jamk

Hardening- Lab 5

Group 4

Nguyen Ngo - AE8880
Dung Doan - AA7785
Syed Fawaz - AD9946
Jasper (Franciscus) van de Klundert - AG9056
Sanka De Silva - AC4892

Exercise Lab 5

Hardening

21/4/2025

Bachelor’s Program of Information and Communication Technology

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

Contents

Figures

1 Introduction

In this lab, the purpose of this exercise is to explore **Multi-Factor Authentication (MFA)** and configure it for use with **WordPress** and **SSH login** to the web server. The exercise will be carried out in the **VLE environment**, as illustrated in **Figure 1**.

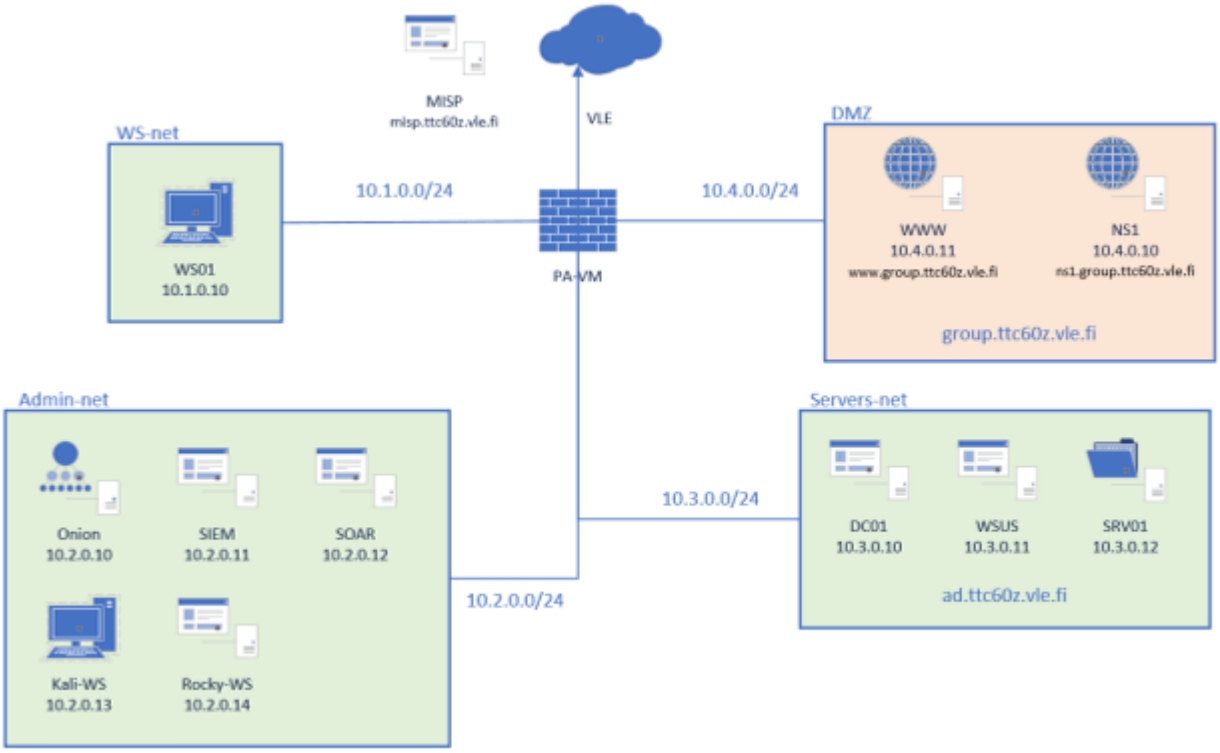

Figure 1 VLE Environment

2 Theory of the Lab

2.1 MFA

Multi-Factor Authentication (MFA) is a security measure that verifies a user's identity using two or more authentication methods. Typically, this means that in addition to a username and password, a third form of authentication is required—such as a code sent to the user's phone or an approval via a mobile app. (Stephen, 2025)

The main goal of MFA is to prevent unauthorized access to user accounts and enhance overall security. Even if a password is compromised, an attacker cannot log in without the additional authentication method. This significantly reduces the risk of phishing attacks and password breaches. (Eliza, 2025)

MFA is especially important for services that handle sensitive information, such as email accounts or internal company systems. It is an effective way to protect user accounts and minimize their vulnerability to cyberattacks. (Stephen, 2025)

2.2 Google Authenticator

In both parts of the lab, we'll use the **Google Authenticator** app to enable multi-factor authentication. Once configured for **WordPress** or **SSH**, logging in will require not only a password or key but also a code generated by the app on a mobile device.

Google Authenticator generates **one-time passwords (OTP)** using two algorithms: **HOTP (Event-based)** and **TOTP (Time-based)**.

1. **Secret Key**

   Both the server and the client (Google Authenticator) share a **secret key**, securely stored on both sides. This key is usually provided in the form of a **QR code**.

2. **HOTP (Event-based)**

   HOTP uses the secret key along with a **counter**. Every time a new password is generated and used, the counter increases. Both the server and client stay in sync by tracking this counter.

3. **TOTP (Time-based)**

   TOTP also uses the secret key, but instead of a counter, it relies on the **current time**. Both the client and server synchronize time—typically using **Network Time Protocol (NTP)**—to ensure the generated codes match.

4. **Code Generation**

   Google Authenticator generates a 6-digit one-time password by combining the secret key with either the time or counter through a secure algorithm. The user enters this code during login, and the server verifies it by calculating the same code.

5. **Security**

   Because the secret key and time/counter are shared securely, both the client and server can generate matching codes **without needing to communicate directly** during login. This makes the system highly secure.

(Udoh, 2023)

3 Lab Work Progress

3.1 Adding 2fa plugin to Wordpress container

We began adding multi-factor authentication to WordPress by logging into the WordPress admin dashboard at:
www.ttc60z.vle.fi

From the **Plugins** tab, we searched for **“WP 2FA”** and clicked **“Install Now”** to install the plugin (see **Figure 2**).

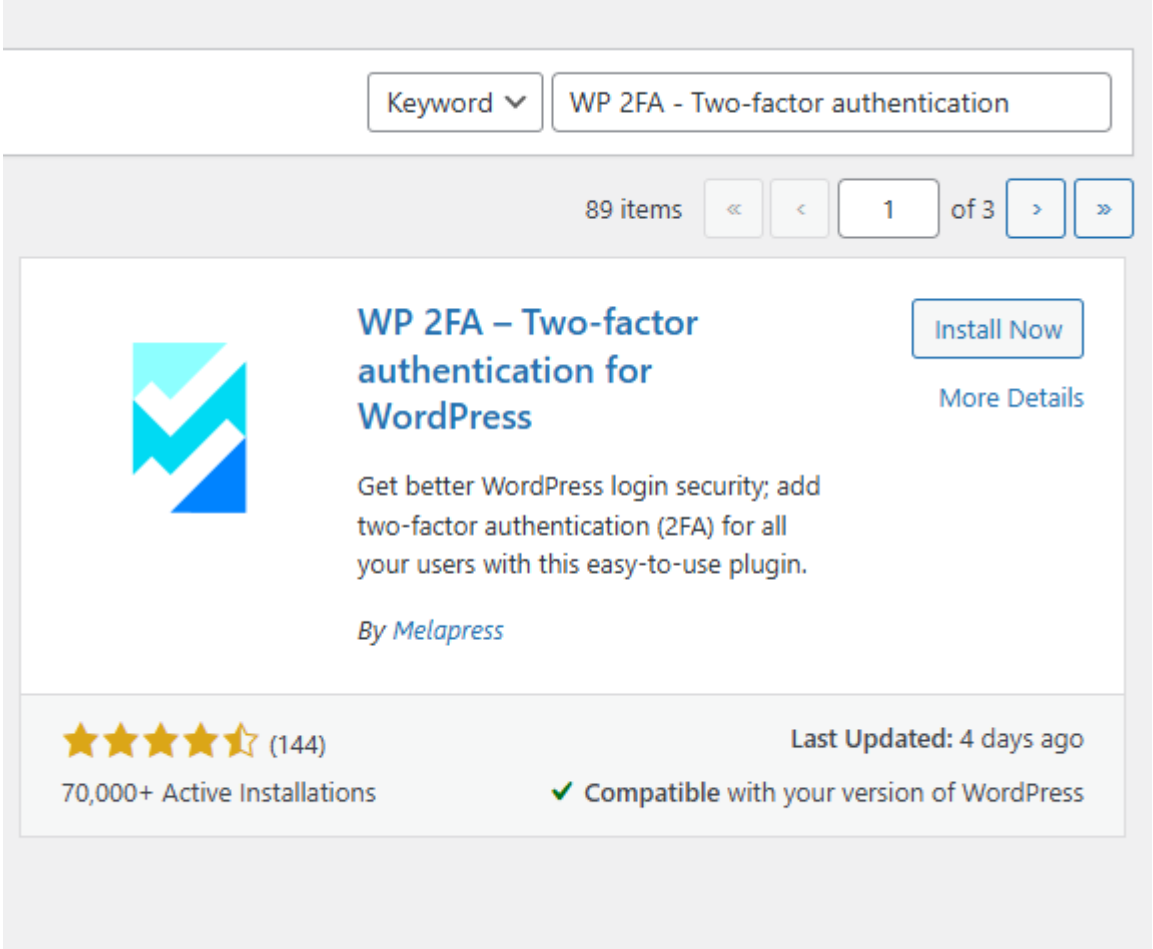

Figure 2 WP 2FA installation

According to the instructions, we were supposed to encounter an error at this stage stating that the folder could not be created. However, we had already completed this step during a previous

exercise when updating plugins. (Figure 3)

Figure 3 Successfully installed

After installing the plugin, we clicked activate and automatically moved to the Setup Wizard. (Figure 4)

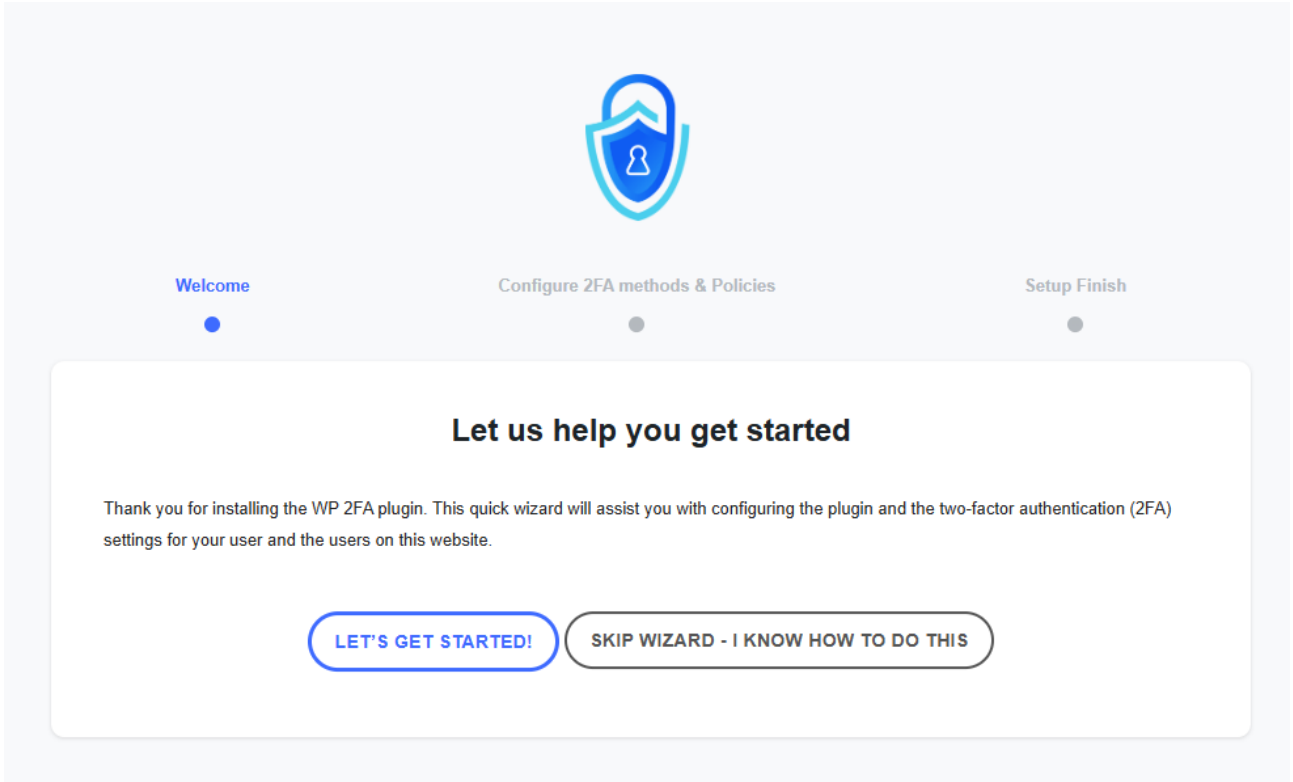

Figure 4 Setup wizard

In the next step, we did not enable email verification—instead, we only used **one-time codes** generated by the **authenticator app**. (Figure 5)

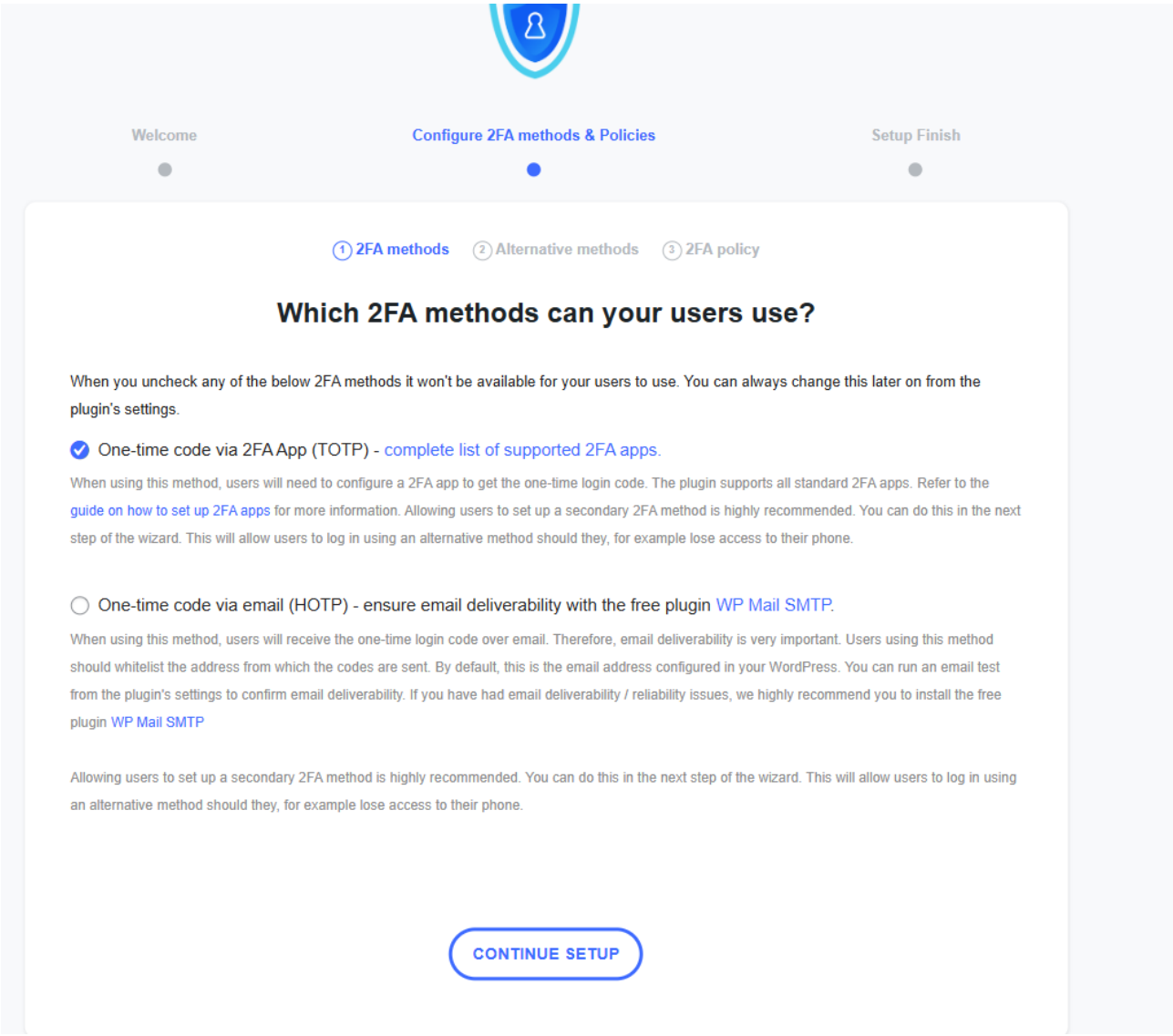

Figure 5 Configure methods and Policies

We then enabled the option that **requires all users** to use **multi-factor authentication**.(Figure 6)

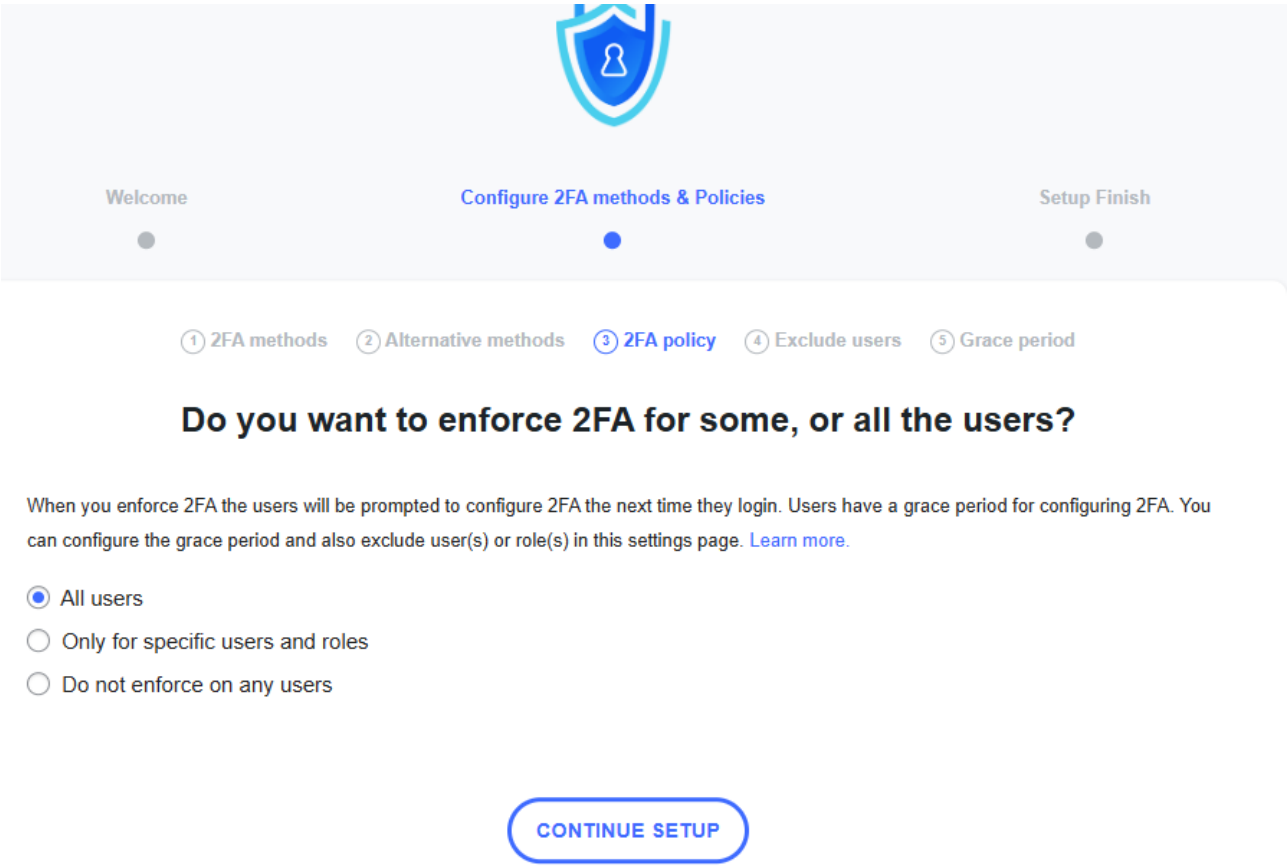

Figure 6 All user enforcement

Next, we selected the option that **requires users to enable multi-factor authentication immediately**, with **no grace period**. (Figure 7)

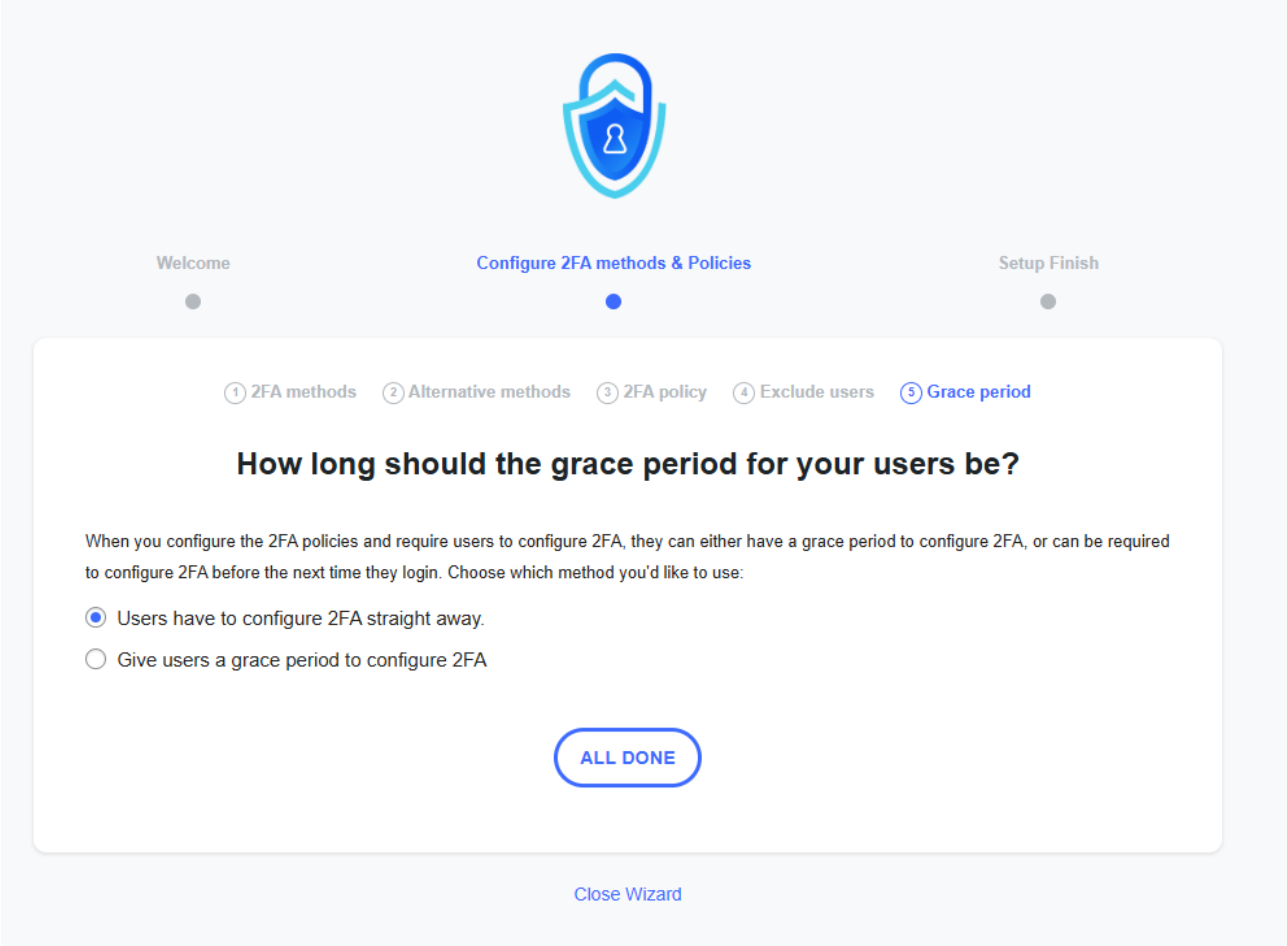

Figure 7 Grace Period Choice

We clicked **“All done”** and then selected **“Configure 2FA now.”**

As the authentication method, we chose the **Google Authenticator app**.

We scanned the **QR code** (shown in **Figure 8**), and received a **6-digit code** in the authenticator app on our phone.

Additionally, we created a **dedicated email address for our group** to use for authentication purposes.

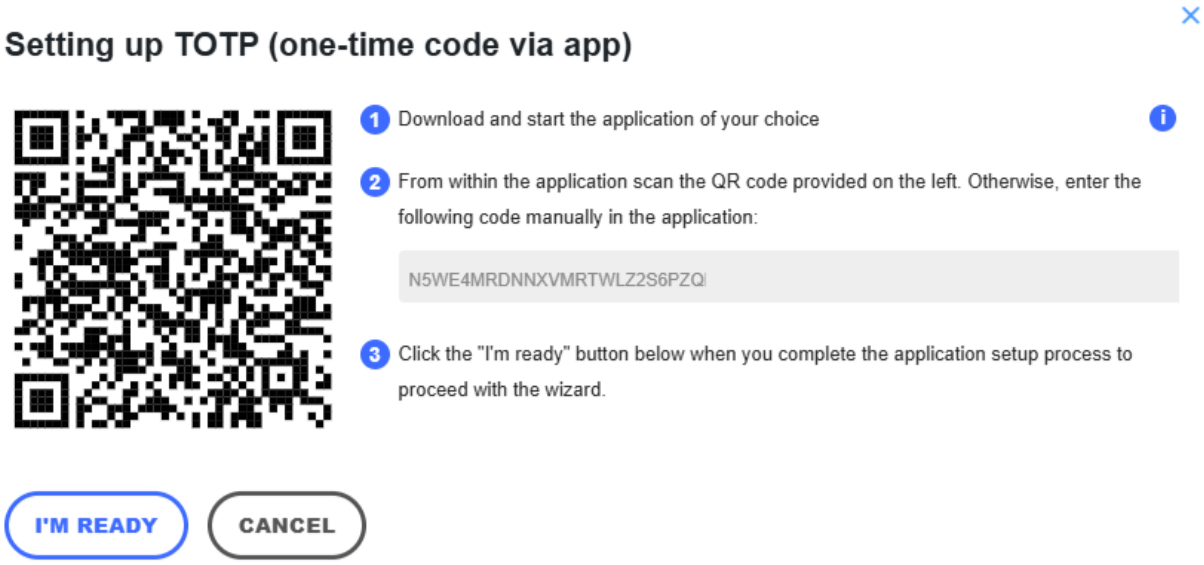

Figure 8 QR code for implementing the application.

Then we need to add the code from the google authenticator app to move to the next step. After entering the code from the authenticator app, we were given the option to **generate backup codes**. We chose to **generate them later**. (Figure 9)

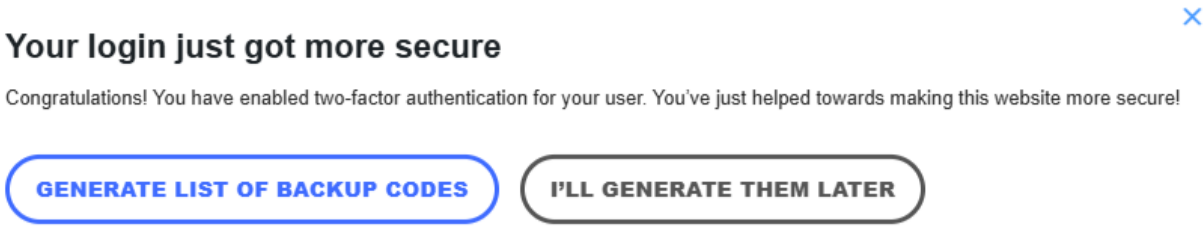

Figure 9 Backup code generate

Next, we tested the login using **two-factor authentication**. When logging into the **WordPress admin dashboard**, we were prompted to enter an **authentication code**. (Figure 10)

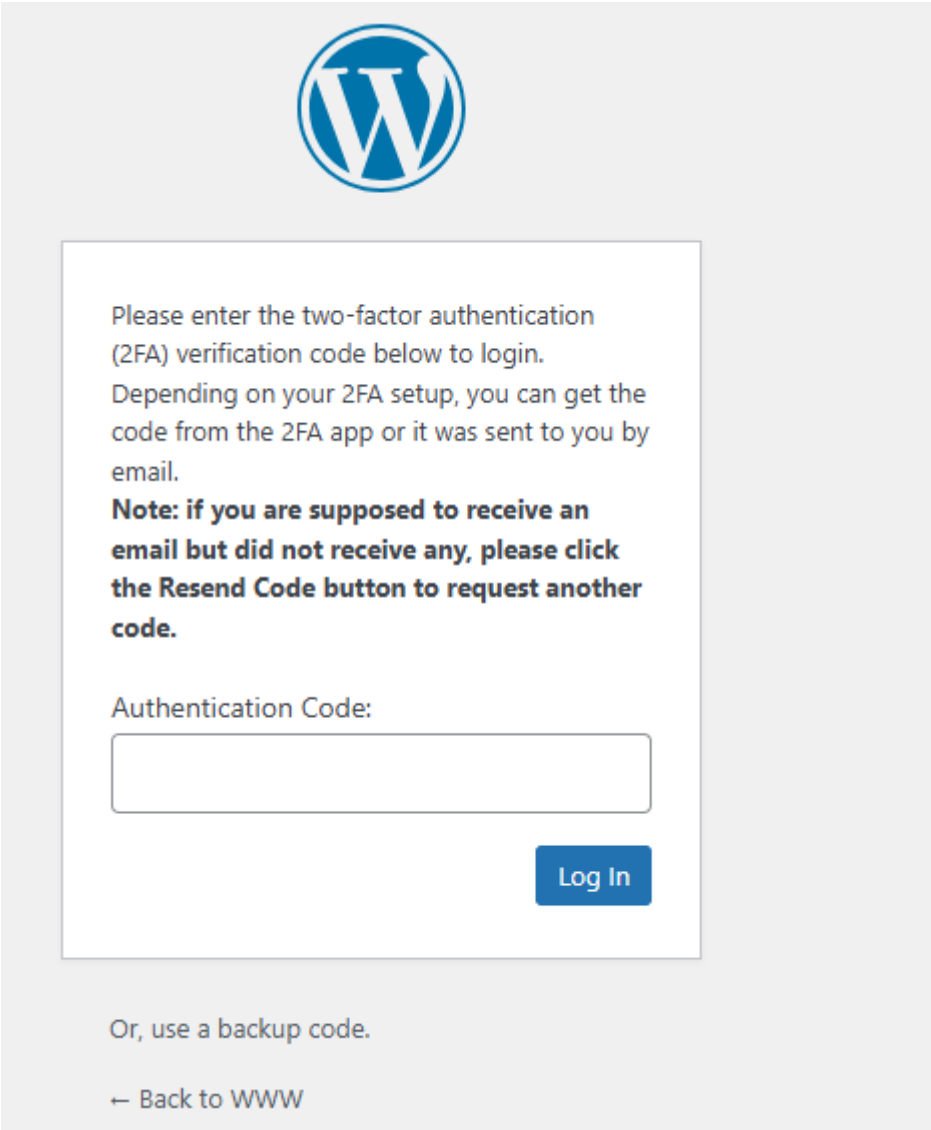

Figure 10 Authentication Code Requirement

From this, we entered the 6-digit code displayed in Google Authenticator and accessed the control panel.

3.2 Adding 2FA to Centos

The next task was to configure **multi-factor authentication for SSH login** to the web server.

We installed **Google Authenticator** on the server using the commands shown in **Figure 11**.

```
sudo dnf install -y epel-release

sudo dnf install -y google-authenticator qrencode qrencode-libs
```

Then run the `google-authenticator` command to create a new secret key in the `~/.ssh/` directory.

```
google-authenticator -s ~/.ssh/google_authenticator
```

Figure 11 Instructions for Installing multi-factor authenticator for ssh login

The **last command in Figure 11** generates a **secret key** for authentication and saves it in the **.ssh** directory under the name **google_authenticator**.

Next, we were asked whether we want the authentication tokens to be **time-based**, and we answered **yes**. (Figure 12)

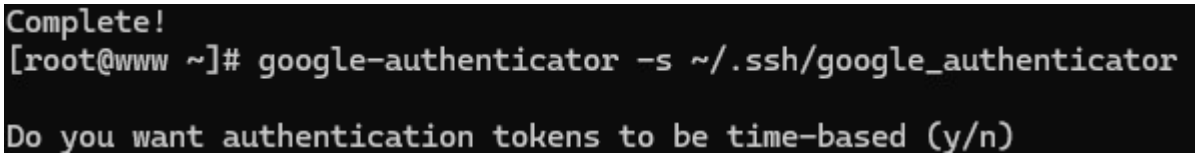

Figure 12 Time-based tokens

After responding to the prompt, a **QR code** appeared in the terminal. We scanned it using the **Google Authenticator app** on our phone. (Figure 13)

Figure 13 QR code on a web server

We entered the code displayed in the app into the **command line**. We were then given a set of **emergency backup codes**, which we saved in a **secure location**. (Figure 14)

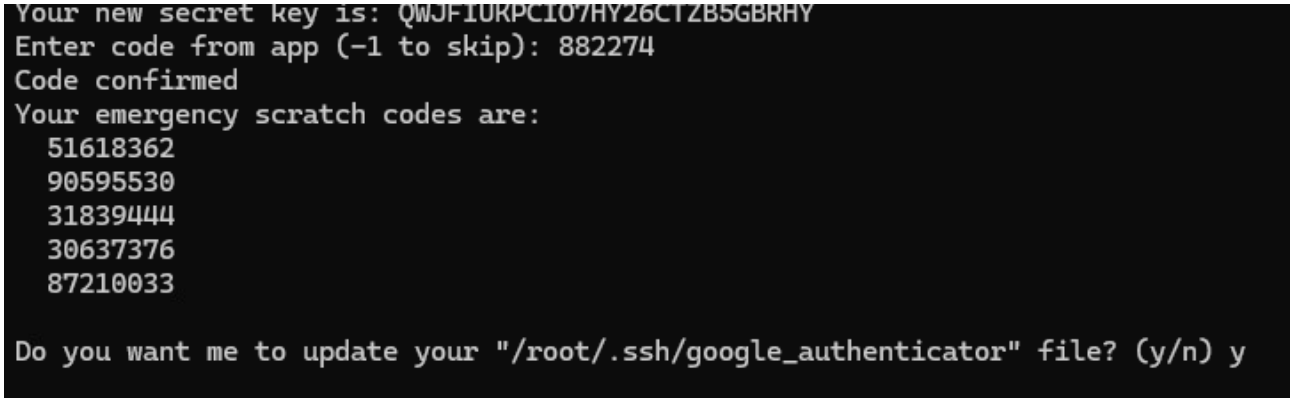

Figure 14 Emergency codes

Next, we answered **"yes" to all the prompts** shown in **Figure 15**.

These questions were related to **security settings**. For example, the final question asked whether to enable **rate limiting**, which helps protect against **brute-force attacks**. The first one is about the multiple access using the same token to prevent the MITM(Man-in-the-middle) attack.

Figure 15 Security settings

Next, we configured the **SSH daemon to use Google Authenticator** for login.

Our group had previously been using **SSH keys** for user authentication, but during the final stages of the previous exercise, we had to **reset the web server**. So, we started by reconfiguring SSH key access.

As an example, we created an **SSH key for the root user** by running the command: ssh-keygen. (Figure 16)

```
[root@www ~]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:6JR1oUUfw4eqkWrBYhnpCn5uN/dj/5qhthztuvdRLo0 root@www.group4.ttc60z.vle.fi
The key's randomart image is:
+---[RSA 3072]----+
|      .   .+.o.  |
|    o    o ooo.  |
|   . +  o.....   |
|.   = o+o..      |
|.. o .+oSo     . |
| ... oo . .   =  |
|  o  ..  . o E o |
|   o o ..++.o o  |
|  . . o +BB=+o   |
+----[SHA256]-----+
```

Figure 16 ssh-keygen command

We copy the public key to the authorized keys folder using the command cp id_rsa.pub authorized_keys. We also copied the key id_rsa to WS01 for the Putty login. (Figure 17)

File Edit View

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABCCaHdcaX
fPPRPH/26wmAxhAAAAEAAAAAEAAAGXAAAAB3NzaC1yc2EAAAADAQABAAABgQCuwUirT+zu
fYlGXt9iCtvrRTuECuQQBMTA0sQQySanJ3W19vxPEaixWGBEE1J46RctRKvqKFATBFfYf7
b6cRWlnv8vlGWcWi+y8G3Yq1d3ai/CfY2ynkupO01eK224y8uoq7anYT9dZPew1IYVauyh
O/YJZso62J70s7za44R2s/G47aL6XDGy0/l3EOPhVYRSdRt+XkIjT67kIvekn3BAGRZnoF
tfJNMw6XfUZ2xgczePkj3ZnnNhoEM86s44R5DNkD4u1sKKxLW3hAJNH23dAaxBUP2HH+UD
e2kSS0ITbIx9HTlyK5dW+H8jEJKoexqgXllITC3bTJ92DmbB6t6cmmGB7Js0O6CfXAMf/+
BwhwORNndgQ0qxGGpk6xDdB+1K50f1LIOkXpc2ovO9MlPfs4EO6t0KCPPtdQwJWxYcjlIT
n9TjVNXkXIaKKSYTX0WDSqAk1bTe808uUm5ikxBV122LVkX0/b2xbh0NTyrBZwXJpAvC4G
mjVUBDvOAGfjMAAAWgS2kS9t69tqPb2Bmj97xC5Bz4L0AAk17D5HKK8SSTbPUnOdvTXxOA
gX9rkSFOYHAswioN6iiONm8FNJFgR0mDRwhSO8sy743ShFke/hMiZgqEOl102mqUMcuvOz
yDU+fsFqoM6IKUpRLznUBjAUiwe+7jG5ovnj4/hwNE8dXyqEfidGir0Ylfv8VziA9mEhXz
5lcADLMFczQGWZP+f2G3NKmNWLuDu5spSc9SUBDEO+41Fi7OfIlcrJXz7Bh6HWM/gXj26f
BxUMwFNfu+C4RF/5tEWCX3Q9Z11yL+IPMnLF2sJ8XpgFTROp7ZqCCIDDXvqXIhXnU2/2Pg
FKNX38h0bYoG0Ty2JBPVNxVf1Ii2Ho9J1fY9ZC1cplu7wd6uzHWlDbR69Fz2Pk/1v9iCjs
9vvQG96dtWrIcA/ZVmPZGUrQp+/XzPm7bc1ei45AFS3w3GGki08TEa1ZbcpwtcUxhu79Zs
U12gOxUB2fWFQOGeIsKDVjDDurg6jLIj+TwM323aQCtA1O82k8gZMIv9PbiStXHP8mA8MP
fSI6h4WsratYMG1rVbvZG7zaYSDPIz0Mzaa0plN5Kv+NLpWU3RxKH9DMlgN4xtdjBd36N1
9/jw/6UqzGvlDbGGFeq+d7LFA5cJNyP8Gsud3porZvAZmzmSIJqnL7qn6LTuiFjB3n50wX
a7azWMwiBT3EBKumAKUxSY8gNOYvMQ7+j530dxfgT3LdZ5D6Dk/LushhtsPmxt7kU14Ybd
pDOzT9wR1S6D6nohDzuD9dobDgJhkogL4THRexck3K55KNCvodOrG/RnxgzN9mAVcK8dHp
h23DE+5SK4qSuISSCQYKKiWWSdNkgn5f1m66IR1fYJHTHAJr92nxvTcii1GsRxLQ+7sNNx
kRVL8sohSBQorR888/KK7Unu+hZKPcPPQrQK2mxfgKGUGB11sA1yndikHQyyTiPkyVhV8S
XhKJ5VxKwOfc3fee9qM9/dPQ0cfc1eEKK1PJcOsx6aFiKscS/CS2snGFAir1KykxWJBXkb
D2IaJxbggNjQ6jFTd5Q46C7NIcohY9cQLlot5i5c2dDIVN9oZz3V6OuIKP+20rt1IZ1wzD
GOstScYUsRU6/mjT0Mrf13S298Vvg+Tr6S/5DSkQ5TwsZRyFhOZKr3UCaVkW1ExtY2/pts
pCvXoyZO1X6ud1B1BenAwg91Cq8hVYEndVAEDCO+8AO1i1TWqwemCasrkui/2BFvFGANkY
1myGz3mV3yKyzqpjjUPyJRR1Z4brsKYcbxfNBWyC7eYEC3Tvbh6ryZqj7cVTeBhl2m8p91
hTWg6PzTT8VQDcrtIg+NY/VoKfCqqK1F76IHDnFTZQwsVaJxIW+BSH+p8yC4mTOm4e6ef/
Y/6T1Z+MSusptM5wZ3wnadIS7AhSjQ5yIpoPOkwfsGcXLA60PUrqhaPL6XcbRr+b+Cfo5S
G0ZLnjTHuGyrvKEqDzAKHmaW6u8V1n32WodczqbOzg4GVWQXmJ6vzOkptqyQsDenLBVhvg
AIMFL/Msyi2oLYVQbeeMeXiWpvdLfdIR/eCI13ZKOgZBrrtObc79HE/yooI9QbHlJGWOLC
Afy7CAM34f4lzI6jaNWMIa6skSMI3qDMbfGsWrhxR4hLOPknYpTqy94kRie1gguacjWOwk
0kr9rFfwtwDeeZ5HRa2Y2RatjeOkciHiuqOSFh0GZo7uYafZ3EDHdIkS+yWKKsfPD9tvx0
zeIKyMtEgSWdEnbjhfVjtoY8GDlDuTKFiyelIwgb1mgyGX2tPfQyrFqA+gDfXmNbpaUT8e
RwRt+3UDSogayjZ+/MoCvvJOdb9xbUoG+LY//jqxXor8FrNvPaWg0G06ZK2x1ZpZgY5ifx
yCV7hr3EETpWC8jzirAOrIvq7FxaQUU2EX12TanPwr4SHyXF
-----END OPENSSH PRIVATE KEY-----
```

Figure 17 id_rsa

We created the SSH key from a **text file using the PuTTYgen application** and saved it by clicking **"Save private key."** (Figure 18)

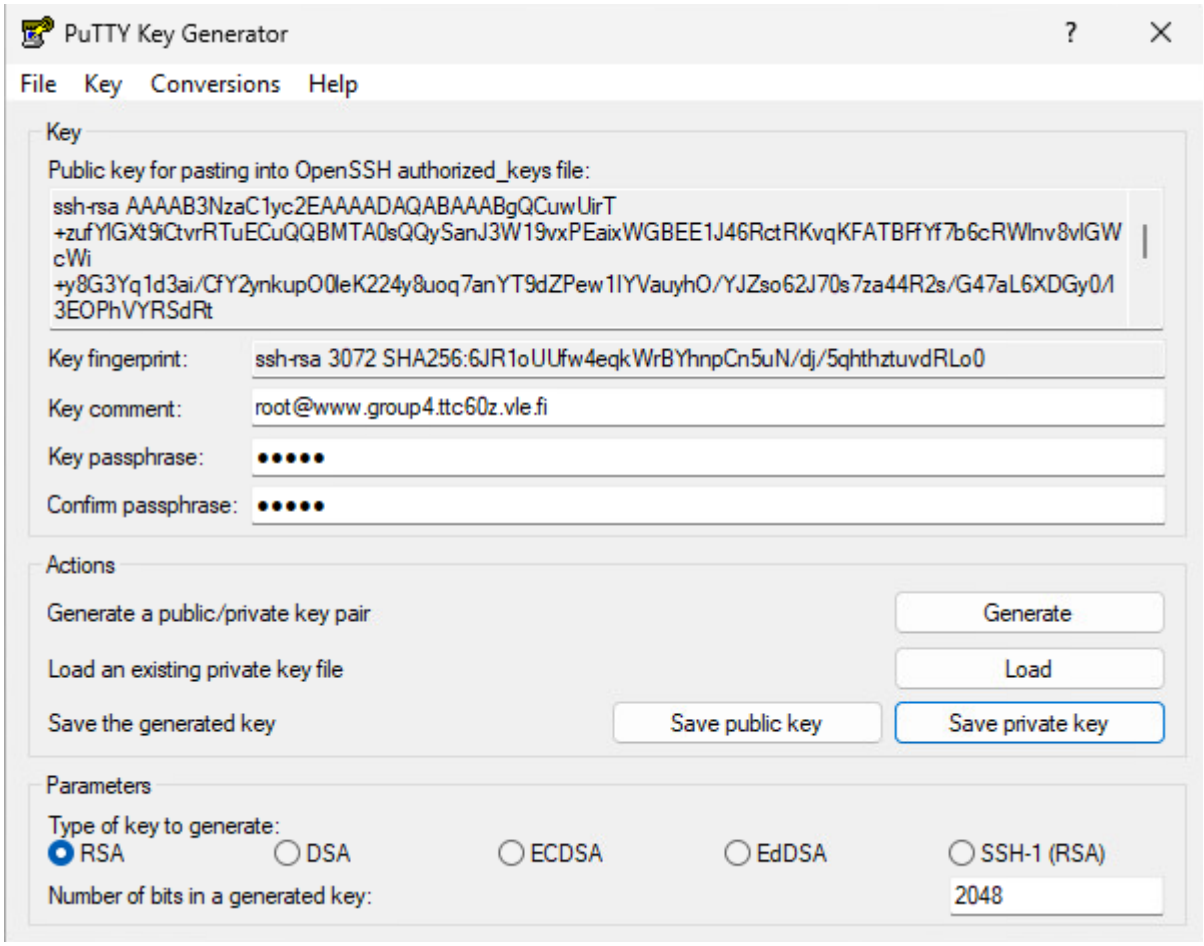

Figure 18 Puttygen

We edited the /etc/ssh/sshd_config file to require SSH key authentication when logging in. (Figure 19)

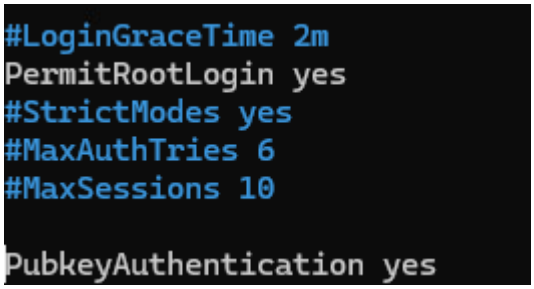

Figure 19 PubkeyAuthentication

We restarted the SSH daemon using the command: **systemctl restart sshd** to apply the new settings.

To enable **multi-factor authentication**, we made the following changes to the **sshd_config** file:

- Set **UsePAM yes** (see **Figure 20**)
- Set **ChallengeResponseAuthentication yes** (see **Figure 21**)
- Added an additional configuration line as shown in **Figure 22**

```
# problems.
UsePAM yes
```

Figure 20 UsePam setting

```
# Change to no to disable s/key passwords
ChallengeResponseAuthentication yes
#ChallengeResponseAuthentication no
```

Figure 21 ChallengeResponseAuthentication

```
AuthenticationMethods publickey,keyboard-interactive
```

Figure 22 AuthenticationMethods

Because it showed error after we restart the sshd, we edited the /etc/pam.d/sshd file by adding the lines shown in Figure 23.

```
#%PAM-1.0
auth       substack     password-auth
auth       include      postlogin
#two-factor authentication via Google Authenticator
auth     required     pam_google_authenticator.so secret=${HOME}/.ssh/google_authenticator
account    required     pam_sepermit.so
account    required     pam_nologin.so
account    include      password-auth
password   include      password-auth
# pam_selinux.so close should be the first session rule
session    required     pam_selinux.so close
session    required     pam_loginuid.so
# pam_selinux.so open should only be followed by sessions to be executed in the user context
session    required     pam_selinux.so open env_params
session    required     pam_namespace.so
session    optional     pam_keyinit.so force revoke
session    optional     pam_motd.so
session    include      password-auth
session    include      postlogin
```

Figure 23 Editing the pam.d/sshd file

We restarted the SSH daemon again using the command: **systemctl restart sshd** to apply the changes.

When we reconnected to the **web server via SSH using PuTTY**, we were prompted for **two-factor authentication**.
We entered the **verification code from the Google Authenticator app on our phone** (see **Figure 24**).

Figure 24 Multi-factor authentication when opening an SSH connection

4 Conclusion

In this lab exercise, we explored the configuration and use of **two-factor authentication (2FA)**.
To our surprise, the process was **easier than we expected**, and we completed everything in **under 30 minutes**.

While configuring 2FA for the server, we were pleased to discover that **rate limiting** is also enabled by default. This feature helps protect the server from **DDoS attacks** by limiting the number of incoming requests.

The instructions for setting up multi-factor authentication in **WordPress** were clear, and thanks to the plugin, the installation was **straightforward**.
The setup for **SSH access** was also simple—it required just a few commands and some minor file editing. This made it easy to understand what was happening at each stage of the process.

The **ease of implementation** highlights how important it is to **promote the use of two-factor authentication**, not just in corporate environments but in personal ones as well.

Many organizations still **neglect proper security practices**, and in our view, enabling two-factor authentication should be considered a **basic security requirement**.

4.1 Reflection

This lab gave us practical experience in setting up two-factor authentication (2FA) for both WordPress and SSH access. The process was quicker and easier than expected, taking less than 30 minutes.

We learned how 2FA significantly improves security, and we were pleased to see that features like **rate limiting** were also enabled during setup, offering protection against brute-force and DDoS attacks.

Both the WordPress plugin and SSH configuration were straightforward, helping us understand the underlying mechanics of multi-factor authentication.

Overall, the lab showed how simple it is to implement 2FA and highlighted its importance in both personal and professional environments. It should be considered a basic security practice everywhere.

References

Stephen, J. B., 2025. What is multifactor authentication?. Accessed on April 19, 2025. What is Multifactor Authentication (MFA)? | Definition from TechTarget

Eliza, T., 2025. Multi-Factor Authentication (MFA): Importance and Benefits. Accessed on April 19, 2025. What is Multi-Factor Authentication, and How Does it Work?

Udoh, B., 2023. How does Google Authenticator work?. Accessed on April 19, 2025. How does Google Authenticator work? | by Benjamin Udoh | Nerd For Tech | Medium