Lab Protocol System Hardening

Benjamin Lampart

Karim Salem

#### Abstract

This project explains how we securely deployed a Deno-based web application, a simple Todo app, on an Ubuntu 24.04 virtual machine. The app is made accessible over HTTPS using Nginx as a reverse proxy to manage traffic and enforce encrypted connections. We used Certbot with Let's Encrypt to automatically get and renew SSL/TLS certificates. To run the Deno app safely and automatically, we created a systemd service with strong restrictions. This helps limit what the app can access and keeps it isolated from the rest of the system.

We also focused on hardening the server to reduce the risk of attacks. This includes configuring OpenSSH for secure remote access, setting up a firewall with UFW, and using Fail2ban to block repeated login attempts. We enabled automatic security updates with Ubuntu Pro and added tools like AIDE for checking if system files were changed and auditd for detailed logging. The result is a secure and reliable setup for hosting small web applications, following best practices to improve system safety and stability.

Contents

[1. Procedure and Set-Up 2](#_Toc199792055)

[1.1 System Overview 2](#_Toc199792056)

[1.2 Key Software Components and Their Roles 2](#_Toc199792057)

[2. Analysis Part 4](#_Toc199792058)

[a. User creation and OpenSSH 4](#_Toc199792059)

[b. Deno-App and DB 5](#_Toc199792060)

[c. Fail2ban 8](#_Toc199792061)

[d. Ubuntu Pro and unattended-upgrades 9](#_Toc199792062)

[e. Nginx and certbot using Let’s Encrypt 11](#_Toc199792063)

[f. UFW (Uncomplicated Firewall) 14](#_Toc199792064)

[g. Aide and auditd 16](#_Toc199792065)

[h. General Hardening 22](#_Toc199792066)

[3. Conclusion 25](#_Toc199792067)

[4. References 25](#_Toc199792068)

[5. Images 26](#_Toc199792069)

# Procedure and Set-Up

For documentation purposes and to make collaboration easier this whole project has a Git Repository accompanying it which can be found here: <https://github.com/icefishii/syshardening>

The website can be found here:  
<https://syshardening.lampart.dev/>

Our objective is to securely deploy a Deno-based web application, making it accessible over the internet via HTTPS. This section provides an overview of the system's architecture, key software components, and the general security goals guiding its configuration. The application itself is a Todo-App where users can add and delete Todo’s, but it can be easily replaced with any other application due to it being set up behind a reverse proxy which handles most of the security and traffic.

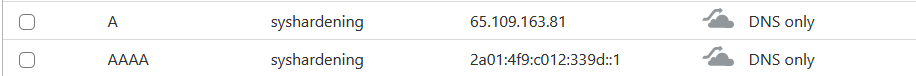
## 1.1 System Overview

The application is hosted on an Ubuntu 24.04 virtual machine. This VM acts as a dedicated server for our Deno application, which interacts with a local SQLite database. Internet exposure is managed through Nginx, configured as a reverse proxy, ensuring all external communication is encrypted via HTTPS.

The “hardware” of the virtual machine is not relevant for this project, but its running on a Proxmox Cluster. It has 3 Cores and 4GB of RAM and a virtual 75GB disk.

We didn’t expect either the Deno-Application nor the webserver to need more resources.

For the installation of Ubuntu, we cloned a template for an Ubuntu System, already present on the Proxmox Cluster which is based on a screenshot taken directly after a basic Ubuntu installation with all default values to serve as a base for future Machines.

The only change made was to assign the VM a public IP-Address and add the DNS-Records pointing to that IP-Adress so we can later obtain an SSL-Certificate for the server.

## 1.2 Key Software Components and Their Roles

For reaching our goal we first decided to evaluate which software we would need to focus on. Below is an overview of the software/tools running on the server and general areas we focused on in the hardening process.

* Deno: The runtime for our web application, chosen for its secure-by-default nature and because it allowed us to compile the application into a Linux binary which eliminated the need to install the Deno runtime on our system.
* SQLite: We chose an SQLite Database for this deployment as it matches the size of our application and is a lot easier to run in a secure way, as for example PostgreSQL as we only need to ensure correct file permissions on the database.
* Nginx: Nginx was chosen due to its popularity and the number of well-written guides on how to configure it in a secure way. Here we focused on important configurations like forcing https via HSTS (which was already enforced as we are using a .dev domain) and things like rate-limiting and ensuring secure standards were used for TLS.
* Certbot with Let's Encrypt: Essential for enabling HTTPS. This is the easiest way to obtain an SSL-Certificate for us as it offers a plugin for Nginx which automatically requests the Certificate from Let's Encrypt via an HTTP-Challenge.
* systemd: It’s the default for most Linux-Systems and gives us an easy way to execute the Deno-Application in a secure and restricted way. Additionally, it helps by making it easier to disable unused/unnecessary services on the machine to reduce the attack surface.
* UFW (Uncomplicated Firewall): Chosen because it’s the easiest way for us to implement a host-firewall by allowing only the Connections/Ports we require.
* Fail2ban: Protects against brute-force attacks, particularly for SSH. Its configuration for detecting and banning malicious IPs needs to be reviewed.
* SSH (OpenSSH): Probably the biggest target for any attacker and thus we tried to make sure its hardened as much as possible by enforcing things like ssh-keys for login and disallowing the root user to log in via ssh.
* Ubuntu Pro Security Features (esm-infra, livepatch, fips): Those were enabled as they provide access to more Repos containing security fixes and longer support directly by Canonical and livepatch which can apply those updates to the running system.

During the process of hardening the server we also installed multiple other programs to secure the server in specific ways or to ensure our configs are secure.

For example, we used Lynis to audit various configuration files and installed software against know-good values.

Our goals for the project are to ensure that we can serve our application in a way where there is no risk to the server and the other way around.

# Analysis Part

We organized the project into folders with configuration files and scripts to make setup reproducible.

## User creation and OpenSSH

To begin with, a non-root user with sudo access was created, and SSH keys were added for secure login. Next, we installed and ran Lynis to check the system’s security and applied its suggestions to the SSH configuration (/etc/ssh/sshd\_config), setting all suggested options to “no” except for the port. Additionally, we manually set “PermitRootLogin no”, even though it was not in the Lynis report.

Figure 1: The lynis report of SSH before any configuration.


Figure 1 SSH -Before



Figure 2 SSH -After

## Deno-App and DB

The Deno Web Application simply uses the inbuilt serve function to host a web server at localhost:8000 and has a basic API to GET the index.html which is inlined in the Typescript code as to bundle it in the binary executable. The connection to SQLite Database is by using the inbuilt deno bindings, it created the todos.db file and creates the table for the todos. The API endpoints then call helper functions which then execute SQL Prepared Statements to fight against SQL Injections.  
  
A screen shot of a computer code

AI-generated content may be incorrect.

Figure 3 Deno Source 1

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure 4 Deno Source 2

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 5 Deno DB

Then, after everything was set up, we compiled the Deno app using the command:  
`deno compile --allow-net --allow-read --allow-write --output todo-server main.ts`  
This gives the binary only the required permissions: network, file read and write access. We copied the binary to the server using scp into a separate webserver directory and created a dummy todos.db file there.

We added a new user called denoapp without shell access, no home directory, and no sudo rights. Ownership of the binary, database file, and directory was set to denoapp:denoapp. We also changed the permissions:

* binary to 500 (read and execute),
* database to 600 (read and write),
* directory to 700 (read, write, execute).

This webserver directory was placed in /opt.

To run the app automatically and securely, we created a systemd service. It runs the server as the denoapp user, restarts it on failure, and applies several sandboxing options like isolating file system access, memory usage, and disabling access to kernel logs and devices. It also sets limits on memory, CPU, and I/O usage to keep the service lightweight and contained.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 6 Deno Systemd Servie

## Fail2ban

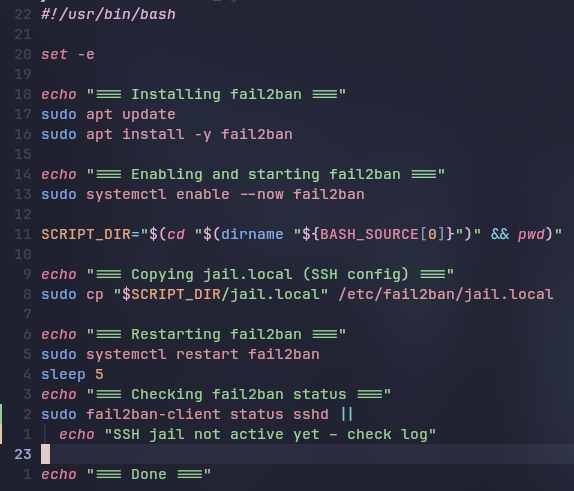
Afterwards, we configured Fail2Ban to further secure SSH by banning IP addresses for one hour if they fail authentication five times within a 10-minute window..

Figure 7 Fail2Ban script

This script installs Fail2Ban, enables its service, and copies the configuration file to the correct /etc/fail2ban/ directory. It then restarts the service and prints its status. The configuration is straightforward: it sets the port to ssh, the filter to sshd, specifies the log path, and defines the maximum number of retries, the time window, and the ban duration.

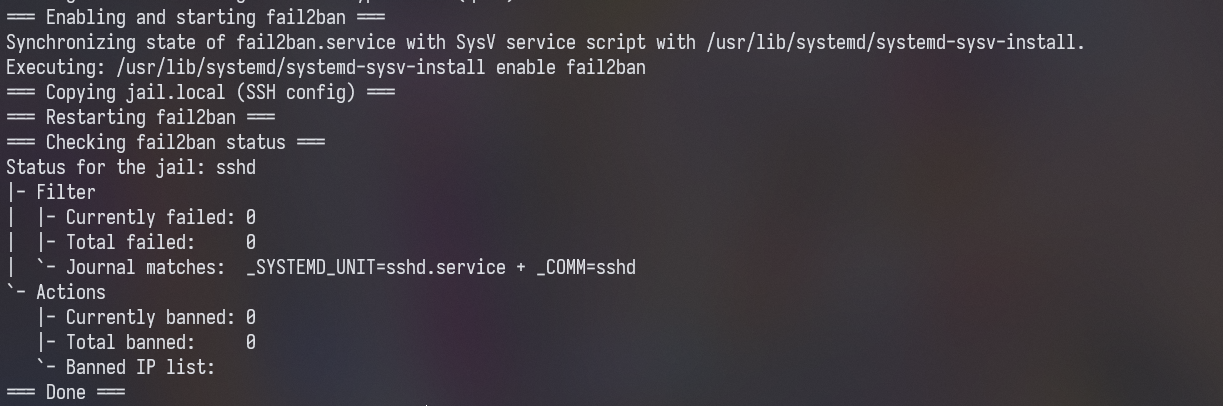


Figure 8 Fail2Ban Status

## Ubuntu Pro and unattended-upgrades

We chose to enable and configure the unattended-upgrades as a way to automatically receive and install software and security updates from the Ubuntu repositories. First, we created a script which takes the token needed to authenticate our machine with the Ubuntu Pro service to verify that we have a valid license as it is aimed at business customers. They do provide the first 5 machines free of charge tho, so we took advantage of this to have access to the additional ESM (Expanded Security Maintenance) Repositories which provide LTS support for security updates.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9 APT Sources Config

Above is a picture of one of those sources which got automatically added by the ubuntu pro client. We used a shell-script to enable and apply various features offered by it. The one we didn’t enable was the Ubuntu Security Guide as it was redundant due to our use of Lynis to validate configurations and the defaults seemed way too strict as it tried to remove Nginx multiple times even after directly specifying that we wanted to set the server up as a webserver.

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 10 Ubuntu Pro Features

This is a part from the above-mentioned script which just enables all features from the list above.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 11 Ubuntu Pro Client

The next step was to configure the unattended-upgrades service to run once a day and install both security- and normal feature-updates. We decided to also restart the server if an update required it as for our case the added security benefits outweigh the short downtime from a reboot.

There are two configuration files which decide how the service runs.

The first enables parts of the service and the second decides when and which upgrades are installed.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 12 20auto-updates

The options here tell the service to update the local package database first before checking for outdated software and set it to clean that cache every 7 days.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 13 50unattended-upgrades

The second one here specifies which repositories we want to use for the updates and the time for the reboot.

The script for enabling the service copies those two files from templates to the correct location and afterwards installs the service via apt which starts the service in the process.

## Nginx and certbot using Let’s Encrypt

We chose to use Nginx as a reverse proxy to securely expose our Deno Todo App to the internet. It is a popular and well-documented web server acting as our reverse proxy.  
With Nginx we can allow to route traffic to our local Deno server, apply security headers, enforce HTTPS and filter requests. Certbot automates the process of requesting and renewing TLS certificates from Let’s Encrypt, which helps us keeping our connection secure with minimal manual work. We made a shell script to automatically install Nginx and certbot, obtain a TLS certificate from Let’s encrypt and copy the Nginx config to their specified directories, lastly restarting Nginx itself.

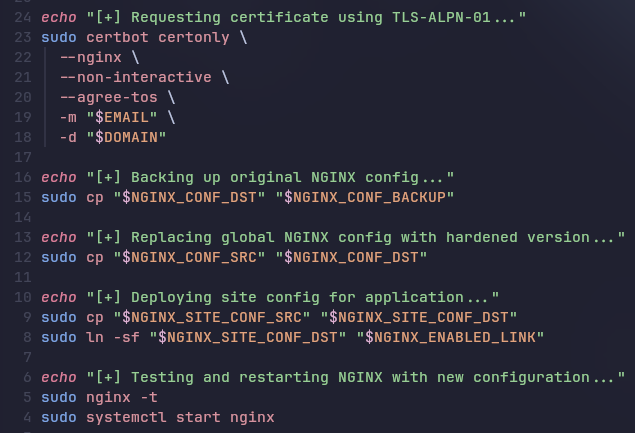


Figure 14 Nginx Script

We configured Nginx to redirect all HTTP traffic to HTTPS, and setup SSL with the needed certificate and resolver. It also includes modern TLS protocols (v1.2 and v.1.3), only strong cipher suites and security headers such as HSTS and X-Frame-options. We also added request rate limits, allow only GET, POST, HEAD and DELETE methods, and deny access to hidden files. Lastly, we have the actual reverse Proxy, by setting the URI of the Deno web application. Below are some screenshots of the configuration for the Deno web application.

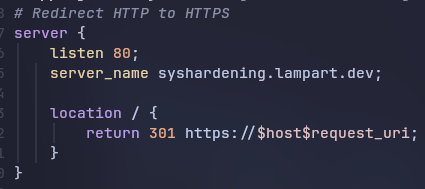


Figure 15 Nginx - HTTP Redirect

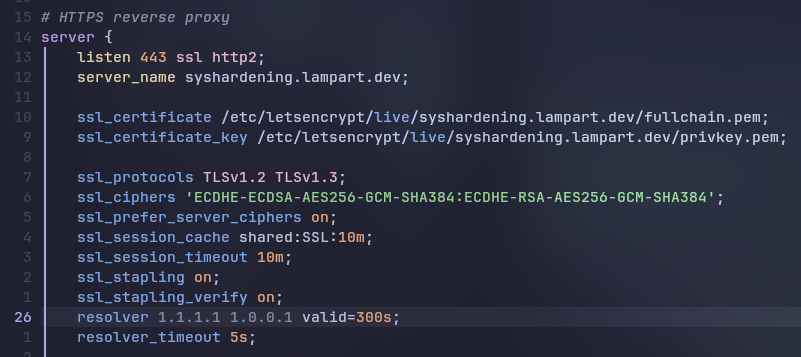


Figure 16 Nginx SSL



Figure 17 Nginx Rate Limit

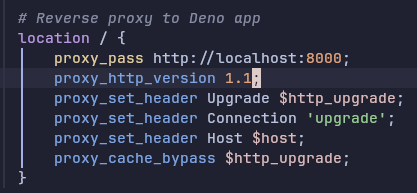


Figure 18 Nginx Proxy Deno

Additionally, we changed the default Nginx configuration to improve security and performance. We lowered timeout values and limited the size of requests to protect against slow or large requests. We also turned off the server version display to avoid leaking information and tweaked the TLS settings to use stronger encryption. To reduce data usage, we turned on Gzip compression, which we just uncommented from the default config. We added and error log and added basic rate limiting to prevent too many requests from one source. Finally, we adjusted buffer sizes to handle large or unusual requests more safely.

## UFW (Uncomplicated Firewall)

For a host-firewall we decided on UFW as it is very simple to set up. We first made a list of all incoming and outgoing connections we wanted to allow and then disabled everything else. In this case that is:

Incoming

* HTTP
* HTTPS
* SSH

Outgoing

* HTTP
* HTTPS
* DNS
* NTP

We also enabled logging for all traffic so we would have the ability to later monitor the traffic for anomalies.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 19 UFW - Status

Below is a randomly selected section of the ufw.log which shows attempted connections to our server which were blocked by the Firewall. Those are most likely from systems trying to scan servers for open ports or similar.

A computer screen with white text

AI-generated content may be incorrect.

Figure 20 UFW - Log

## Aide and auditd

Aide (Advanced Intrusion Detection Environment) is software by Red Hat which checks the integrity of files and directories against a internal database. This enables us to create that database after setting up the server to ensure that any changes to the configuration, done either by updates or a potentially malicious 3rd party are immediately visible. This is done via a cronjob which is run daily, that then prints out the results into a logfile.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 21 Aide - Changes

Here, for example, we installed some software and edited some files so the system would be able to find differences to allow us to test the system.

Above is the overview of all changes to files monitored by the system.

Below that in the log we are given a table-view of all edited, deleted or modified files and directories. Here for example the newly added ones.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 22 Aide - Added files

This is the shell-script we created to automate the deployment of aide to the machine for us.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 23 Aide – ShellScript

It starts out with installing aide via apt. We make a change to the default configuration with the software to use the more secure sha512 hashing algorithm but leave it default aside from that. Afterwards we initialize a new database on the system which takes a while as it will calculate the hash for each file on the system, aside from exceptions like temporary files. Then we take the resulting file and overwrite the existing (empty) database. The last step creates a cronjob to check the database against our server daily and saves the results.

In addition to aide we are also using auditd to monitor various other events happening on the machine. For example, executed commands, attempts at modifying files or privilege escalation. For this we wrote and collected some rules we deemed useful. Those are stored in a separate file next to a script to install auditd and apply them which makes it really easy to change or update them. The hardest part of setting this up was debating on which sources, be it files, folders or interrupts sent from the auditd kernel module to monitor. Below are some examples and why we chose them.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 24 auditd - rules user

The above rules monitor for changes to users and groups as that should practically never happen after the system is fully configured. Additionally in this snippet we are watching the sudo.log to see which commands are executed that way and the last part ensures that the configuration for our Nginx setup is not altered in any way.

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 25 auditd - rules network

This part checks for changes regarding network changes, either to the firewall or network setup via netplan.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 26 auditd - rules auth

Here we watch for the use of tools which modify users in any way like changing a password or adding a user, additionally it sees every failed login via the logs above.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 27 auditd - rules passwords

This part also watches for password/user changes but in a more direct way by monitoring the /etc/passwd and shadow file

A computer screen shot of white text

AI-generated content may be incorrect.

Figure 28 auditd - rules important files

This last part watches important directories and files like the auth.log but also the /opt folder in which our Deno-App and database are located.

The reports can be searched and viewed by included tools like ausearch and aureport  
A screenshot of a computer

AI-generated content may be incorrect.

Figure 29 auditd - report

Here is an example of using “aureport -i” to see the whole report of all logged activities since the start of the software.

Or here using “ausearch -i -k sudo\_log” to only show the usage of sudo (this is only one entry)  
A screen shot of a computer

AI-generated content may be incorrect.

Figure 30 auditd – search

## General Hardening

This chapter is for everything that didn’t fit into any of the other categories/software. Most of those are based on recommendations from Lynis or the Moodle course. First, we edited various kernel parameters based on the feedback that Lynis gave:

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 31 Lynis Kernel

Some of those are to protect symlinks and hard-links, “kernel.suid\_dumpable” prevents kernel dumps when an elevated program crashes. Most of the others are meant for preventing various network-based attacks like man-in-the-middle attacks. The one rule we did not enable was “kernel.modules\_disabled” which interfered with auditd and other modules which were not correctly loaded anymore.

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure 32 General Hardening Rule Override

The overrides are stored in a separate file which then gets copied into /etc/sysctl.d/ and applied via “sysctl --system”.

Next, we tightened the permission on various folders and files as the defaults set by Ubuntu were not ideal for everything. Below a snippet from the script which applies all the changes in this category

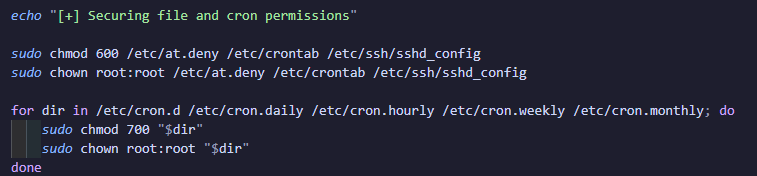


Figure 33 General Hardening Files

Another small thing was changing /etc/issue and /etc/issue.net to not show the OS version by overwriting them with a disclaimer.

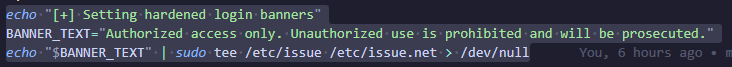


Figure 34 General Hardening Banner

After that we installed chkrootkit to monitor the server daily for signs of a rootkit similar software. As there is not really anything to configure, we just installed it via our script and set up a cronjob with logging for it. When we checked the logs the only thing it found was a false-positive for system-networkd.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 35 General Hardening chkrootkit

The next step was to disable certain unused network protocols as suggested by Lynis, which was done by disabling associated kernel-modules and rebuilding the initramfs

A computer screen with white text

AI-generated content may be incorrect.

Figure 36 General Hardening Initramfs

The last things were installing auditing tools like acct which monitors user activity on the system and changing the default umask from 022 to 027 to better isolate the files from one user to another. This obviously only applies to new files, but we still wanted to include it as an easy fix that could help prevent a user reading a file that they shouldn’t. The last thing we did was set a timeout for inactive users via the TMOUT variable in the profile file.

# Conclusion

Overall, we found a lot of different configuration options and tools that helped us harden our server setup. One major advantage was that we weren’t starting from scratch—there are many resources and examples online from others who have built similar systems. This made it a lot easier to figure out what we could change or install to improve security. Of course, there’s always more that could be done. For example, setting up a SIEM or XDR system like Wazuh would add another layer of protection, but it would also require a separate server and was beyond the scope of this exercise.

Our main goal was to securely deploy a Deno-based web application and make it accessible over the internet via HTTPS. Based on the steps we followed, we believe we’ve reached a solid level of security for both the Deno app and the Ubuntu 24.04 virtual machine it runs on.

In conclusion, we were able to create a secure environment for our Deno web app. By taking a layered security approach—covering everything from the host system and network to the application itself—we were able to reduce the risk of common attacks. Using a mix of automation tools and manual configurations helped us build a system that’s not only secure but also well-monitored and ready to be used in a real-world internet-facing setup.

All the configuration files and scripts and other files (including this Document) are stored in a public Git Repository on GitHub: <https://github.com/icefishii/syshardening>

# References

(n.d.). Retrieved from https://www.digitalocean.com/community/tutorials/how-to-harden-openssh-on-ubuntu-20-04

(n.d.). Retrieved from https://documentation.ubuntu.com/pro/pro-client/enable\_cis/

(n.d.). Retrieved from https://wiki.ubuntuusers.de/Aktualisierungen/Konfiguration/

(n.d.). Retrieved from https://www.thomas-krenn.com/de/wiki/SSH\_Login\_unter\_Debian\_mit\_fail2ban\_absichern

(n.d.). Retrieved from https://certbot.eff.org/instructions?ws=nginx&os=snap

(n.d.). Retrieved from https://beaglesecurity.com/blog/article/nginx-server-security.html

(n.d.). Retrieved from https://beaglesecurity.com/blog/article/hardening-server-security-by-implementing-security-headers.html

(n.d.). Retrieved from https://gist.github.com/renomureza/97da93192022fd0a962755e1f8206771

(n.d.). Retrieved from https://www.digitalocean.com/community/tutorials/ufw-essentials-common-firewall-rules-and-commands

(n.d.). Retrieved from https://linuxconfig.org/how-to-increase-the-security-of-systemd-services

(n.d.). Retrieved from https://askubuntu.com/questions/1174376/how-to-create-a-user-with-the-least-privileges-permissions-but-enough-to-do-ssh

(n.d.). Retrieved from https://www.digitalocean.com/community/tutorials/how-to-harden-openssh-on-ubuntu-20-04

(n.d.). Retrieved from https://www.thomas-krenn.com/de/wiki/SSH\_Login\_unter\_Debian\_mit\_fail2ban\_absichern

(n.d.). Retrieved from https://certbot.eff.org/instructions?ws=nginx&os=pip

(n.d.). Retrieved from https://wiki.ubuntuusers.de/Aktualisierungen/Konfiguration/

(n.d.). Retrieved from https://documentation.ubuntu.com/pro/pro-client/enable\_cis/

(n.d.). Retrieved from https://www.linkedin.com/pulse/hardening-ubuntu-server-2204-part-1-latte-sipping-basics-kollar

(n.d.). Retrieved from https://www.nuharborsecurity.com/blog/ubuntu-server-hardening-guide-2

(n.d.). Retrieved from https://www.reddit.com/r/Ubuntu/comments/1ap58d0/a\_guide\_on\_hardening\_ubuntu/

(n.d.). Retrieved from https://bugs.launchpad.net/usg/+bug/2064663

(n.d.). Retrieved from https://www.youtube.com/watch?v=c\_nUv-bn\_ak

(n.d.). Retrieved from https://www.youtube.com/watch?v=rxOTDG1peLw

(n.d.). Retrieved from https://www.youtube.com/watch?v=16k-bAtsUl8

# Images

[Figure SSH -Before 4](#_Toc199791543)

[Figure SSH -After 5](#_Toc199791544)

[Figure 3 Deno Source 1 5](#_Toc199791545)

[Figure 4 Deno Source 2 6](#_Toc199791546)

[Figure 5 Deno DB 6](#_Toc199791547)

[Figure 6 Deno Systemd Servie 7](#_Toc199791548)

[Figure Fail2Ban script 8](#_Toc199791549)

[Figure Fail2Ban Status 9](#_Toc199791550)

[Figure APT Sources Config 9](#_Toc199791551)

[Figure Ubuntu Pro Features 10](#_Toc199791552)

[Figure Ubuntu Pro Client 10](#_Toc199791553)

[Figure 20auto-updates 11](#_Toc199791554)

[Figure 50unattended-upgrades 11](#_Toc199791555)

[Figure Nginx Script 12](#_Toc199791556)

[Figure Nginx - HTTP Redirect 12](#_Toc199791557)

[Figure Nginx SSL 13](#_Toc199791558)

[Figure Nginx Rate Limit 13](#_Toc199791559)

[Figure Nginx Proxy Deno 13](#_Toc199791560)

[Figure UFW - Status 15](#_Toc199791561)

[Figure UFW - Log 15](#_Toc199791562)

[Figure Aide - Changes 16](#_Toc199791563)

[Figure Aide - Added files 16](#_Toc199791564)

[Figure Aide – ShellScript 17](#_Toc199791565)

[Figure auditd - rules user 18](#_Toc199791566)

[Figure auditd - rules network 18](#_Toc199791567)

[Figure auditd - rules auth 19](#_Toc199791568)

[Figure auditd - rules passwords 19](#_Toc199791569)

[Figure auditd - rules important files 19](#_Toc199791570)

[Figure auditd - report 20](#_Toc199791571)

[Figure auditd – search 21](#_Toc199791572)

[Figure Lynis Kernel 22](#_Toc199791573)

[Figure General Hardening Rule Override 23](#_Toc199791574)

[Figure General Hardening Files 23](#_Toc199791575)

[Figure General Hardening Banner 23](#_Toc199791576)

[Figure General Hardening chkrootkit 24](#_Toc199791577)

[Figure General Hardening Initramfs 24](#_Toc199791578)