# ACME Network Setup

For more clear explanation, we consider that our internal network where everything is set up is the **192.168.9.0/24** network and the IP of the server for the containers is **192.168.9.10**. Also, the FreeIPA server, i.e the container that is going to run on the host pc, is going to be named **server.final.test** and the domain is of course final.test. The host machine runs Ubuntu 20.04 but since we run almost everything in containers, few things should change in the configurations below, which will be noted if there is a difference in other Unix distributions.

## SETTING UP THE ROUTER WITH DD-WRT:

We flash the router with DD-WRT. For this we used the recovery tool provided by Asus for their router RT-AC68U (The tool can be downloaded here and only work on Windows). We put the router in recovery mode. We connect to the router with a ethernet cable and set a static IP for our computer in the configuration panel. Then we launch the Asus' tool and when prompted add the DD-WRT's firmare for the Asus RT-AC68U. After we reboot the routers and DD-WRT is installed.

Then we did the default configuration and we changed the IP adresses of the two subnets as defined in the previous figure.

After we set everything up and change the networks provided, we need to go to **services->wireless** security and pick **wpa2-eap**, add the IP **192.168.9.10** in the FreeRadius one (leave the default port and default encryption method) and for the secret use.

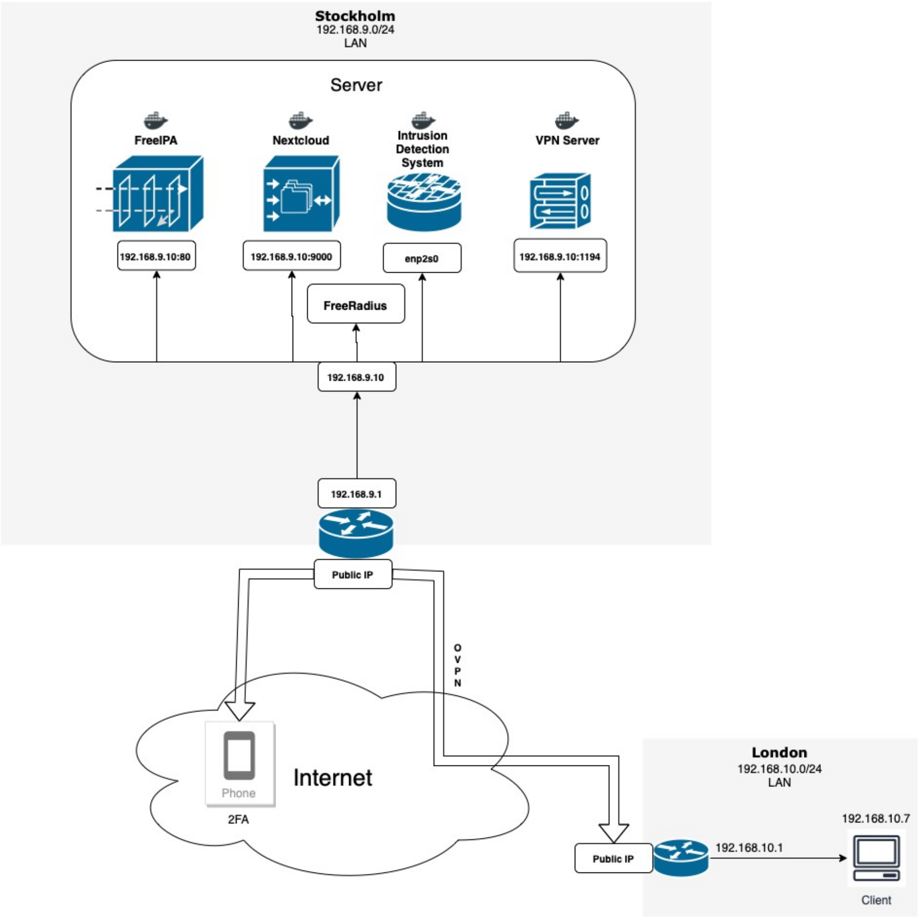

Figure 1: Final-topology

## DOCKER-ENGINE AND DOCKER COMPOSE

We install both so anyone can either chose to install every container either with docker run commands ( like we did ) or use a Dockerfile and use the docker compose up command. To install docker engine first go to the docker docs official site and follow the instructions for your distributions.

For Ubuntu Linux we do:

```
$ sudo apt-get update

$ sudo apt-get install ca-certificates curl gnupg lsb-release

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share

$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/ubuntu \ $(lsb_release -cs) stable"

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

These commands should have docker engine running: Confirm with

```
$ docker run hello-world
```

To install docker-compose:

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker- compose-

$ sudo chmod +x /usr/local/bin/docker-compose

$ sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

And to confirm you should get a result with the command:

```
docker-compose --version
```

## FREEIPA

Now that we have installed docker, we need to set up FreeIpa and Nextcloud. Before we do that though, we need to configure some details. First, in the /etc/hosts file we need to add the following entries:

```
192.168.9.10 server.final.test ipa-server
```

```
172.17.0.1 host.docker.internal
```

(needed to have communication with containers in Linux distribs)

```
192.168.9.1 <name-of-your-router>
```

( not really needed but just in case for later )

Also, if we consider that our PWD is **/home/myuser**, we create a new directory for the freeipa volumes with the command **/var/lib/ipa-data**.

Now, we run the command

```
docker run -h server.final.test --name my-group-server -p 53:53/udp -p 53:53 -p 80:80 -p 443
```

There might be a problem of port 53 already being used due to DNS and the docker command won't run. To fix that in Ubuntu, do the following:

Check firstly if that's the case with the command:

```
sudo lsof -i :53
```

If that has an output, your port is being used. You then need to edit the /etc/system/resolved.conf file to the following version:

```
[Resolve]
DNS=1.1.1.1 #pick any DNS server this is the one I picked #FallbackDNS=
#Domains=
#LLMNR=no
#MulticastDNS=no
#DNSSEC=no
#DNSOverTLS=no
#Cache=no
DNSStubListener=no
#ReadEtcHosts=yes
```

Create a symbolic link with the command `bash sudo  ln -sf /run/systemd/resolve/resolv.conf /etc/resolv.conf` and reboot your system.

Now the command

```
sudo lsof -I :53
```

should have no outputs and the docker command will run. For other distributions follow other similar guides. Also, for FreeIPA to run in other distributions you might need to disable SELinux.

If all is done properly, you should be able to type to your browser the 192.168.9.10 and go to the FreeIPA web UI, where you can connect with

```
username admin
```

and

```
password Secret123
```

You can easily users from the web UI with a certain password. Then log out and try to log in with that user. The system will immediately say that the password has expired, and you need to create a new password for the user.

That guarantees that in real life, when the user tries to log in for the first time, the admin won't know the final password of the user. We create a user with

```
username alex
```

and

```
password password
```

to test things out. Also, through the webui you can easily enable two-factor authentication (2FA) for the users. You need to log in as a user first and click on actions in the user profile and add otp token. Pick the TOTP and use the freeOTP app in your phone and connect the user with the OTP. Then log in as an admin and enable two factor authentication (password+OTP ) in the user settings. 2FA should run now if you want to use it.

## NEXTCLOUD

Nextcloud only needs one command to set it up in container, which is

```
sudo docker run --name my-nextcloud-container -p 9000:80 nextcloud
```

Now, when you go to **localhost:9000** ( don't go to **192.168.9.10:9000**, it works but it gives annoying warnings all the time until we have that fixed later on ) you can create an admin account for nextcloud, in our case we used admin/test as a username/password.

Then, once in, go to your profile, click **Apps** and **enable LDAP**. Then go to **settings**, pick **LDAP/AD integration** and add the following:

In the server tab put the IP of **172.17.0.1** in the IP box and the port 389 in the next box. Below put uid=admin,cn=users,cn=accounts, dc=final,dc=test and below that put the password Secret123 in the final bottom box put

```
dc=final,dc=test
```

- In the users tab edit the LDAP query to

```
(|(objectclass=*))
```

- In the login attributes edit the LDAP query to

```
(&(|(objectclass=*))(uid=%uid))
```

- In the group tab edit the LDAP query to

```
(|(cn=ipausers))
```

In the advanced options, check that the configuration active box is checked, in directory settings in the base user tree add

```
cn=users,cn=accounts,dc=final,dc=te
```

In the base group tree add

```
cn=groups,cn=accounts,dc=final,dc=test
```

In the group member association pick uniqueMember and in the special attributes type mail in the email field and cn in the user home folder naming rule box. If you test the configuration it should be valid and now theoretically you should be able to log in with alex/password in Nextcloud. Since we need to access Nextcloud from other pcs as well, we need to edit the trusted domains. Find the config.php file ( in our case its in /var/lib/docker/volumes//_data/config) and edit the trusted domains. It should already have by default the **localhost:9000** so add the PRIVATE IP OF YOUR HOST MACHINE, i.e **192.168.9.10** in our case. Now, you can browse from other pcs in the private network and go to **192.168.9.10:9000** and you should be allowed to log in.

## FreeRadius

To install free radius, do this first:

```
apt install freeradius freeradius-ldap freeradius-utils
```

Then edit your /etc/freeradius/3.0/clients.conf file and add

```
client server.final.test {
   ipaddr= 192.168.9.10 secret = somesecret
}

Client dockernet {
   ipaddr = 172.17.0.0/16 Secret = somesecret2
}

Client myaccesspoint1 {
   Ipaddr = 192.168.9.1 Somesecret = somesecret3
 }
```

And save the file. Now go to /etc/freeradius/3.0/certs and run the command make. This will create some snake-oil certificates which shouldn't be used in production but for this demo we are good to go. Normally, we would edit the various files here to create our own CA,certificates and public and private keys. Run this command to enable eap authentication as well

```
ln -s /etc/freeradius/3.0/mods-available/eap /etc/freeradius/3.0/mods-enabled/
```

Freeradius should run and show ready to process requests. Now, we need to add users to authenticate, so whenever we add a user in FreeIPA, we need to add him here as well, so go to the /etc/freeradius/3.0/users file and add in the beginning of the file add

Now, run systemctl stop freeradius.service and rerun it with freeradius -X

Freeradius should run and show ready to process requests. Now, we need to add users to authenticate, so whenever we add a user in FreeIPA, we need to add him here as well, so go to the /etc/freeradius/3.0/users file and add in the beginning of the file add

```
<username> Cleartext-Password := "<the-password>"
```

Then stop the service and rerun it as before. If the router is set up properly and you followed the instructions, you should be able to connect to the wifi with your username and password.

## OpenVPN Access Server

Firstly, lets download the appropriate image. Go to /home/alex and do docker pull linuxserver/openvpn-as. Also, create a directory here with mkdir vpn-data. Moreover, go to the router with dd-wrt and firstly,notice the public IP that it has and also allow port forwarding for UDP port 1194. Now, go back to the host machine and run the command:

```
docker run \
--name=openvpn-as \
--cap-add=NET_ADMIN \
-e PUID=1000 \
-e PGID=1000 \
-e TZ=Europe/London \ ##change that to your area, for me its Europe/Stockholm
-e INTERFACE=eth0 `#optional` \
-p 943:943 \
-p 9443:9443 \
-p 1194:1194/udp \
-v /home/alex/vpn-data:/config \
--restart unless-stopped \
ghcr.io/linuxserver/openvpn-as
```

and now you can go to http://localhost:943/admin and you will be prompted to connect as an admin. The default settings are admin/password so go there. Go to configuration->network settings and in hostname or IP address, add the public IP of the router, pick UDP as a protocol, use port number 1194 then save settings on the bottom of the page and update running router on the top of the page.

Then go to VPN settings and in the routing part, add in the box the address of your private network, i.e 192.168.9.0/24. Save and update again. Go to authentication and pick ldap and then go to the ldap file and add the appropriate configurations just like in nextcloud. Don't add in the first box the uid=admin one but add uid in the box that has prefilled the SambaAccounts. Also in the base dn in the bottom, put cn=accounts,dn=final,dn=test. Save and update.

Now, you can log out as admin, go to http://192.168.9.10 and here you can connect as a user from freeipa. Log in and download the file that it has for you. Now, to connect to the vpn from your phone for example, send this file to your phone ( with a mail for example), download the OpenVPN connect command

and import that file there. Use your password ( and OTP if you have 2FA on) and you should be able to connect to the VPN. With all these done, you can connect from anywhere to the Stockholm branch, connect to nextcloud and share files with anyone who used nextcloud. If you are in the Stockholm branch, you can connect to the Wi-Fi with FreeRadius and then access Nextcloud.

## Internal VPN

In order to be able to connect through the London branch to Stockholm, we need to create an internal vpn from London to Stockholm.

For this, the Stockholm router runs an OpenVPN server and the London router and OpenVPN client. This server and client are included in the DD-WRT firmware.

For the server, we connect to the router GUI and go to **Services->VPN**. Then we enable the OpenVPN server. We configure the VPN server to start when the router is on. And the VPN is run on a tunnel mode to have two separate subnets. We also set all the ciphers' algorithm used for the router. We also add a firewall rule to enable the communication via the tunnel to the LAN network.

For the client side, we connect to the router GUI and go to **Services->VPN**. Then we enable the OpenVPN client, set the IP address to the public IP address of the Stockholm router and all the other parameter equal to what was set on the server side. In the additional configuration field we add the rule ***ADD THE CODE HERE*** to forward the traffic to 192.168.9.X to the correct destination via the tunnel.

Then we configure the certificates for the client and the server. For this we generate a key pair and then the CA from FreeIPA provide the certificates via a CSR. The private key and the certificate of the server (resp. the client) are added to the corresponding field in the server (resp. client) GUI. We also add the certificate of the CA on both the server and the client and finally add a Diffie Hellman Parameters in the dh.pem field.

Step by step configuration of the routers for the VPN can be found here (you need to be logged in to acces the pdf).

Once we do that, the London router is connected securely to the Stockholm router via an OpenVPN tunnel. This means that all the traffic from London that need to go to Stockholm is going through the secure tunnel. Now, if you set it up appropriately, users in London can connect to the London wifi with freeradius authentication from Stockholm!

## INTRUSION DETECTION SYSTEM - SNORT

Intrusion Detection System (IDS) is deployed using the open source IDS software named Snort.

### Docker image

Snort is deployed as a container using Docker. The Docker image is available at emilstahl/snort on Dockerhub. To run the Snort image, execute the following command on Unix:

```
$ docker run -it --rm --net=host emilstahl/snort /bin/bash
```

where –net-host is needed to analyze traffic on the host machines interfaces.

### Snort configuration

Once in the container, the entrypoint is in the ./Snort directory where two directories are present, ./etc and ./rules In ./etc the snort.conf is located where the user can specify the subnet of which to analyze traffic. This is specified at line 45 and is predefined to

```
ipvar HOME_NET 192.168.9.0/24
```

Furthermore, one can specify which IP range the $EXTERNAL_NET shall correspond to, the default is the inverse of the $HOME_NET, meaning all adresses except the $HOME_NET.

**NOTE:** When deploying this image on another network than initially attended, make sure to update $HOME_NET and $EXTERNAL_NET accordingly.

### Applied rules

In ./rules the local.rules file is located which includes site specific rules such as:

- ICMP requests originating from the $EXTERNAL_NET with destination matching the $HOME_NET
- FTP connection attempts
- SSH connections from the $EXTERNAL_NET
- Potential bruteforce attacks due to three failed SSH authentications during the last 60 seconds originating from any IP-adress, including $HOME_NET.

In .snort/rules/standard-rules/, various other rule files are located with the purpose of analyzing situations such as malicious port scannings, ddos attacks, SQL injections, dns lookups, and NTP. All .rules files are included in the ./etc/snort.conf file.

### Test Snort configuration

To test and run the configuration, a .bashrc script is provided in .snort/.bashrc.

The script must be sourced with the command

```
source ./.bashrc
```

Once sourced, the current specification is tested with the `bash $ testsnort` command. The output shows number the of applied rules and the status of current configuration.

### Run Snort

To start the Snort IDS, execute the command `bash $ runsnort`. The IDS can also be started explicitly with the following command:

```
snort -A console -c /root/Building-Networked-Systems-Security-EP2520/Project/snort/etc/snort
```

N O T E: Make sure to specicy the correct network interface to listen on. Either in the alias of .bashrc or in the command above. The default interface is enp1s0. The interface is specified with the -i flag.

As default, Snort writes alerts to the console. To write to log file, execute:

```
snort -A console -c /root/Building-Networked-Systems-Security-EP2520/Project/snort/etc/snort
```

Alternatively, remove the "-A console" from the command.

## Commmon alerts

Below, some common alerts are shown including SSH connection attempts, ICMP requests, Port scannings, FTP connections, and SSH Brute Force Attack.

```
03/04-16:41:28.802112  [**] [1:1000004:1] SSH incoming [**] [Priority: 0] {TCP} 192.168.9.2:
03/04-16:41:28.805635  [**] [1:628:8] SCAN nmap TCP [**] [Classification: Attempted Informat
03/04-16:41:56.717214  [**] [1:368:6] ICMP PING BSDtype [**] [Classification: Misc activity]
03/04-16:41:56.717214  [**] [1:384:5] ICMP PING [**] [Classification: Misc activity] [Priori
03/04-16:41:56.717273  [**] [1:1000001:0] Pinging... [**] [Priority: 0] {ICMP} 192.168.9.26
03/04-16:42:15.831005  [**] [1:453:5] ICMP Timestamp Request [**] [Classification: Misc acti
03/04-16:42:15.851048  [**] [1:620:11] SCAN Proxy Port 8080 attempt [**] [Classification: At
03/04-16:42:18.397314  [**] [1:1000005:4] Potential SSH Brute Force Attack [**] [Classificat
03/04-16:42:18.606497  [**] [1:1421:11] SNMP AgentX/tcp request [**] [Classification: Attemp
03/04-16:42:18.827199  [**] [1:618:10] SCAN Squid Proxy attempt [**] [Classification: Attemp
```