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: Dumm und Dümmer/7

htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Supervisor: SPAC, ZIVK

Exercise dates: 3.1.2025, 4.1.2025

Submission date: 4.1.2025

htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Contents

1 Task definition

2 Summary

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.

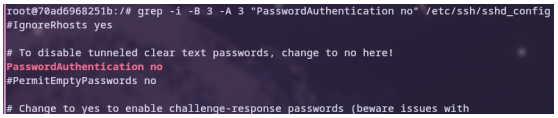

Figure 3: Password authentication disabled

4.2 Testing the SSH connectivity.

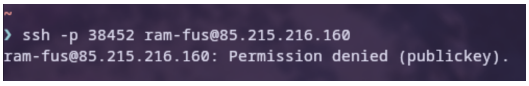

Figure 4: No SSH key available

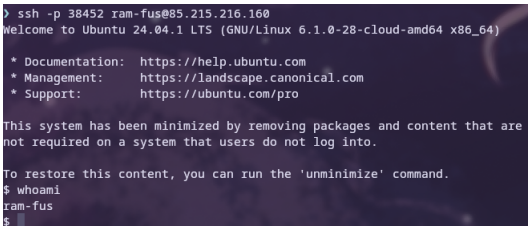

Figure 5: ram-fus authenticating via SSH key

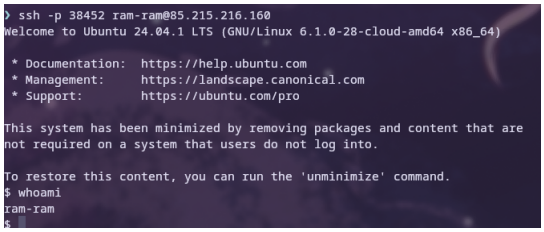

Figure 6: ram-ram authenticating via SSH key

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"
```

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

```
#!/bin/bash
alias relaunch="sh -c 'docker stop itsi && docker rm itsi &&\
                       docker buildx build -t itsi:latest . &&\
                       docker compose up -d && docker exec -it itsi /bin/bash'"
alias rebuild="sh -c 'docker buildx build -t itsi:latest . &&\
                      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.

```
...
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. [1]

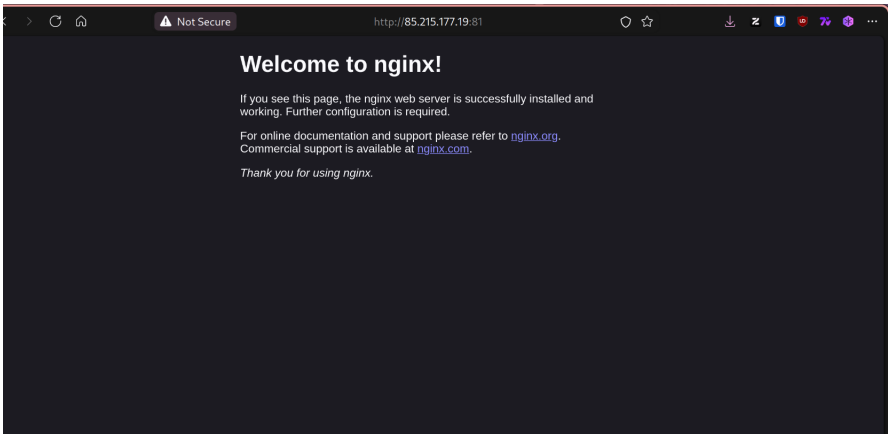

Figure 7: Default nginx site

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.

[1] 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.

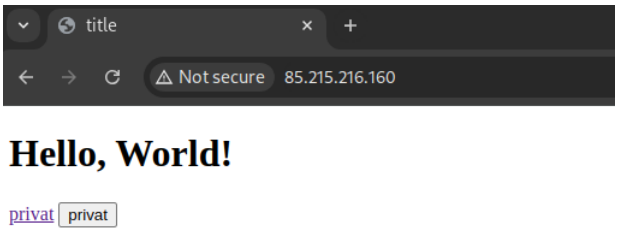

Figure 8: Viewing the index of the website

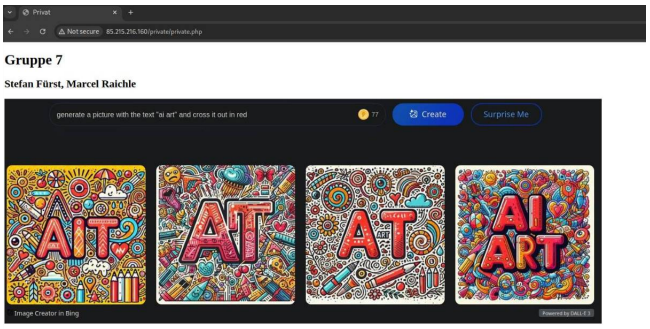

Figure 9: Viewing the private part of the website

4.4 Securing Nginx with Basic Authentication

4.5 Configuring HTTPS with Self-Signed Certificates

4.6 Adding a Domain

5 References

References

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

6 List of figures

List of Figures