do not
make your machine send out mail from root@example.org unless root@example.org has told you to.

This name will also be used by other programs. It should be the single, fully qualified domain name (FQDN).

Thus, if a mail address on the local host is foo@example.org, the correct value for this option would be example.org.

System mail name:

example.com

                <Ok>                                                    <Cancel>
```

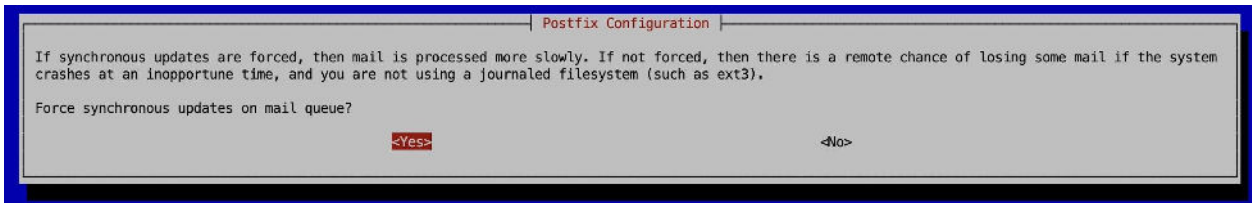
Set the address for the actual system admin's account:



```
Postfix Configuration
Mail for the 'postmaster', 'root', and other system accounts needs to be redirected to the user account of the actual system administrator.

If this value is left empty, such mail will be saved in /var/mail/nobody, which is not recommended.

Mail is not delivered to external delivery agents as root.

If you already have a /etc/aliases file and it does not have an entry for root, then you should add this entry.  Leave this blank to not add one.

Root and postmaster mail recipient:

jsmith@example.com

                <Ok>                                                    <Cancel>
```

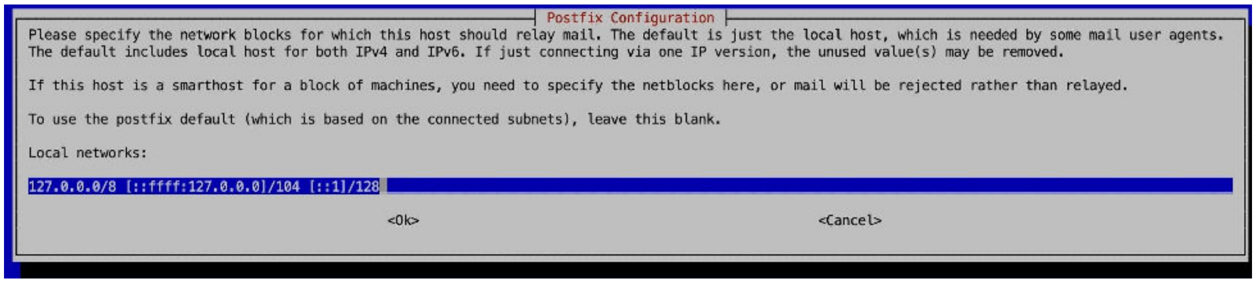


```
Postfix Configuration
Please give a comma-separated list of domains for which this machine should consider itself the final destination. If this is a mail domain gateway, you
probably want to include the top-level domain.

Other destinations to accept mail for (blank for none):

example.com, localhost

                <Ok>                                                    <Cancel>
```

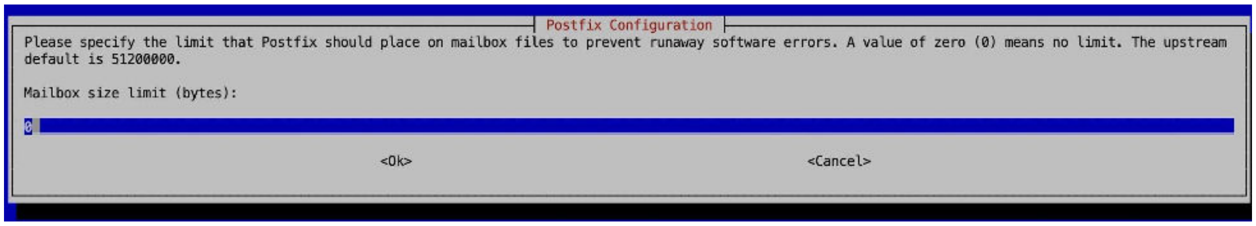


```
Postfix Configuration
If synchronous updates are forced, then mail is processed more slowly. If not forced, then there is a remote chance of losing some mail if the system
crashes at an inopportune time, and you are not using a journaled filesystem (such as ext3).

Force synchronous updates on mail queue?

                <Yes>                                                   <No>
```

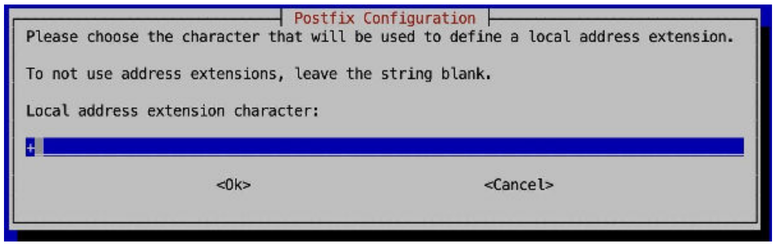
All outbound email will be produced locally:



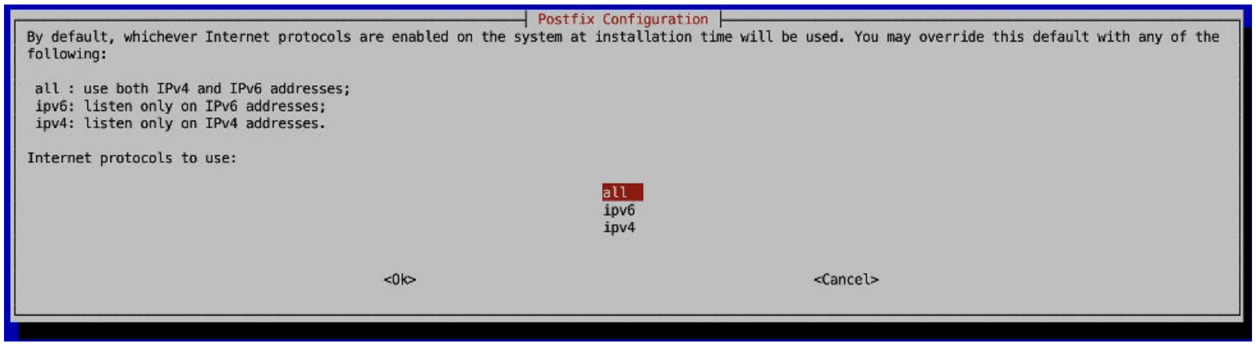
```
Postfix Configuration
Please specify the network blocks for which this host should relay mail. The default is just the local host, which is needed by some mail user agents.
The default includes local host for both IPv4 and IPv6. If just connecting via one IP version, the unused value(s) may be removed.

If this host is a smarthost for a block of machines, you need to specify the netblocks here, or mail will be rejected rather than relayed.

To use the postfix default (which is based on the connected subnets), leave this blank.

Local networks:

127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128

                    <Ok>                                      <Cancel>
```



```
Postfix Configuration
Please specify the limit that Postfix should place on mailbox files to prevent runaway software errors. A value of zero (0) means no limit. The upstream
default is 51200000.

Mailbox size limit (bytes):

0

                    <Ok>                                      <Cancel>
```



```
Postfix Configuration
Please choose the character that will be used to define a local address extension.

To not use address extensions, leave the string blank.

Local address extension character:

+

            <Ok>                        <Cancel>
```



```
Postfix Configuration
By default, whichever Internet protocols are enabled on the system at installation time will be used. You may override this default with any of the
following:

 all : use both IPv4 and IPv6 addresses;
 ipv6: listen only on IPv6 addresses;
 ipv4: listen only on IPv4 addresses.

Internet protocols to use:

                                   all
                                   ipv6
                                   ipv4

                    <Ok>                                      <Cancel>
```

To avoid a warning later from the lynis security auditing tool around `https://cisofy.com/lynis/controls/MAIL-8818/` , consider changing the banner Postfix displays. Go to `/etc/postfix/main.cf` and ensure the banner has been simplified to just:

```
smtpd_banner = $myhostname ESMTP
```

You may also want to do:

```
# postconf -e disable_vrfy_command=yes
```

Confirm postfix is running:

```
    # postfix check
    # systemctl enable postfix
    # systemctl status postfix
```

Let's now install a command line email client, `alpine (see https://alpineapp.email/):`

```
    # apt install alpine
```

Use alpine to try sending an email to another account you may have. Also send an email from that other account to your account on the new server.

27. What About Maildir?

We opted to used the default Postfix spool format, Mbox (see https://en.wikipedia.org/wiki/Mbox), but some may prefer to use Maildir instead (see https://en.wikipedia.org/wiki/Maildir).

If you're interested in trying Maildir, please see

https://help.ubuntu.com/community/PostfixBasicSetupHowto#Setting_Postfix_Support_for_Maildir-style_Mailboxes

for configuration recommendations.

28. Adding SPF

Assuming basic `postfix` is now working well, let's setup email authentication in postfix using SPF (see https://en.wikipedia.org/wiki/Sender_Policy_Framework). SPF helps to limit who can "send email as you," or to control "where email from your domain can originate." If you're a big New York bank, for example, perhaps it would be good if someone at a cybercafé in South America couldn't send phishing emails purporting to be you, eh?

We have two things to handle here:

- Creating and publishing our own SPF record (to limit who can send email as us "outbound"), and
- Routinely checking the SPF records that others have published, for the email we receive "inbound."

The right hand side (Rdata) of a typical TXT record for SPF (assuming you only send email for this domain from your server) might look like:

```
"v=spf1 a mx ip4:your_IPv4_address ip6:your_IPv6_address ~all"
```

If you use third party mail senders or have other more complex requirements, SPF record can become quite complex, particularly when including SPF records from other domains, but let's not overthink this. Publish a TXT record that looks like the above for your domain in your authoritative name servers, then check it with:

```
https://www.dmarcanalyzer.com/spf/checker/
```

Now we want to tell postfix to verify SPF for the incoming email it receives. There are several steps to doing that. Begin by installing the required postfix policy package:

```
# apt install postfix-policyd-spf-python
```

Now tweak two postfix configuration files, `/etc/postfix/master.cf` and `/etc/postfix/main.cf` by following the recipe at `https://help.ubuntu.com/community/Postfix/SPF`

At the bottom of `/etc/postfix/master.cf,` add the two lines:

```
policyd-spf  unix  -    n       n       -       -       spawn
  user=nobody argv=/usr/bin/policyd-spf
```

At the bottom of `/etc/postfix/main.cf`, add the following (we omit showing the remainder of that file)

```
policyd-spf_time_limit = 3600s
smtpd_recipient_restrictions =
   permit_sasl_authenticated
   permit_mynetworks
   reject_unauth_destination
   check_policy_service unix:private/policyd-spf
```

Check and reload postfix:

```
# postfix check
# postfix reload
```

Confirm that you're still able to send and receive email OK with `alpine.` Watch `/var/log/mail.log` for any issues.

29. Adding DKIM

DKIM (see `http://www.opendkim.org/)` uses cryptography to digitally confirm the origin of messages. Again, there's an outbound and inbound component to be configured. We're going to basically follow the approach outlined at `https://www.linuxbabe.com/mail-server/spf-dkim-postfix-debian-server` with some changes

We'll begin by installing the packages we need (if they aren't already installed):

```
# apt install opendkim opendkim-tools dns-root-data
```

Add postfix to the opendkim group:

```
# gpasswd -a postfix opendkim
```

Edit `/etc/opendkim.conf`, confirming the following lines are present and uncommented:

```
Canonicalization   relaxed/simple
Mode               sv
SubDomains         no
Nameservers        127.0.0.1
```

Setup the key file directory:

```
# mkdir -p /etc/opendkim/keys
# chown -R opendkim:opendkim /etc/opendkim
# chmod go-rw /etc/opendkim/keys
```

Edit `/etc/opendkim/signing.table` (replace ***example.com*** with the domain you're mailing from):

```
*@example.com        default._domainkey.example.com
```

Edit `/etc/key.table` (replace ***example.com*** wherever it occurs with the domain you're mailing from):

```
default._domainkey.example.com
example.com:default:/etc/opendkim/keys/example.com/default.private
```

Edit `/etc/opendkim/trusted.hosts` (replace ***example.com*** with whatever domain you're mailing from; note the leading dot on the domain name):

```
127.0.0.1
```

```
localhost
::1

.example.com
```

Let's now create our DKIM keys (replace ***example.com*** with your domain); note that the key from `default.txt` may be broken into several chunks, but it's basically one long alphanumeric public key, that's what the "*blahblahblah*" represents:

```
      # mkdir /etc/opendkim/keys/example.com
      # opendkim-genkey -b 2048 -d example.com -D
/etc/opendkim/keys/example.com -s default -v
      # chown opendkim:opendkim
/etc/opendkim/keys/example.com/default.private
      # chmod 600 /etc/opendkim/keys/example.com/default.private
      # cat /etc/opendkim/keys/example.com/default.txt
      Default._domainkey   IN   TXT   ( "v=DKIM1; h=sha256; k=rsa;
p=blahblahblah" )
```

Now add your DKIM public key to your domain's authoritative DNS as a new TXT record:

```
default._domainkey  "v=DKIM1; h=sha256; k=rsa; p=blahblahblah"
```

Finish lashing together the DKIM connection to Postfix:

```
mkdir /var/spool/postfix/opendkim
chown opendkim:postfix /var/spool/postfix/opendkim
```

Edit `/etc/opendkim.conf` and ensure the Socket line looks like:

```
SOCKET  local:/var/spool/postfix/opendkim/opendkim.sock
```

Edit `/etc/postfix/main.cf` adding at the end of that file:

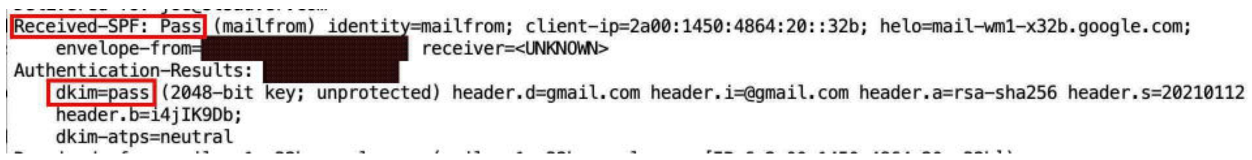
```
# Milter
milter_default_action = accept
milter_protocol = 6
smtpd_milters = local:opendkim/opendkim.sock
non_smtpd_milters = $smtpd_milters
```

Check for syntax errors, then reload Postfix:

```
# postfix check
# postfix reload
```

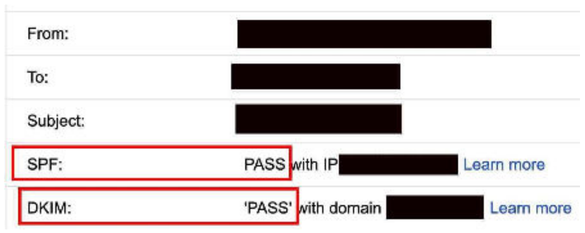
Check your DKIM setup with a public DKIM tester such as `https://www.dmarcanalyzer.com/dkim/dkim-checker/`
When using that tester, the "selector" should be specified as "default" (w/o the quotes).

Now try sending an email to your server from a 3rd party that publishes an SPF record and does DKIM signing, such as Gmail. Open the received message in `alpine`. While viewing it, hit h to toggle headers. You should see headers that include a "Pass" for both SPF and DKIM:



```
Received-SPF: Pass (mailfrom) identity=mailfrom; client-ip=2a00:1450:4864:20::32b; helo=mail-wm1-x32b.google.com;
    envelope-from=[redacted] receiver=<UNKNOWN>
Authentication-Results: [redacted]
    dkim=pass (2048-bit key; unprotected) header.d=gmail.com header.i=@gmail.com header.a=rsa-sha256 header.s=20210112
    header.b=i4jIK9Db;
    dkim-atps=neutral
```

Testing outbound -- now try sending an email TO your Gmail account. When you receive it on Gmail, look at the message, then go to the "three vertical dots" menu (found to the right of "Reply") and use "Show original." You should see "PASS" results as shown here:



From:	[redacted]
To:	[redacted]
Subject:	[redacted]
SPF:	PASS with IP [redacted] Learn more
DKIM:	'PASS' with domain [redacted] Learn more

30. Adding DMARC

While SPF and DKIM do a great job at what they do, they aren't perfect. For example:

- SPF normally focuses on the domain seen in the Return-Path: header -- that's a header that users rarely look at, unlike the user-visible From:
- DKIM verifies that a DKIM-signed message has a valid signature. But what about UNSIGNED messages? How should they be handled?
- What about situations where the use-visible From: is totally unrelated to the Return-Path: domain (the domains aren't "aligned")? Surely that's an anomaly worth calling out, isn't it?
- Wouldn't it be nice if you could get a report from sites that have seen attempts at spoofing your domain?

DMARC (see https://dmarc.org/) fixes those issues, so let's get DMARC installed and running, too.

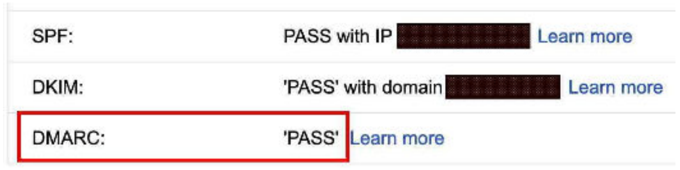
```
    # apt install opendmarc
```

Now create your DMARC record. An easy way to do that is with `https://dmarcian.com/dmarc-record-wizard/`

You're going to publish a new TXT record with this data in your domain's authoritative DNS. The records's name should be `_dmarc.`***example***`.com.` (Remember to substitute your domain name for ***example.com***). Be aware that many authoritative DNS control panels may automatically postpend your base domain -- if so, obviously use JUST `_dmarc` as the name).

After you get that TXT record published, validate it with `https://dmarcian.com/dmarc-inspector/`

You should also be able to see a "DMARC Pass" tag in Gmail if you send a new message to your gmail account, look at that message, and then check the Three Dot Menu Show Original:



SPF: PASS with IP [redacted] Learn more
DKIM: 'PASS' with domain [redacted] Learn more
DMARC: 'PASS' Learn more

31. Blocking Email from Entire TLDs

The Messaging, Malware and Mobile Anti-Abuse Working Group published a document entitled "M3AAWG Recommendations for Preserving Investments in New Generic Top-Level Domains (gTLDs)," see `https://www.m3aawg.org/gTLDInvestments`

It explained the risk that new gTLD owners face: if a domain becomes strongly associated with abuse, some may simply refuse traffic from the **entire** gTLD.

One list of abused TLDs is the list that Spamhaus publishes:
`https://www.spamhaus.org/statistics/tlds/`

Let's assume that we've decided we no longer want to accept any mail from the (totally fictitious) ".oregon" TLD. We could create a file called `/etc/postfix/reject_domains` that includes the line:

```
      /.oregon$/ REJECT We reject all .oregon domains
```

If there were other TLDs we also didn't want, we could include additional lines of that sort in the file, one for each unwanted TLD. After modifying that file, we'll create a dot db file of it by saying:

```
      # postmap reject_domains
```

Now we need to tell postfix to pay attention to our new rule (or rules). We'll do that by modifying `/etc/postfix/main.cf` to include the line:

```
      smtpd_sender_restrictions = check_sender_access
pcre:/etc/postfix/reject_domains
```

Note the "`pcre`" in that line. Using postfix's `pcre` capability requires that we have installed the `postfix-pcre` package, which we did at the same time we installed the main `postfix` package, so we should be fine (but if you see complaints about that `pcre` element, now you know what you must have somehow overlooked).

And then we can check syntax for sanity and reload:

```
      # postfix check
      # postfix reload
```

IX. HTTP and HTTPS (Web) using NGINX

> ***"The world wide web has really been quite spectacular and not something I would have predicted."***
>
> ***-- Jon Postel, Computer Scientist***
> *(https://en.wikipedia.org/wiki/Jon_Postel)*

32. Installing and Configuring NGINX to Serve Unencrypted Web Pages

The other "classic" Internet application (besides email) is http and https (the world wide web). We're going to use `nginx (https://www.nginx.com/)` to handle those protocols. We'll begin by installing the software we need:

```
# apt install nginx
```

We've previously allowed port 80 and port 443 through ufw, but if you didn't do that step back when, at least let 80/tcp through now:

```
# ufw allow 80/tcp
# ufw status
```

Ensure that `/etc/nginx/sites-available/default` looks like:

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    root /var/www/html;
    index index.html index.htm;
    server_name _;

    location / {
       # First attempt to serve request as file, then
       # as directory, then fall back to displaying a 404.
       try_files $uri $uri/ =404;

       autoindex on;
    }
}
```

Ensure that there's a config for your specific domain in `/etc/nginx/sites-available`. For the domain `example.com`, you'd want `/etc/nginx/sites-available/`***`example.com`*** to look like:

```
server {
    listen 80;
    listen [::]:80;
    root /var/www/html;
    index index.html index.htm;
    server_name example.com www.example.com;

    location / {
       # First attempt to serve request as file, then
```

```
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ =404;

        autoindex on;
    }
}
```

Now symbolic link the `/etc/nginx/sites-available/` files to `/etc/nginx/sites-enabled/`

```
ln                -s                /etc/nginx/sites-available/default
/etc/nginx/sites-enabled/default
ln              -s              /etc/nginx/sites-available/example.com
/etc/nginx/sites-enabled/example.com
```

Check your configuration is sane with:

```
      # nginx -t
      nginx:  the  configuration  file  /etc/nginx/nginx.conf  syntax  is
ok
      nginx:   configuration   file   /etc/nginx/nginx.conf   test   is
successful
```

Start `nginx` and confirm it is running:

```
      # systemctl start nginx
      # systemctl status nginx
```

Now try accessing a web page on your server using a web browser on your laptop using an http: (NOT an http**s**: URL). For example, if your web site is `example.com`, try going to `http://example.com/`

You'll probably see a default page. To install real content, go to `/var/www/html` and create (or upload) an `index.html` page.

While you're there, you might also want to create a `robots.txt` file explaining your site's policy when it comes to being crawled by well-mannered search engines and similar bots. For example, if you're okay with having all of your web pages crawled, `robots.txt` should look like:

```
      User-agent: *
      Allow: /
```

If, on the other hand, you want your site to NOT get crawled AT ALL by well-mannered search engines, change `Allow` to `Disallow`

Note that `robots.txt` policies are not ***technically*** "binding" -- ill-mannered search engines and bots will often completely ignore `robots.txt` files, or even use exclusions found in them as pointers to "particularly interesting" content to be retrieved FIRST.

Anyhow, let's now get and install a free Let's Encrypt TLS certificate so we can serve TLS-secured pages over https.

33. Obtaining and Configuring Let's Encrypt Certificates for TLS

We'll begin by installing `certbot`:

```
# apt install certbot python3-certbot-nginx
```

Ensure that **`443/tcp`** is allowed through the firewall:

```
# ufw allow 443/tcp
# ufw status
```

Now run the certbot (replace `example.com` with your actual domain):

```
# certbot --nginx -d example.com -d www.example.com
```

After running that, certbot tweaks `/etc/nginx/sites-enabled/example.com`

```
server {
    if ($host = www.example.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    if ($host = example.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [::]:80;

    root /var/www/html;
    index index.html index.htm;
    server_name example.com www.example.com;

    location / {
       # First attempt to serve request as file, then
       # as directory, then fall back to displaying a 404.
       try_files $uri $uri/ =404;

       autoindex on;
    }
}
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;

    root /var/www/html;
    index index.html index.htm;
    server_name www.example.com example.com;
    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
# managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
# managed by Certbot

    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
     index index.html;
       autoindex on;
     try_files $uri $uri/ =404;
     }
}
```

If you want to ensure that that configuration as shown is actually running:

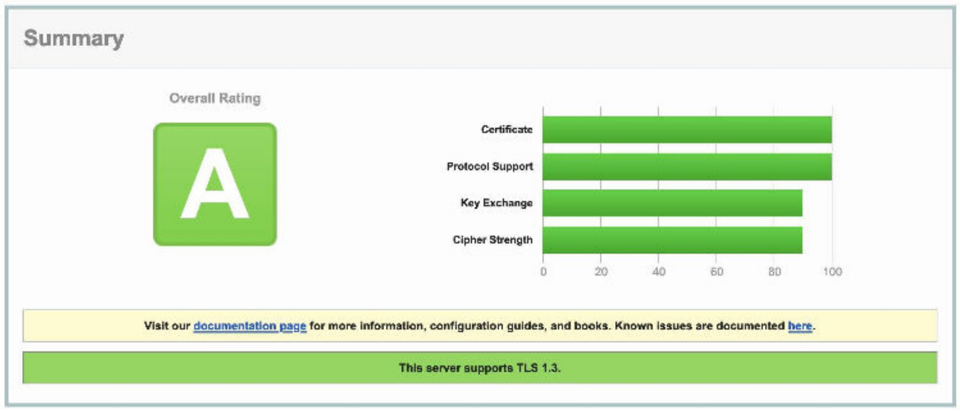
```
# nginx -t
# systemctl restart nginx
# systemctl status nginx
```

Now we just need to check and confirm that the certbot renewal service (to handle periodic cert renewal) is also running:

```
# systemctl status certbot.timer
```

If we want to watch the logs, check `/var/log/nginx/access.log` and `/var/log/nginx/error.log`

A typical report for the above configuration, btw, as summarized by `https://www.ssllabs.com/ssltest/`



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server supports TLS 1.3.

We're pretty happy with that sort of "report card."

That said, we'll freely concede that the above configuration is not as fully locked down as it could be. For a potentially better configuration, try the "Mozilla SSL Configuration Generator" at `https://ssl-config.mozilla.org/` That site will even help you to enable HTTP Strict Transport Security `(https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security)` and OCSP Stapling `(https://en.wikipedia.org/wiki/OCSP_stapling)`

X. Opportunistic TLS for Postfix MTA-to-MTA Traffic

> ***"Opportunistic Security" (OS) is defined as the use of cleartext as the baseline communication security policy, with encryption and authentication negotiated and applied to the communication when available. Cleartext, not comprehensive protection, is the default baseline. An OS protocol is not falling back from comprehensive protection when that protection is not supported by all peers; rather, OS protocols aim to use the maximum protection that is available."***
>
> ***-- RFC 7435, "Opportunistic Security: Some Protection Most of the Time"***

34. Configuring Opportunistic TLS for Postfix

Email historically was transferred over the Internet in plain text, but now it is much more common for SMTP traffic to be encrypted with TLS. Postfix is one of the SMTP products that supports this, see https://www.postfix.org/TLS_README.html

If TLS can't be successfully negotiated between the sending and receiving MTA, the message will fall back and be transmitted in "clear text" (e.g., unencrypted)

Just like TLS for a web site, TLS for email requires a digital certificate. We've already obtained certificates from Let's Encrypt for our web server, but we can use the same cert for SMTP, too. Let's dive right in…

a) Ensure the hostname has been set:

```
# postconf -h myhostname
example.com
```

If it isn't set, set it with:

```
# postconf -e 'myhostname = example.com'
```

Now continue with the required settings (change ***example.com*** to be your real domain). Settings are documented at `https://www.postfix.org/postconf.5.html`

```
# postconf -e 'smtp_tls_security_level = may'
# postconf -e 'smtp_tls_note_starttls_offer = yes'

# postconf -e 'smtpd_tls_cert_file =
/etc/letsencrypt/live/example.com/fullchain.pem'
# postconf -e 'smtpd_tls_key_file =
/etc/letsencrypt/live/example.com/privkey.pem'
# postconf -e 'smtpd_tls_security_level = may'
# postconf -e 'smtpd_tls_received_header = yes'
# postconf -e 'smtpd_tls_loglevel = 1'

# postfix check
# postfix reload
```

Check to see if `postfix` is now offering to do `STARTTLS` for ***inbound*** email coming to your server (in this case, yes, yes it is):

```
# telnet localhost 25
ehlo localhost
[…]
250-STARTTLS
[…]
QUIT
```

All's good in terms of opportunistic TLS for ***inbound*** email.

Now let's check the TLS status of ***outbound*** email. Use your server to send a test email to some `gmail.com` account you control. While viewing that test email, go to the "Three Vertical Dots Menu" (over next to the right of Reply) and do "Show Original." While viewing the raw message, look for the "handoff host" in the Received: header where your domain connects to a google.com host for the first time.. For example:

```
Received: from example.com ([IP_address]) by mx.google.com with ESMTPS
id [identifier here]
      for <user_here@gmail.com> (version=TLS1_3
      cipher=TLS_AES_256_GCM_SHA384 bits=256/256);
         [etc]
```

All's good ***outbound***, too!

XI. Malware

> *"The problem of viruses is temporary and will be solved in two years."*
>
> **-- John McAfee (1988)**

35. Scanning for Malware

While malware is less of a problem on Un*x systems than on MS Windows hosts, it can't be completely disregarded. One discussion of Debian anti-malware tools is `https://www.debian.org/doc/manuals/securing-debian-manual/ch08s08.en.html` ("Antivirus Tools.")

Normally we'd suggest running ClamAV, but if you're on a very small VPS (e.g., <= 512MB of RAM), you'll likely find it infeasible to run ClamAV (`https://www.clamav.net/`), even just in `clamscan` one-off mode. The problem is that running ClamAV, even with the default rules (to say nothing of additional 3rd-party rules), require more RAM than we have on a very small VPS, see for example `https://unix.stackexchange.com/questions/114709/how-to-reduce-clamav-memory-usage`

Fortunately, **`chkrootkit`** `(http://www.chkrootkit.org/), works fine:`

```
# apt install chkrootkit
```

When you run `chkrootkit`, you should see something like:

```
# chkrootkit
ROOTDIR is `/'
Checking `amd'...                                           not
found
Checking `basename'...                                      not
infected
Checking `biff'...                                          not
found
[lots more output elided here]
```

Hopefully, nothing interesting will be found!

Let's now move on to also try **`rkhunter`** `(https://rkhunter.sourceforge.net/).` We'll install it with:

```
# apt install rkhunter
```

Before running `rkhunter`, ensure `/etc/rkhunter.conf` is configured. I'd suggest:

```
MAIL-ON-WARNING=root
AUTO_X_DETECT=0
ALLOW_SSH_PROT_V1=0
ENABLE_TESTS=ALL
```

```
# DISABLE_TESTS=suspscan hidden_ports hidden_procs
deleted_files packet_cap_apps apps
SCRIPTWHITELIST=/usr/bin/egrep
SCRIPTWHITELIST=/usr/bin/fgrep
SCRIPTWHITELIST=/usr/bin/which
SCRIPTWHITELIST=/usr/bin/ldd
SCRIPTWHITELIST=/usr/sbin/adduser
ALLOWHIDDENDIR=/etc/.java
PASSWORD_FILE=/etc/shadow
```

After ensuring the config file is propertly configured, update the rkhunter properties with:

```
# rkhunter --propupd
[ Rootkit Hunter version 1.4.6 ]
File updated: searched for 179 files, found 143
```

And then run rkhunter:

```
# rkhunter --check  --skip-keypress
[LOTS of output elided here]

System checks summary
=====================

File properties checks...
    Files checked: 143
    Suspect files: 0

Rootkit checks...
    Rootkits checked : 502
    Possible rootkits: 0

Applications checks...
    Applications checked: 3
    Suspect applications: 0

The system checks took: 2 minutes and 43 seconds

All results have been written to the log file:
/var/log/rkhunter.log

No warnings were found while checking the system.
```

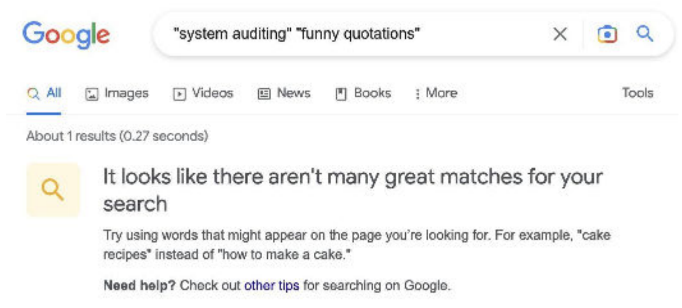
Note: Arrange RELIABLE BACKUPS of your VPS. If you become infested with malware, clean backups will be critical to your recovery.

Backup service will often be offered in a provider-dependent way, either bundled in the basic VPS monthly charge (or offered as an extra-cost option). Ensure you understand the terms of use for those backups. Can you easily restore individual files (perhaps from a snapshot), or are those backups merely meant for wholesale restoration of your entire site after a critical incident (such as a disk failure)?

You may also want to understand how (or IF), those backups can be used to export a copy of your server to an alternative hosting provider, should your current provider go out of business, or you simply decide you want to change providers. If there is any doubt about that capability, consider saving portable compressed tar achives to offline media you personally control.

In these days of significant ransomware threats, it is hard to overstate the value of multiple generations of solid backups (and/or snapshots). Backups can be critically important for even small VPS sites.

XII. System Auditing Tools



About 1 results (0.27 seconds)

It looks like there aren't many great matches for your search

Try using words that might appear on the page you're looking for. For example, "cake recipes" instead of "how to make a cake."

Need help? Check out other tips for searching on Google.

Exactly.

36. Lynis

Finally, let's also run `lynis (https://cisofy.com/lynis/)`, another open source security auditing tool, against our VPS.

```
# apt install lynis
```

Once installed, we can run a scan:

```
# lynis audit system --quick
[ Lynis 3.0.8 ]
  […]
  -[ Lynis 3.0.8 Results ]-

  Great, no warnings

  Suggestions (39):
  [suggestion details omitted here]
```

Not that the suggestions from `lynis` are often both detailed and (at times) technically esoteric. As an initial goal, we'd suggest prioritizing elimination of any warnings that `lynis` may identify. Some of the more esoteric suggestions may require substantial work to address, or may have side effects, so be sure to carefully research, plan, and test any changes before deploying them.

Some might want to look at `/etc/sysctl.conf` settings next. An approachable introduction is "Linux Kernel /etc/sysctl.conf Security Hardening," `https://www.cyberciti.biz/faq/linux-kernel-etcsysctl-conf-security-hardening/`

Those who've licensed the CIS benchmarks (or strictly non-commercial users) who want a fine-grained guide to detailed hardening of Debian Linux 11 should consider careful review of the "CIS Debian 11 Linux Benchmark" (09-22-2022, 874 pages), see **`https://www.cisecurity.org/cis-benchmarks/`**

XIII. Memory Considerations

37. Avoiding OOM (out of Memory) Events by Careful Selection of The Services to Be Run

This guide assumes we're using a very-small-memory VPS, with perhaps as little as 512MB of RAM:

```
# cat /proc/meminfo | egrep "(MemTo|MemFre|SwapTot|SwapFre)"
MemTotal:          484448 kB
MemFree:            92976 kB
SwapTotal:        2097148 kB
SwapFree:         2090688 kB
```

If you do experience an OOM condition due to the VPS being unable to allocate the amount of memory that's been requested, the operating system will typically terminate that process, logging an entry to `/var/log/error`. Note that the out-of-memory error is NOT going to necessarily report the process that's the "memory hog," it's just going to kill and log the name of the process that asked for memory it couldn't get. For example:

```
[…] kernel: Out of memory: Killed process 2158305 (clamd) […]
```

Having the OS terminate processes (more or less at random), is not a good thing when the selection of packages has already been carefully limited to essential network services (sshd, postfix, and nginx for static pages) and supporting security tools (such as ufw, unbound, NTP, etc.). For that reason, we're NOT going to use some often-terrific (but less-essential) security tools and services.

- Amavis (see https://www.amavis.org/)
- AppArmor* (see https://gitlab.com/apparmor/)
- auditd (see https://packages.debian.org/bullseye/auditd)
- ClamAV (see https://www.clamav.net/)
- DANE (see https://en.wikipedia.org/wiki/DNS-based_Authentication_of_Named_Entities)
- Fail2Ban (see https://www.fail2ban.org/wiki/index.php/Main_Page)
- Process Accounting (see https://packages.debian.org/bullseye/acct)
- SNMP (see https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol)
- SpamAssassin (see https://spamassassin.apache.org/)
- syslog-ng (Community Edition) (see https://www.syslog-ng.com/community/)
- sysstat/sar (see https://packages.debian.org/bullseye/sysstat)
- Tripwire (Open Source) (see https://github.com/Tripwire/tripwire-open-source)
- Web Application Firewalls (such as https://github.com/nbs-system/naxsi)

* AppArmor runs by default on Debian for a limited set of preconfigured services. We're going to leave those on/as-is.

38. Managing Swapping

`MemFree` will often look "low" on Linux systems because Linux will aggressively use most available memory for buffers/cache when it isn't needed for other purposes. Swap is normally reserved for the (hopefully rare) moments when you need more memory than you've got, but since it involves reading and writing to disk, swapping tends to be slow. You can influence the way the operating system uses memory with the "swappiness" value (see the discussion at "What does swappiness do and how does it affect swap_tendency?" at `https://access.redhat.com/solutions/103833)`. Let's check our sample system to see what its "swapiness" is set to:

```
# cat /proc/sys/vm/swappiness
60
```

For a server, some suggest we might see better performance by setting the swappiness to 10 (or some other value of your choice):

```
# sysctl vm.swappiness=10
vm.swappiness = 10
```

If we wanted to make that setting persistent, we'd add

```
sysctl vm.swappiness=10
```

to the bottom of `/etc/sysctl.conf`

XIV. Conclusion

> ***"We are not certain, we are never certain. If we were we could reach some conclusions, […]"***
>
> ***-- Albert Camus, Philosopher and Nobel Laureate in Literature***

While you can get a Un*x virtual private server cheaply and easily today, there are still many details associated with bringing up a functional and secure system.

This document assumes technical users are working from a Mac laptop, and are spending $10-$20/month for a remote Virtual Private System (VPS) running Debian 11 ("Bullseye"). At that price point, the VPS may have quite modest resources, including perhaps as little as 512MB of RAM.

Despite those constraints, we've demonstrated:

- The basic authoritative DNS records that should be created in one's DNS provider's control panel
- Bringing up sshd for encrypted remote access with public key authentication and Yubikey MFA support
- Getting automatic patching set up
- Setting up ufw (with ipsets) for firewall service
- Using NTP for time synchronization
- Configuring and running a DNSSEC-enabled recursive resolver service
- Installing Postfix for email, complete with SPF/DKIM/DMARC and opportunistic TLS
- Setting up an NGINX web server to deliver static web pages with Let's Encrypt free TLS certificates
- While malware's not the problem for Un*x systems that it is for Windows, we did also install two anti-rootkit products and a system auditing tool as part of the build
- Finally, we addressed the reality that having only 512MB forces us to forgoe many classic security tools and sevices we really wish we could have shoe-horned in, including staples such as ClamAV and SpamAssassin.

No security document can guarantee a "bulletproof" system, but we hope this document has provided a solid and usable foundation for those setting up a basic VPS like the one described above.

XV. Acknowledgements

> ***"With A Little Help from My Friends"***
>
> ***https://www.youtube.com/watch?v=LpwabPQW4p4***
> ***Joe Cocker (29 million views)***

We'd like to thank all those who reviewed and commented on earlier draft versions of this document, whether explicitly listed here or not, including:

- Mr Todd Herr, Technical Director, Standards and Ecosystem, Valimail
- Mr Marc Evans, retired, ex-Farsight Security/DomainTools

Any remaining issues are solely the responsibility of the author.

"When The Levee Breaks," Originally written and recorded by Kansas Joe McCoy and Memphis Minnie (1929)
https://www.youtube.com/watch?v=vFBpqR7eLc4

"If it keeps on rainin', levee's goin' to break
If it keeps on rainin', levee's goin' to break
When the levee breaks, I'll have no place to stay"

Re-arranged and re-recorded by Led Zeppelin as well as cover bands such as Zepparella (2010)
https://www.youtube.com/watch?v=xH-_9cwdLug