## Zero to Zero.Five

### From Zero to Zero.Five — Hardening My First Homelab with Proxmox, Tailscale, and SSH Security

Over the past few weeks, I took a dusty old gaming PC and transformed it into a secure, purpose-built homelab. Not just to spin up random VMs — but to create something **reliable, hardened, and production-like**, where I could simulate real-world identity, networking, and red-team scenarios in a safe environment.

This write-up documents that process — from installing Proxmox, to securing remote access with Tailscale, to enforcing zero-trust principles through firewalls, ACLs, and SSH key management.

The goal? Treat this like an actual enterprise asset, even if it's just running under my desk.

---

### 🔧 Installing Proxmox Virtual Environment

I started with Proxmox VE — a Type 1 hypervisor built on Debian, widely used for homelabs due to its GUI, container support, and KVM-based virtualization.

The install process was simple:

1. Created a bootable USB with Rufus
2. Installed Proxmox on bare metal
3. Configured a hostname, root password, and static IP


From here, I was dropped into a full-featured hypervisor. But before I started deploying anything, I knew I needed to address the elephant in the room:

---

### 🚨 Default Security Gaps in Proxmox

By default, Proxmox prioritizes usability — not security. That's fine for internal-only labs, but mine was going to support simulated attacks and remote admin access. Some major concerns:

- **Root login via SSH was allowed** — attackers could directly target the most privileged account
- **SSH password authentication enabled** — makes brute-force login attempts trivial
- **No multi-factor authentication (MFA)** for the web UI
- **Proxmox web GUI exposed on default port 8006**, often left open to the internet

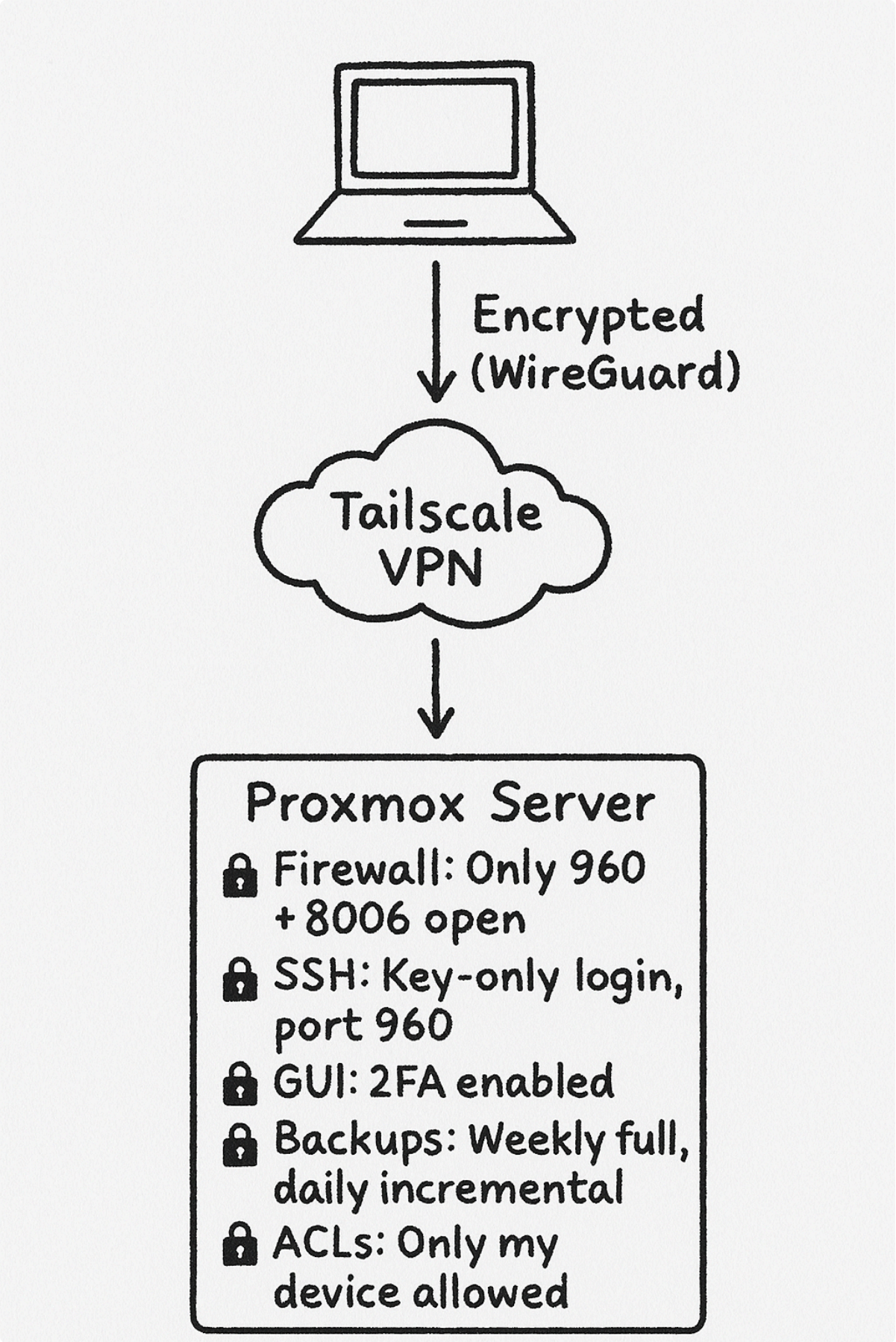
These are all commonly exploited misconfigurations in real-world breaches. So instead of rushing to create VMs, I decided to take a "secure-first" approach and lock the environment down immediately.

---

### 🌐 Private Remote Access with Tailscale

In enterprise networks, VPNs are a standard requirement for accessing anything sensitive. At previous jobs, I'd seen solutions like GlobalProtect or Cisco AnyConnect. I wanted a similar experience — but lightweight and identity-aware.

I initially tried WireGuard, but quickly pivoted to **Tailscale** — a drop-in VPN built on WireGuard that takes care of networking, firewalls, and DNS for you.




Why Tailscale?

- **Peer-to-peer encrypted mesh** — faster and more secure than hub-and-spoke models
- **Identity-aware access** — enforces access based on who you are, not just your IP

- **Zero port forwarding** — no need to expose my router to the world
- **Extremely easy setup** — literally `tailscale up`, log in with Google, and you're online

This took my Proxmox GUI from **"visible to the internet"** to **"only reachable through an encrypted tunnel authenticated to my identity."**

---

### 🔥 Basic Firewall with UFW

Even though Tailscale effectively "hides" the machine from the public internet, I wanted to go a step further. **Defense in depth** is the name of the game. If something fails (like Tailscale disconnecting), I don't want SSH or the GUI suddenly accessible.

So I installed **UFW** (Uncomplicated Firewall) and set strict rules:

bash

CopyEdit

`sudo ufw allow 960/tcp # SSH (custom port) sudo ufw allow 8006/tcp # Proxmox GUI sudo ufw default deny incoming sudo ufw default allow outgoing sudo ufw enable`

```
apt install ufw
ufw allow 960/tcp     # Allow SSH on custom port
ufw allow 8006/tcp    # Allow Proxmox Web GUI
ufw default deny incoming
ufw default allow outgoing
ufw enable
ufw status
```

Why This Matters:

- Reduces my server's network "attack surface"
- Keeps services reachable **only** from trusted sources (Tailscale peers)
- Protects against accidental exposure if I ever change my networking config

I also changed SSH from the default port 22 to 960 — not true security by itself, but it dramatically reduces automated scan attempts from bots that blindly target 22.

### 🚫 Killing Root Login and Using Least Privilege

Direct root access over SSH is dangerous. Even in small setups, it's best to avoid logging in as root entirely. Instead, I:

- Created a new user ( `denney` )
- Gave it sudo access for administrative tasks
- Disabled SSH login for `root` in `/etc/ssh/sshd_config`

Why?

- Limits the "blast radius" if the account is compromised
- Enforces better operational habits (privilege escalation only when needed)
- Aligns with real-world practices (nobody logs in as root in modern enterprise)

---

### 🔐 SSH Key-Based Authentication

Passwords are the weakest link in any authentication chain. Even strong ones can be phished or brute-forced. So I replaced SSH password login entirely with **public-key authentication**.

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/yourname/.ssh/id_ed25519):
```

Steps taken:

- Generated an ED25519 key pair
- Copied the public key to the server's `~/.ssh/authorized_keys`
- Disabled password login in `sshd_config`: nginx CopyEdit `PasswordAuthentication no` `ChallengeResponseAuthentication no PubkeyAuthentication yes`
- Restarted the SSH service

Now, login is only possible using my private key. No key? No access.
Even if someone *did* find my server, there's no username/password prompt to exploit.

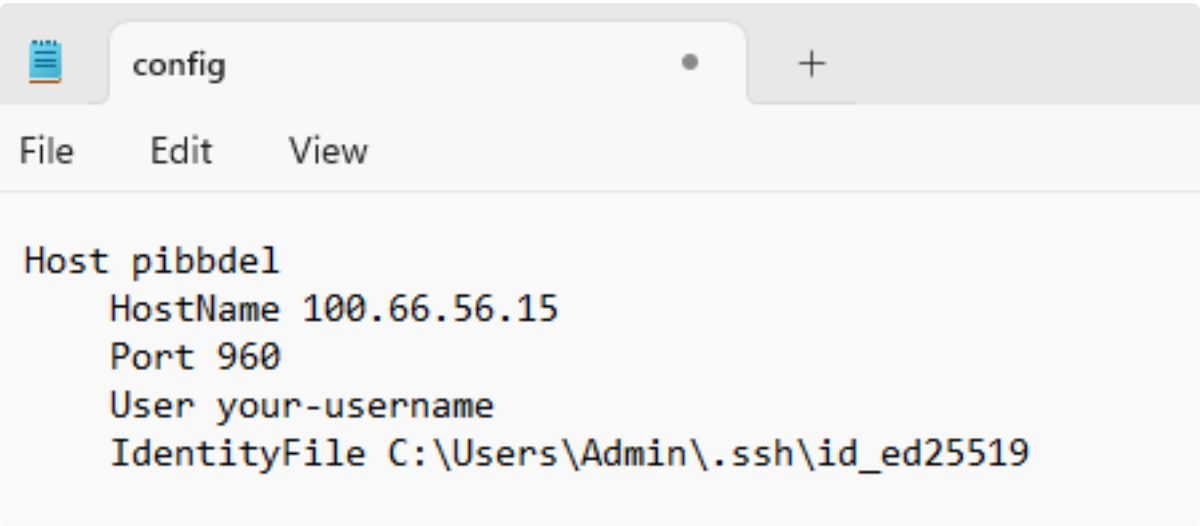
### 🧠 SSH Config Quality of Life

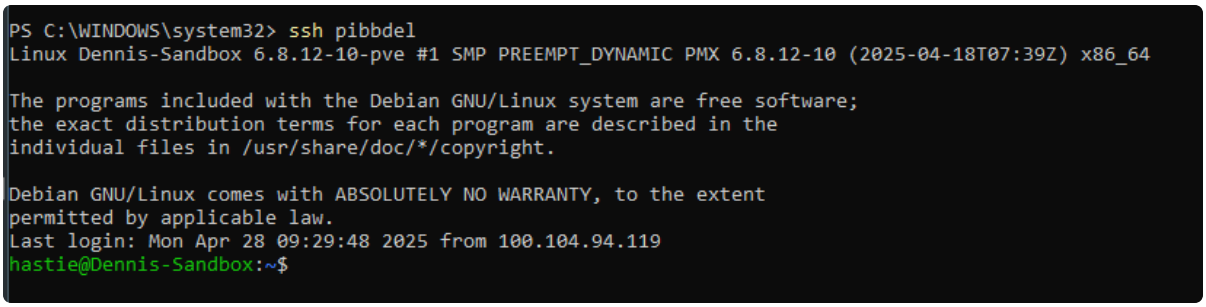
To avoid typing full SSH commands every time, I created a simple config entry on my local machine:

bash

CopyEdit

`Host pibbdel HostName 100.64.x.x Port 960 User denney IdentityFile ~/.ssh/homelab_ed25519`




Now I just type `ssh pibbdel` and I'm in.

---

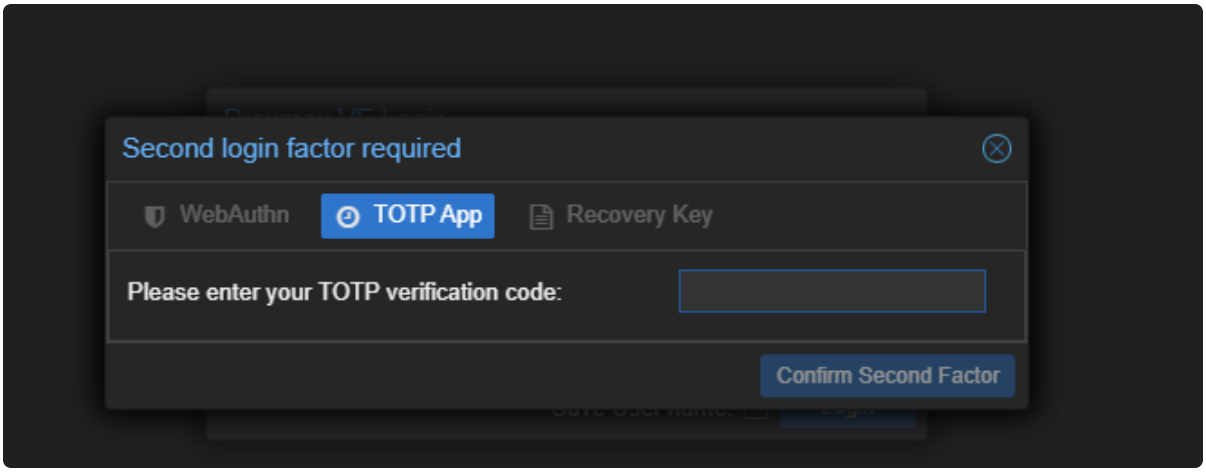
### 🔒 Two-Factor Authentication for the Web GUI

SSH wasn't the only thing I wanted secured. Proxmox's web interface is powerful — and dangerous if compromised.

So I enabled **TOTP (Time-Based One-Time Passwords)** for GUI access.

Steps:

1. `Datacenter > Realms > pam > Edit` → Enable TOTP
2. `Datacenter > Two Factor > Add > TOTP`
3. Scanned QR with an authenticator app (I use 2FAs Auth)



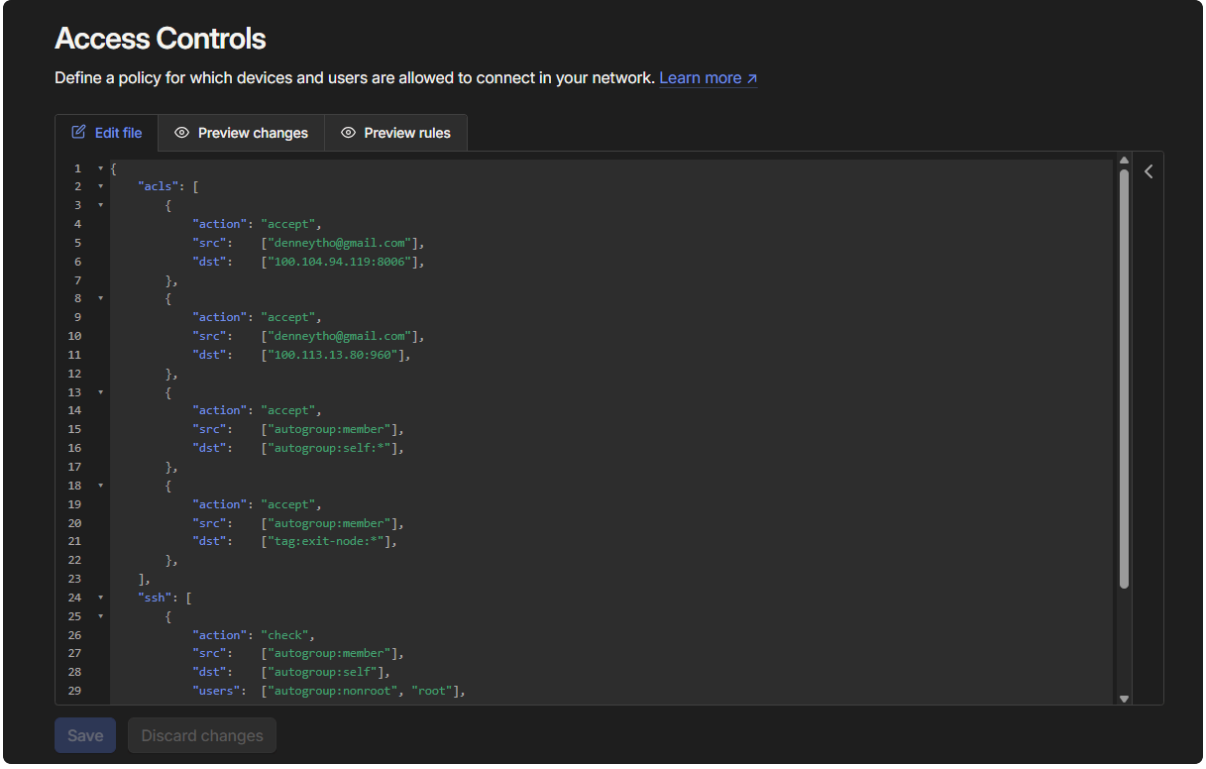

Now, logging in requires **both** a password and a one-time code from my phone — just like enterprise services.

---

### 🧱 Enforcing Zero-Trust Access with Tailscale ACLs

Tailscale has a powerful ACL system that lets you define **who can talk to what**. I edited my config so that:

- Only my device (`denneytho@gmail.com`) can reach:
  - Port `8006` (Proxmox Web GUI)
  - Port `960` (SSH)




Everyone else — including other Tailscale devices on the network — are denied. No lateral movement, no assumptions.

This is **zero-trust security**, even in a homelab.

---

### 🧬 Automated VM Backups

Even a well-secured system can fail — hardware crashes, config errors, or fat-fingered deletions.

So I set up scheduled VM backups via Proxmox:

- **Full backups weekly**
- **Incremental backups nightly**
- Stored to local disk (for now)
- Email notifications enabled for failures

Why?

- Quick rollback in case of breakage
- Safe experimentation without fear of loss
- Critical in labs that simulate malicious activity or system compromise

---

## ✅ Final Thoughts

This homelab isn't just a playground — it's a **controlled environment** where I'm simulating enterprise-grade security practices on a personal scale. Everything I did here — VPN tunneling, 2FA, key-based auth, firewalling, zero-trust ACLs — is directly applicable to real-world IT and cybersecurity roles.

It also laid the groundwork for my next project: simulating **Active Directory and Azure hybrid environments** with built-in misconfigurations, insider threats, and privilege escalation paths.

---

## ➡️ SOON Up Next: The HollowRoot Lab

Next post: I'll be diving into **Phase 1** of HollowRoot — a red/blue team project where a rogue intern compromises Active Directory and pivots into cloud infrastructure.