htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Exercise 6: GNU/Linux - Securing active components

Laboratory protocol

Figure 1: Grouplogo

Subject: ITSI

Class: 3AHITN

Name: Stefan Fürst, Marcel Raichle

Gruppenname/Nummer: Team 7/7

Supervisor: SPAC, ZIVK

Exercise dates: 6.12.2024. 13.12.2024, 20.12.2024, 3.1.2025, 4.1.2025, 5.1.2025

Submission date: 4.1.2025

htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Contents

1 Task definition

Task 0 - Preparation
Ensure your server from Exercises 4 and 5 is configured with SSH. Verify that you can connect to the server via SSH using a client with a GUI.

Task 1 – Installing a Web Server
Install a web server (e.g., Apache or Nginx) and deploy a static HTML page displaying your group number, team members, and an AI-generated image. (Bonus: Deploy a dynamic PHP page.) Demonstrate access to the page from a client browser.

Task 2 – Securing with Basic Authentication
Set up Basic Authentication on the server. Create user accounts in the format `nnv-webuser` and for your instructors (e.g., `zivk-webuser`). Demonstrate authentication functionality. (Bonus: Capture the password using Wireshark.)

Task 3 – Encrypting with HTTPS
Enable HTTPS with a self-signed certificate, including your group number. Demonstrate encrypted access and explain potential issues. Install the certificate on a client to show why this action is not required in the public internet.

Bonus Task – Local DNS Setup (Optional)
Set up DNS on the server using `bind9` for local access via `xxx.itsi3.local`. Demonstrate DNS resolution and access the website by domain name.[1]

2 Summary

As preparation for the exercise, I optimized the Docker workflow by using Docker Compose for easier management, improved the readability of the Dockerfile, and, most importantly, created a `.env` file along with a build script that utilizes it, so I no longer hardcode my passwords in the Dockerfile. Additionally, I disabled password authentication and now copy the `authorized_keys` file into the container, allowing for key-based authentication from the start and enabling me to disable password authentication.
We need to install a web server, for which I chose `nginx`. I used it in conjunction with `php-fpm` to deploy a dynamic PHP webpage. The webpage includes our group number, names, and an AI-generated image. However, since this information should only be accessible with credentials, I implemented Basic Authentication to secure it. For this, the `apache2-utils` package was used to generate a `.htpasswd` file containing the credentials.
We demonstrated with Wireshark that the credentials were transmitted in plain text while using HTTP. To address this, a self-signed SSL certificate was generated using `openssl`, with the group number included in the `OU` field of the certificate. The server was then configured to use HTTPS. We showed that the credentials could no longer be read with Wireshark, as the traffic was now encrypted.
Lastly, we set up a domain, created a DNS record to point to the server, and generated a proper SSL certificate with Let's Encrypt, ensuring it is trusted and does not display a warning in the browser.

[1] This task definition was summarized by ChatGPT using the prompt: "Summarize this task definition in English and LaTeX and make it short and abstract."

3 Complete network topology of the exercise

Figure 2: Network topology of this exercise

4 Exercise Execution

4.1 Preparation

The requirements for this exercise are a headless Linux server with hardened SSH, which only allows connections via key pairs. However, I removed the OTP authentication added in the last exercise, as it was overkill for this use case and became a burden to use.

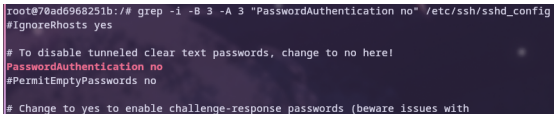


Figure 3: Password authentication disabled

4.2 Testing the SSH connectivity.

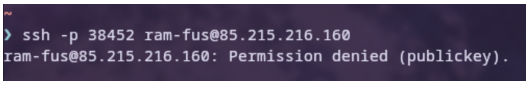


Figure 4: No SSH key available

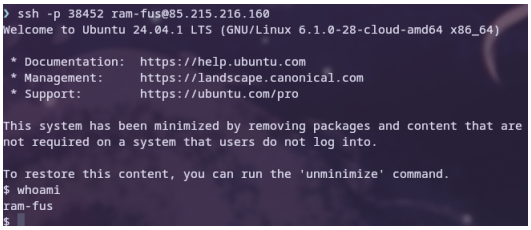


Figure 5: ram-fus authenticating via SSH key

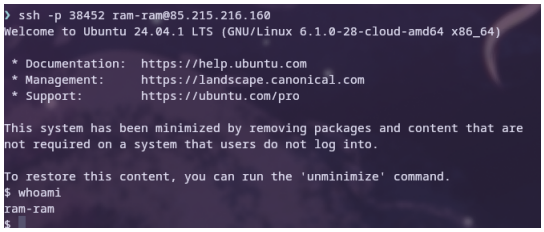


Figure 6: ram-ram authenticating via SSH key

4.2.1 Changes to the Docker setup

To improve the quality of life when working on this project, I switched from aliasing a long and hard-to-read run command to using Docker Compose, which allows you to define and run multi-container applications. Since it's in a YAML file, it is more readable and easier to work with, even in this use case where I only have one container.[1]

```
services:
    #setting the name image and restart policy
    webserver:
        container_name: itsi
        image: itsi:latest
        restart: no
    #setting exposed ports
    ports:
        - "38452:38452"
        - "80:80"
        - "443:443"
```

Furthermore, instead of having all of the credentials in the Dockerfile, I created a `.env` file in which the passwords are set. To utilize that, I made a build script that passes the variables from the file to the Dockerfile [2].

```
#!/bin/bash
export $(cat .env | xargs)

docker buildx build -t itsi:latest\
        --build-arg ROOT_PW=$ROOT_PW \
        --build-arg RAM_WEBUSER_PW=$RAM_WEBUSER_PW \
        --build-arg ZIVK_WEBUSER_PW=$ZIVK_WEBUSER_PW \
        --build-arg RAM_FUS_PW=$RAM_FUS_PW \
        --build-arg RAM_RAM_PW=$RAM_RAM_PW \
        --build-arg RAM_ALOIS_PW=$RAM_ALOIS_PW \
        --build-arg RAM_CHRIS_PW=$RAM_CHRIS_PW \
        --build-arg RAM_BERTA_PW=$RAM_BERTA_PW .
```

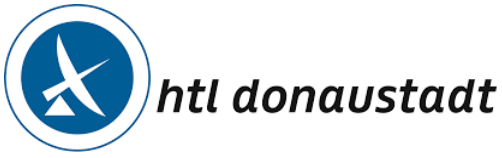
These build-time arguments are referenced in the Dockerfile like this:

```
ARG ROOT_PW
ARG RAM_WEBUSER_PW
ARG ZIVK_WEBUSER_PW
ARG RAM_FUS_PW
ARG RAM_RAM_PW
ARG RAM_ALOIS_PW
ARG RAM_CHRIS_PW
ARG RAM_BERTA_PW
...
RUN echo 'root:$ROOT_PW' | chpasswd
...
```

Here is what the `.env` file looks like for this project:

```
...
ROOT_PW='some password'
...
```

Note that the quotes are only necessary if the password contains characters like `&`, which the shell will interpret. With this change, I can add the `.env` file to my `.gitignore` file so I don't accidentally commit my passwords again and handle passwords in a Dockerfile properly.



To still utilize my alias script, I changed every instance of `docker run` to `docker compose up -d`, `docker stop itsi && docker rm itsi` to `docker compose down`, and added the use of the build script to it.

```
#!/bin/bash
alias relaunch="sh -c 'docker stop itsi && docker rm itsi &&\
                       ./build.sh &&\
                       docker compose up -d && docker exec -it itsi /bin/bash'"
alias rebuild="sh -c './build.sh &&\
                      docker compose up -d && docker exec -it itsi /bin/bash'"
alias stop="sh -c 'docker compose down'"
```

Furthermore, instead of having to upload my container every time I rebuild, I added these three lines to copy the `authorized_keys` file with the devices I use to the container, so that every time I relaunch, I can just immediately SSH into it.

```
COPY ./mapped-files/authorized_keys /root/.ssh/authorized_keys
COPY ./mapped-files/authorized_keys /home/ram-fus/.ssh/authorized_keys
COPY ./mapped-files/authorized_keys /home/ram-ram/.ssh/authorized_keys
```

Lastly, the line in the Dockerfile that specifies the exposed ports is edited to expose ports 80 and 443, as they will be required for this exercise.

```
EXPOSE 38452 80 443
```

4.3 Installing an active component

Now, it's required to install a web server. I chose Nginx because I am most familiar with it, and due to its high performance and simplicity of use.
It's installed and run by adding these lines to the Dockerfile and including `service nginx start` on the last line.[2]

```
...
RUN apt install -y nginx
...
CMD service ssh start && service nginx start && tail -F /dev/null
```

After modifying the Dockerfile, rebuilding, and redeploying, if we now open the web browser and go to the server's IP, we see the following.

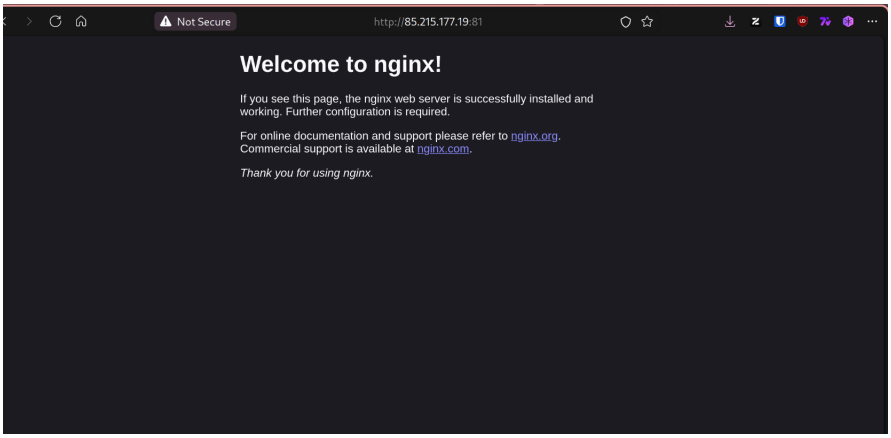


Figure 7: Default nginx site[3]

The HTML site displayed is located at `/var/www/html/index.nginx-debian.html`. Additionally, I replaced the `/var/www/html` directory with `/var/www/metyr.xyz`, in which I have the following file structure:

```
`-- html
   |-- private
   |   `-- private.php
    `-- public
        `-- index.php
```

These two directories are mapped onto the Docker container in the `docker-compose.yml` file, as shown below. Since they are mapped, every time the files are changed on the host, the changes carry over to the container, allowing for an easy and fast development workflow without the need to exec into the container or copy the files when creating the image. Nginx won't serve those PHP files without further configuration, which is explained in the next section.

```
volumes:
    - ./mapped-files/public:/var/www/html/public:rw
    - ./mapped-files/private:/var/www/html/private:rw
```

Additionally, I edited the Dockerfile to delete the default Nginx configuration file, located at `/etc/nginx/sites-enabled/default`, a symlink to the file `/etc/nginx/sites-available/default.conf`, and replaced it with one matching my domain name for better readability.

[2]While `init.d` has been deprecated in distributions such as RHEL and Oracle Linux, on Ubuntu it's not yet deprecated but considered legacy. Since this is a Docker container that's supposed to be isolated, adding `systemd` would break that isolation since it changes many low-level parameters. Almost all of its features besides process management are useless to me, so I opted for `init.d` since it doesn't require any further setup, doesn't break isolation, or add unnecessary weight.

[3]The IP and port are different since, initially, I used a VPS that already had a web server running, so I had to change the port. After that, I switched to a new VPS, which will be used for the rest of the exercise.

```
RUN rm -rf /var/www/html/
RUN mkdir -p /var/www/metyr.xyz/html
RUN rm /etc/nginx/sites-available/default
RUN rm /etc/nginx/sites-enabled/default
#copying the configuration file to the container during the build process
COPY ./mapped-files/metyr.xyz /etc/nginx/sites-available/metyr.xyz
#symlinking the new configuration file to enable the site
RUN ln -s /etc/nginx/sites-available/metyr.xyz /etc/nginx/sites-enabled/metyr.xyz
```

4.3.1 Setting up PHP-FPM with Nginx

To give Nginx the ability to serve PHP files, the `php-fpm` (FastCGI Process Manager) package is required. With this package installed, the following lines can be added to the server block in the Nginx configuration file.

```
server{
        ...
        #setting the location of the index file to serve
        index public/index.php;
        #location block, which matches requests for files ending with .php
        location ~ \.php$ {
                #include the fastcgi-php configuration file
                include snippets/fastcgi-php.conf;
                #passing the requets to the FastCGI server on set socket
                fastcgi_pass unix:/run/php/php8.3-fpm.sock;
        }
        ...
}
```

Additionally, the `php-fpm` service has to be started, so the default command of the container is edited.

```
CMD service ssh start && service nginx start && service php8.3-fpm start\
    && tail -F /dev/null
```

If we now rebuild the container, deploy it, and go to the IP address of the server in the browser, we can see the PHP page displayed. The site has a public part, located at `public/index.php`, and a private part, located at `private/private.php`. The private part contains the number and members of my group, along with an AI-generated image, which is why it requires authentication to access. This will be explained in the next section.

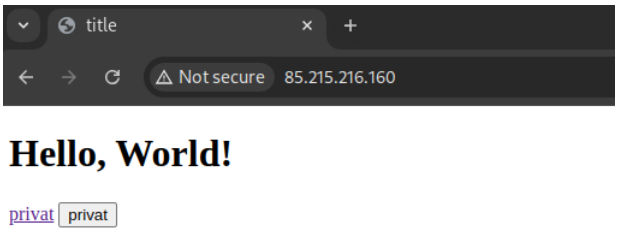


Figure 8: Viewing the index of the website

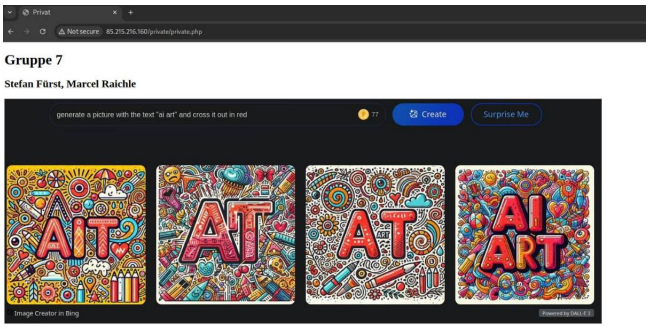


Figure 9: Viewing the private part of the website

4.4 Securing Nginx with Basic Authentication

To restrict access to the website or certain parts of it by implementing username/password authentication, a file containing usernames and passwords is required. This file can be generated using tools such as `apache2-utils`, which I will use for this exercise. Additionally, HTTP Basic Authentication can be combined with other access restriction methods, such as IP address or geographical location. [3]

4.4.1 Creating a Password File

With `apache2-utils` installed, we can now generate a password file by using the `htpasswd` command with the `-c` flag to create a new file. The file path is specified as the first argument, and the username is specified as the second argument. However, to avoid having to manually type in the password, the `-i` flag is used to take the password from `stdin`, which we pass using `echo`, while using the `-n` flag to remove the trailing newline. The passwords are sourced from the `.env` file mentioned in section 4.2.1. [3, 4, 5]

```
RUN echo -n "$RAM_WEBUSER_PW" | htpasswd -i -c /etc/apache2/.htpasswd ram-webuser
RUN echo -n "$ZIVK_WEBUSER_PW" | htpasswd -i /etc/apache2/.htpasswd zivk-webuser
```

4.4.2 Configuring the authentication in Nginx and testing it

To require authentication for a specific area on the website, we need to create a location block that matches everything in the `/private` directory. To do this, Nginx URL matching is used.[6]
It offers the following operators:

```
= #An exact match of the URI path\abc
~ #A case-sensitive regular expression match.
~* #A case-insensitive regular expression match.
^~ #Indicates that the following path should be considered the best
   #non-regular expression match.

#matching everything that contains /private in the path.
location ^~ /private {
        #making php work in the location
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.3-fpm.sock;
        auth_basic "Private Area";
        #selecting the file from which to use the credentials from
        auth_basic_user_file /etc/apache2/.htpasswd;
}
```

To visualize testing the login, I added this to the private PHP page to show the currently logged-in user: `<h3>Hello <?php echo $_SERVER['PHP_AUTH_USER']; ?></h3>`.[7]

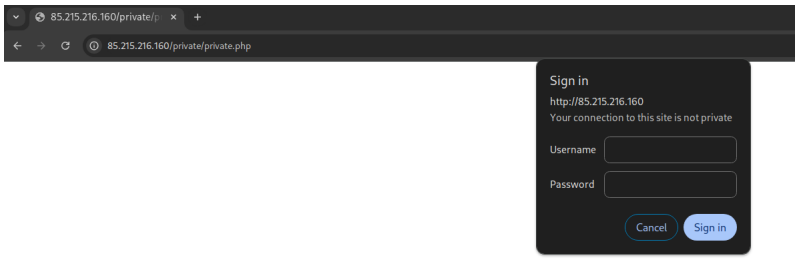


Figure 10: Showing the sign-in prompt

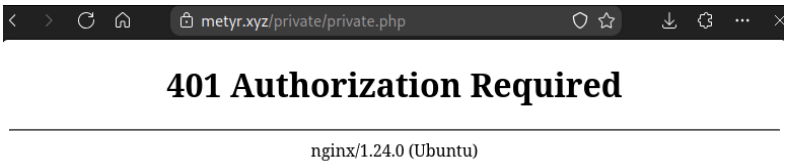


Figure 11: Failed authentication

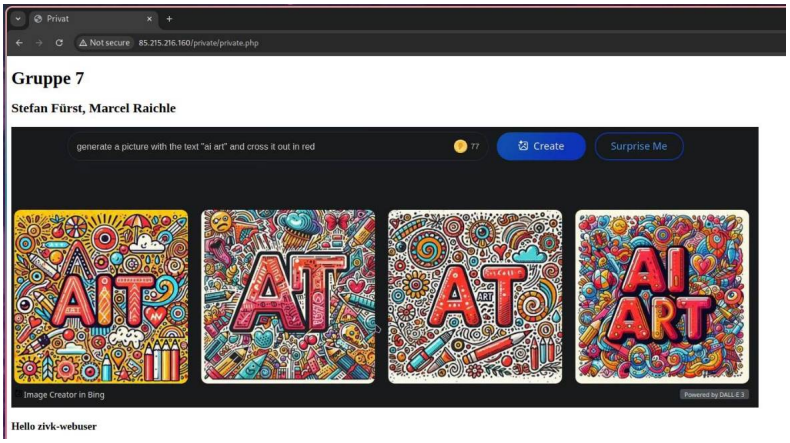


Figure 12: Logged in as zivk-webuser

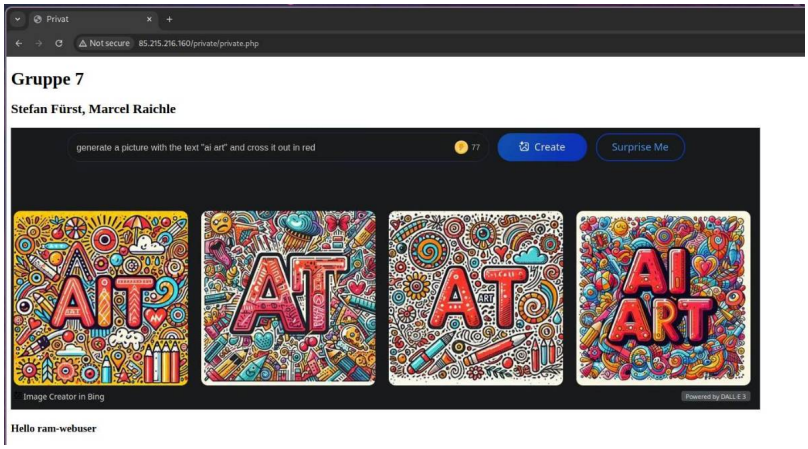


Figure 13: Logged in as ram-webuser

This is still only an HTTP site, though, which means that everything is transmitted in plain text. As a result, with a packet analyzer like Wireshark, the clear-text login credentials can be viewed. To fix this, HTTPS needs to be enabled, which will be covered in the next section.

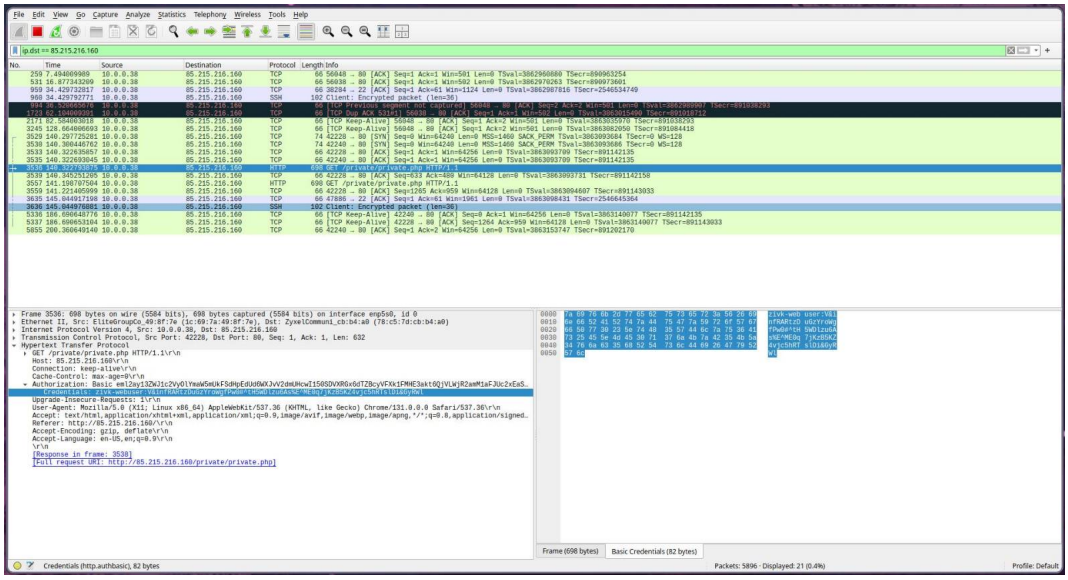


Figure 14: Reading the plaintext credentials of zivk-webuser

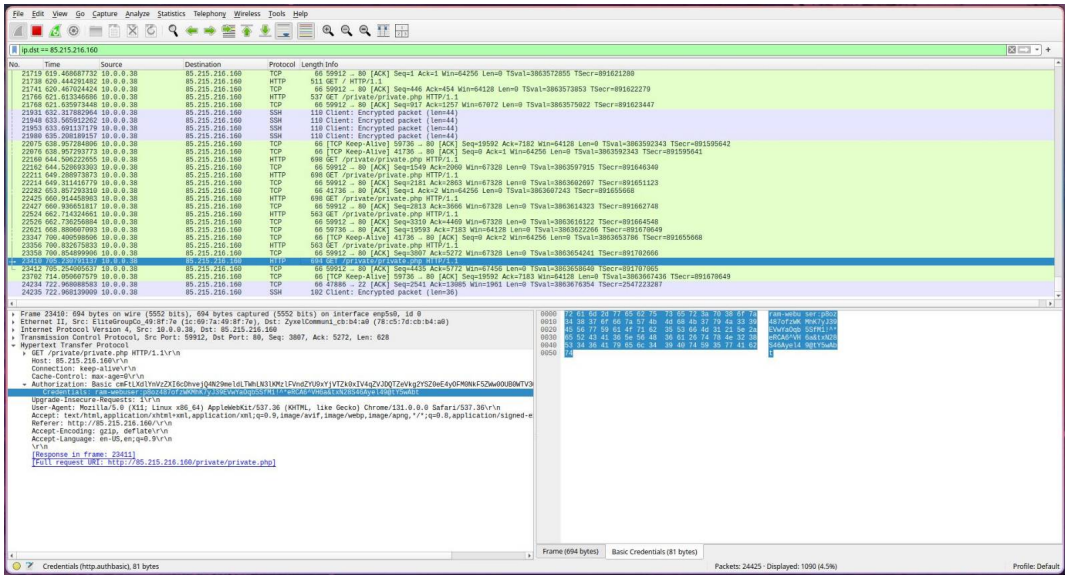


Figure 15: Reading the plaintext credentials of ram-webuser

4.5 Configuring HTTPS with Self-Signed Certificates

To stop an attacker from being able to read the credentials, HTTPS needs to be enabled on the server to encrypt the HTTP traffic with TLS (Transport Layer Security). Before this can be set up, an SSL certificate must first be created. To create a self-signed SSL certificate, the cryptographic toolkit `openssl` is used with the following command:[8, 9]

```
RUN openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -keyout /etc/ssl/private/nginx-selfsigned.key \
    -out /etc/ssl/certs/nginx-selfsigned.crt \
    -subj "/C=AT/ST=Vienna/L=Vienna/O=RAM/OU=7/CN=metyr.xyz/\
          emailAddress=wedm1ebmf@mozmail.com"
```

The options used are the following:

```
openssl #command-line tool to generate and manage OpenSSL certificates,
        #keys, and other files.
req #subcommand to specify useing an X.509 certificate signing request(CSR)
-x509 #telling openssl that we want to create a self-signed certificate
      #instead of generating a signing request
-nodes #tells openssl not to secure the key with a passphrase since nginx needs
       #to be able to read the file without user intervention
-days 365 #sets the length of time that the certificate will be considered valid
-newkey rsa:2048 #creates a new 2048-bit long rsa key at the same time, which
                 #is required to sign the certificate
-keyout #output location of the private key
-out #output location of the certificate
-subj #sets certificate subject
```

The `-subj` flag is required to fill the fields of the certificate without user interaction. I put the abbreviations used in the command into square brackets and added the values I included in the command.

```
Country Name (2 letter code) [C]: AT
State or Province Name (full name) [ST]: Vienna
Locality Name (eg, city) [L]: Vienna
Organization Name (eg, company) [O]: RAM
Organizational Unit Name (eg, section) [OU]: 7
Common Name (e.g. server FQDN or YOUR name) [CN]: 85.215.216.160
Email Address [emailAddress]:wedm1ebmf@mozmail.com
```

Now, in the Nginx configuration file, we need to make the server listen on port 443 and add the SSL certificate and key.

```
server{
        ...
        #making the server listen on port 443 for ipv4 and 6
        listen 443;
        listen [::]:443;
        #setting the ssl certificate file
        ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
        # setting the ssl certificate key file
        ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;
        ...
}
```

After setting up `https`

```
server {
        listen 80;
```

```
        listen [::]:80;
        server_name _;
        return 301 https://$server_name$request_uri;
}
```

If we reload the Nginx configuration, our browser is going to give us a security warning since it recognizes that the certificate was not signed by a trusted organization but by ourselves.

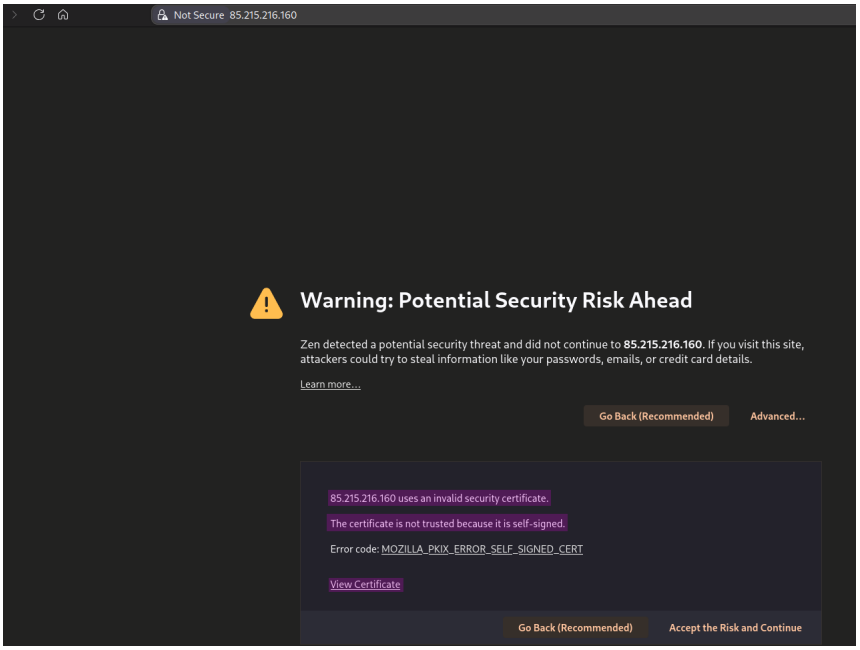


Figure 16: Browser warning for untrusted certificate

If we accept the risk and continue, we can inspect the certificate by clicking ”View Certificate.”

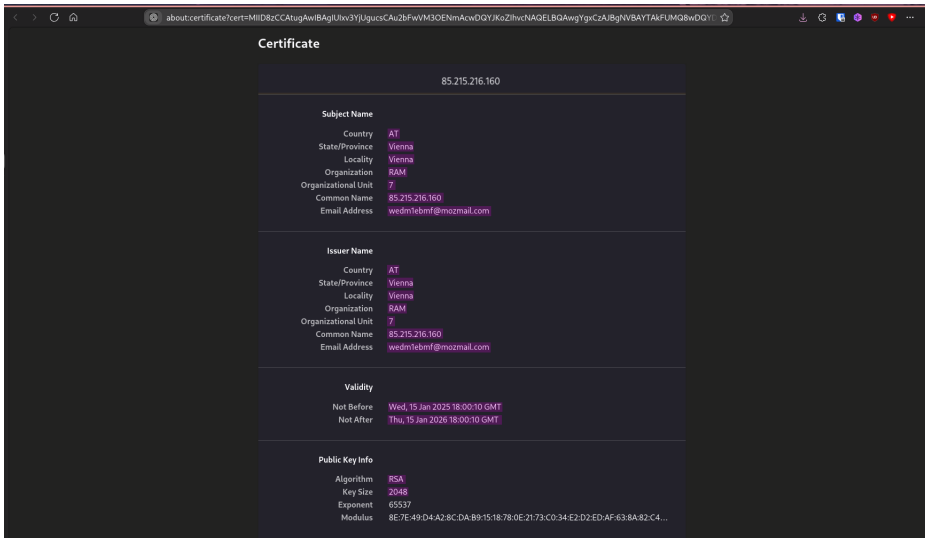


Figure 17: Viewing the self-signed certificate

If we open up Wireshark and inspect our traffic, we can see that we can't view any HTTP traffic. Instead, we only see TLS packets, which contain the encrypted HTTP data, and therefore the credentials can't be viewed anymore.

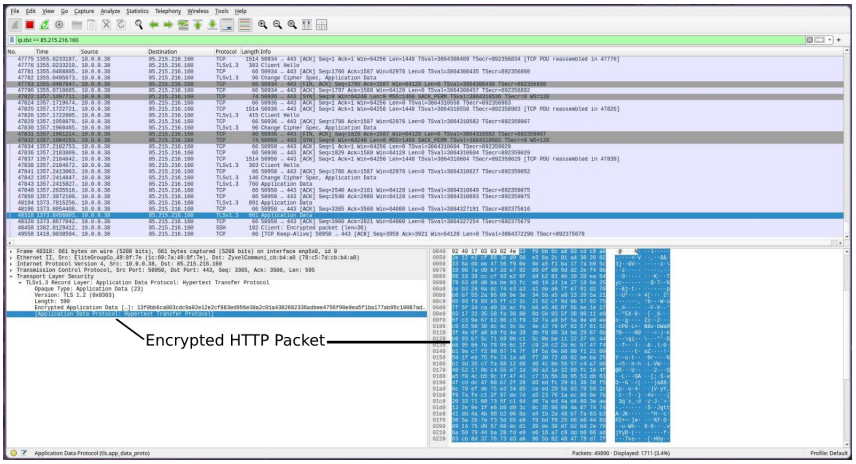


Figure 18: Not being able to see the credentials anymore

4.6 Adding a Domain

Since I am doing this on a public VPS, I can't use a local DNS and need to use a real domain instead. I bought `metyr.xyz` from `https://www.namecheap.com/`. To make Nginx use the domain name, you have to set the `server_name` in the configuration from `server_name _;` to `server_name metyr.xyz www.metyr.xyz;`.
Now we need to create a DNS record for our domain.
This record needs to be of the `A` type, which returns a 32-bit IPv4 address and is commonly used to map hostnames to an IP address. [10]
The `@` in the Host field is used to denote the current origin, which represents the current domain. In this case, it would be `metyr.xyz`. [11]

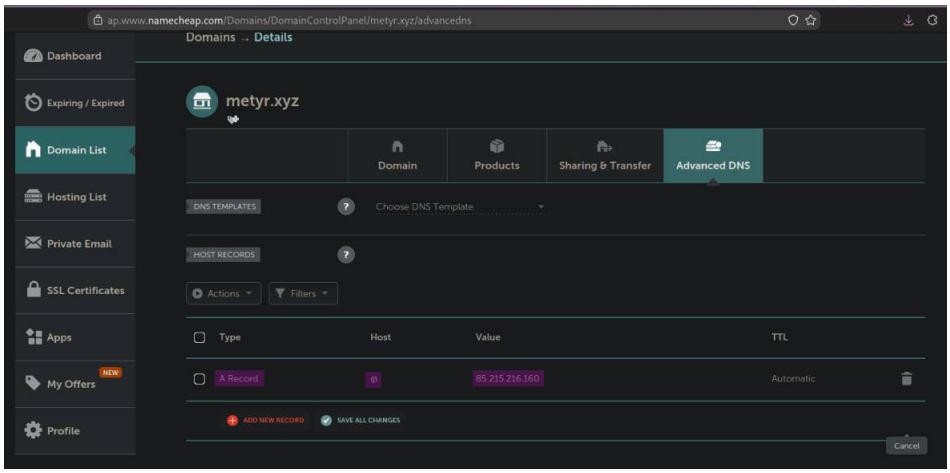


Figure 19: Setting up the DNS record

Lastly, I want to switch from using a self-signed certificate to using an officially signed one by Let's Encrypt. For this, the `certbot` and `python3-certbot-nginx` packages need to be added to our system. Now we can run this command to generate an SSL certificate, which will be signed by Let's Encrypt, so the browser won't give us a security warning anymore.
`certbot --nginx -d metyr.xyz --non-interactive --agree-tos -m wedm1ebmf@mozmail.com`
The options used are the following:[12]

```
--nginx #use the nginx plugin for authentication & installation
-d #comma-separated list of domains to obtain a certificate for
--non-interactive #run non-interactively
--agree-tos #agree to the acme server's subscriber agreement
-m #email address for important account notifications
```

After running this command for the first time and replying if you haven't saved the certificate, you can use the `--force-renewal` flag to forcefully renew the certificate in case you lost it or don't want to set up importing it on a rebuild.[13]
If we visit the website now, we can see that we won't be prompted with a security warning. If we inspect the certificate, it will show that it was issued by Let's Encrypt and is trusted.

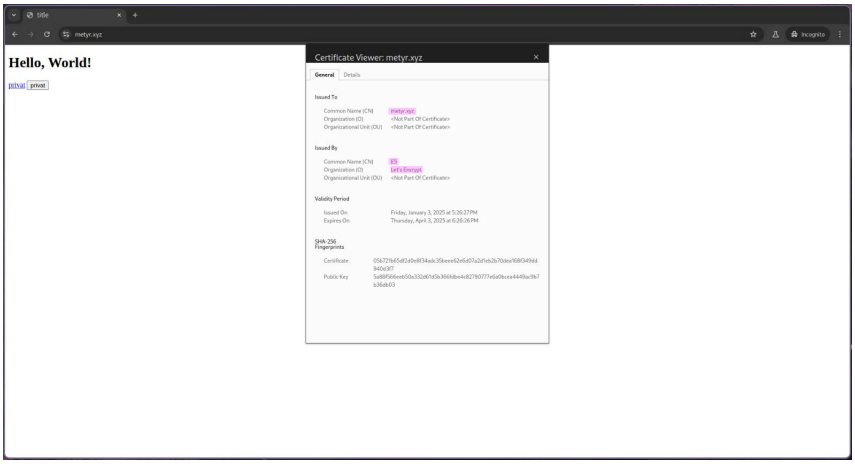


Figure 20: Showing the trusted certificate signed by Let's Encrypt

5 References

References

[1] “”Docker Compose”,” Nov. 2024, [Online; accessed 4. Jan. 2025]. [Online]. Available: https://docs.docker.com/compose

[2] “Docker ARG, ENV and .env - a Complete Guide,” Jul. 2017, [Online; accessed 5. Jan. 2025]. [Online]. Available: https://vsupalov.com/docker-arg-env-variable-guide

[3] “Restricting Access with HTTP Basic Authentication | NGINX Documentation,” Jan. 2025, [Online; accessed 4. Jan. 2025]. [Online]. Available: https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-http-basic-authentication

[4] “htpasswd - Manage user files for basic authentication - Apache HTTP Server Version 2.4,” Apr. 2024, [Online; accessed 5. Jan. 2025]. [Online]. Available: https://httpd.apache.org/docs/current/programs/htpasswd.html

[5] “echo(1) - Linux manual page,” Jan. 2025, [Online; accessed 5. Jan. 2025]. [Online]. Available: https://man7.org/linux/man-pages/man1/echo.1.html

[6] “NGINX location blocks: Understanding and Utilizing URL Matching - Sling Academy,” Jan. 2025, [Online; accessed 5. Jan. 2025]. [Online]. Available: https://www.slingacademy.com/article/nginx-location-block-the-complete-guide

[7] “Getting basic-auth username in php,” Jan. 2025, [Online; accessed 5. Jan. 2025]. [Online]. Available: https://stackoverflow.com/questions/316847/getting-basic-auth-username-in-php

[8] “How To Create a Self-Signed SSL Certificate for Nginx in Ubuntu 20.04 | DigitalOcean,” Dec. 2024, [Online; accessed 5. Jan. 2025]. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-in-ubuntu-20-04-1

[9] admin, “HowTo: Create CSR using OpenSSL Without Prompt (Non-Interactive) - ShellHacks,” *ShellHacks*, Mar. 2017. [Online]. Available: https://www.shellhacks.com/create-csr-openssl-without-prompt-non-interactive

[10] Contributors to Wikimedia projects, “List of DNS record types - Wikipedia,” Dec. 2024, [Online; accessed 5. Jan. 2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_DNS_record_types&oldid=1260647885

[11] “RFC 1035: Domain names - implementation and specification,” Jan. 2025, [Online; accessed 5. Jan. 2025]. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1035#page-35

[12] “certbot — Certbot 3.1.0.dev0 documentation,” Dec. 2024, [Online; accessed 5. Jan. 2025]. [Online]. Available: https://eff-certbot.readthedocs.io/en/latest/man/certbot.html

[13] V. Gite, “How to forcefully renew Let’s Encrypt certificate on Linux or Unix,” *nixCraft*, May 2024. [Online]. Available: https://www.cyberciti.biz/faq/how-to-forcefully-renew-lets-encrypt-certificate

6 List of figures

List of Figures