KTH Royal Institute of Technology
Stockholm

School of Electrical Engineering and
Computer Science

Building Networked Systems Security EP2520

System design and implementation details for ACME Network Project

Authors
Thomas Poncet *(trfpo@kth.se)*
Alexios Stournaras *(alexioss@kth.se)*
Emil Ståhl *(emilstah@kth.se)*
Andreas Åström *(aastro@kth.se)*

March 2022

Contents

1 ACME needs and requirements

In this report, we share our system design, analysis, and implementation of a enterprise network infrastructure as requested by ACME Scandinavia, headquartered in Stockholm. ACME wishes to securely extend their headquarters IT environment to their new branch office in London and enhance their employees access to the company IT resources. ACME needs a new network infrastructure including a new secure web-server in the headquarters of Stockholm. Users at each branch should be able to reach resources located at the other network. Moreover, employees need to be able to connect to the corporate Wi-Fi with their laptops and smartphones as well as share files between each other. If employees are not using their cryptographic credentials they need to use their trusted device for Two-Factor Authentication (2FA).

2 System Design

This section describes the proposed implementation and chosen technologies to achieve the needs specified by ACME.

2.1 Implementation Design

The proposed network topology and design for achieving ACME's needs are illustrated in Figure 1. The figure shows the ACME Stockholm headquarter at the top and the ACME London branch to the bottom right. The Stockholm branch runs one server that hosts all of the required services including FreeIPA, Nextcloud, IDS, FreeRadius, and the VPN Server. The two sites are connected through an OpenVPN tunnel. At the bottom left the figure illustrates a client connecting from an offsite location using 2FA and OpenVPN.

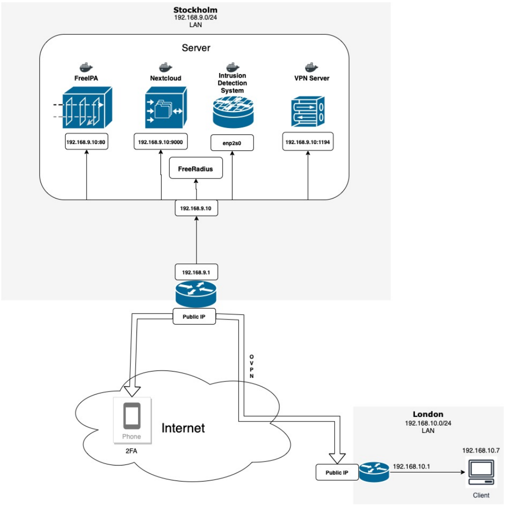


Figure 1: Proposed topology of the new network for ACME

2.2 Security Requirements

This section presents ACME's security requirements and what technologies we have chosen to meet these requirements.

2.2.1 Employee Authentication

For employee authentication we use FreeIPA whih is an open source identity manager. FreeIPA allows for authenticating a user with 2FA and username/password and provides a centralised authority mechanism.

2.2.2 Confidentiality

Communication between the London and Stockholm branch goes through a VPN tunnel. Meaning that the information between the two ACME sites is hidden for third parties. Any outside connections to the Stockholm office are also going through a VPN tunnel, this is useful for employees using their laptop at home or other locations.

2.2.3 Secure connectivity

Accessing the file server located at the Stockholm office requires 2FA. The file server is only accessible for employees with network access, either through Stockholm, London branch, or connecting through the VPN tunnel remotely.

2.2.4 Secure Wireless Access

The tool chosen for authentication and approving network access for users was FreeRadius, an open source RADIUS server. It will handle communication with the routers. It will authorize certain username/password (employees) combinations and approve them for network access. Unfortunately, issues regarding the connection between FreeRadius, FreeIPA and the router emerged. Therefore, we manually update both for any changes and not just FreeIPA. The FreeRadius implementation was done with EAP-TLS using certificates, though self-signed. More details on that in section 3.

2.2.5 Secure File Exchange

NextCloud is the chosen file exchange tool. It connects to the FreeIPA database and only allows users with permission to access it. Users can easily exchange and share files using Nextcloud's capabilities.

2.2.6 Other Security

To be alerted about potential ongoing attacks against the network infrastructure, we are running an Intrusion Detection System (IDS) using the open source IDS software named Snort. The IDS software runs containerized on a server located in the Stockholm branch. The IDS is configured to prioritize alerts regarding outside threats, meaning traffic originating from outside the ACME networks that has a destination residing inside the ACME networks. Regarding inside threats Snort is configured to alert system admins regarding malicious activity such as multiple failed SSH authentications.

3 Discussion

In this section, we discuss our set-up and reflect on how the requirements, functionality, and the security of our solutions are satisfied considering specific use cases.

3.1 Routers

The routers have a simple DD-WRT set up running. After we configured the routers to run DD-WRT properly and give out IPs from the networks 192.168.9.0/24 and 192.168.10.0/24 for Stockholm and London respectively, we created an internal VPN connection between them. This was achieved by creating a new certificate authority, certificates and public and private keys from them with OpenSSL. Since this connection is always open and doesn't have anything to do with the users, we decided to not connect them with FreeIPA and have them set up like that independently. Moreover, we connected the routers with FreeRadius (discussed below), by making them clients of FreeRadius. As such, they can allow internet access via Wi-Fi with EAP-TLS encryption.

3.2 System cooperation for achieving security requirements

Our implementation consists of two routers and one host machine acting as a server for the docker containers running our different services. The two routers, one for each branch, hosts two interfaces, as shown in Figure 1. One of the interfaces for each branch is responsible for communication between the branch and the outside world, while the other is dedicated to handle communication with the respective interface located at the other branch. For the purposes of the demonstration, we haven not created a VPN tunnel from the Internet to London, but by following the same steps as described in the appendix, it is trivial to do so. Figure 1 demonstrates the proper set up, but we will demonstrate only the Stockholm branch to be more concise. The tools that are required for the routers is OpenVPN and DD-WRT. OpenVPN is used to ensure encrypted file exchanges and communication between the branch and the headquarters gateways. We create one VPN connection for London and Stockholm, where the Stockholm router will act as an OpenVPN server, and one VPN connection from an off-site locations to the Stockholm branch, with the router-gateway of Stockholm acting as the server. Thus, we need two public IPs for this demonstration.

DD-WRT allows for greater router configurability and can also act as a firewall. To be more precise, as the reviews suggested, we have firewalls in each of the branches that prevent access to the internal networks, unless the traffic comes from the VPNs that we deploy.

3.2.1 FreeIPA

The host machine is responsible for running an Authentication Server (AS), File Server (FS), IDS and an OpenVPN Access Server on it as containers, FreeRadius runs normally without a container. To act as an AS, the host machine runs FreeIPA, an open source identity management system. FreeIPA takes advantage of Kerberos, LDAP, bind and Dogcert, which is responsible for handling CAs and certificates, and utilises them to become a centralised authority. When a new user is created, it is added to the LDAP and in order to access LDAP safely FreeIPA uses its CA and its certificates. As such, whenever we add another host to the "FreeIPA network" and in its kerberos domain, they get a certificate from that authority that allows them to safely access LDAP and verify the user that tries to log in to their services. Moreover, we do not have to create a user in every different service that we use (i.e Nextcloud, Openvpn access server etc) but we only use the FreeIPA users, which we can easily add and manage either for the WEBUI that it offers or from the command line, and we connect the services with it via LDAP. This allows each employee to have their identity verified by digital certificates, published by a trusted CA and also allows for Two-factor authentication (2FA) for employees, if we choose to. In our case, we do not really add hosts in the "FreeIPA network" because everything runs in containers on the same host machine. But through the docker container networking, the same principles apply and we can authenticate everyone through FreeIPA.

3.2.2 Nextcloud and FreeRadius

As for Nextcloud, we run a simple version of it in a container, and by integrating it with FreeIPA through LDAP, FreeIPA users can share their files through Nextcloud with one another. In theory, the same procedure is followed by connecting FreeRadius with FreeIPA, and having the router, connected via ethernet with the host machine, to verify users trying to access the Wi-Fi with Freeradius. However, while we managed to connect Freeradius with FreeIPA, the configurations of the router do not share the same encryption methods as FreeIPA's LDAP, so FreeRadius is not able to verify the users after all. As such, whenever we add a user in FreeIPA, we manually add them in the users file of FreeRadius as well, and then the router can properly verify these users by using WPA2 and EAP with TLS encryption and sharing a key with FreeRadius.

3.2.3 OpenVPN

Lastly, the host also runs an OpenVPN access server, to allow connections from the outside world to the private network. We realised that, although connecting the two branches with OpenVPN through the routers was running smoothly, using DDWRT and OpenVPN for multiple connections from outside was not optimal. As such, we handle the outside connections with the OpenVPN Access Server (OpenVPN-AS) running in a container on the host. We only need to allow port forwarding in our router to let that happen. Afterwards, we again connect OPenVPN-AS with FreeIPA through LDAP. When a new user is hired, he will also log in to OpenVPN as a client and download a file that allows him to connect through any device, like a mobile phone, that has an OpenVPN client application installed, with his username and password.

3.2.4 Intrusion Detection System

The requirements regarding intrusion detection is clearly satisfied thanks to how we have configured the IDS to function. In general, we consider the ACME LAN's to be relatively safe, meaning that any traffic flowing inside the ACME infrastructure is seen as legitimate. We consider types of attackers differently depending on their likely-hood. Since ACME has dedicated offices extensive security to enter the office buildings we consider situations where the attackers resides from within the ACME network to be relatively unlikely. On the contrary, we argue that attacks coming from outside ACME targeting open ports and public IPs to be considerably higher. Due to this, we have deployed tailored local IDS rules to detect for example ICMP request that has a source originating from outside ACME but a destination on our local LANs. However, we decided to monitor some traffic independently depending on where it comes from. As an example, the IDS catch situations involving a machine that is adversarial and keeps sending network discovery requests and probing the SSH connection of the machines for which it gets the IPs. This particular attack can be simulated with a simple nmap port scanning command and our IDS is proven to successfully detect these types of attacks. Other situations our IDS is configured to detect are attacks regarding DDOS, NTP, an attack-responses. Lastly, we configured the IDS to alert about Log4j-attacks. Since we run all of our services on a single server, we decided to not configure port mirroring on the router for LAN-wide monitoring of attacks. However, this is not needed since all important services are running on containerized on the server which is analyzed by the IDS. However, it would be really easy to extend the infrastructure with port mirroring. The configuration of Snort would not need to updated since it already monitors the Stockholm network. The only thing to configure is the port mirroring itself, that is forward all traffic from the Stockholm gateway to the server where Snort is running. We consider this network to be just as safe with or without port mirroring, however if the network would to be extended with multiple servers then port mirroring would increase the security. Furthermore, the London branch currently have not an IDS either due to the same reasons as mentioned above, that is since the London branch has no services running in the network. However, an IDS could very easily be added to the London branch simply by doing a docker pull and run as described in the IDS section of the Appendix.

3.2.5 Some final details

It is important to note that due to the fact that we configured FreeRadius as described above, we decided to use 2FA everywhere, meaning both when someone connects with a VPN from outside as well as when we are inside the network and want to access a service like Nextcloud. Moreover, although normally we would have another similar set up in the London branch, for the purposes of this report we consider that users in London will connect to their Wi-Fi with authentication from FreeRadius in Stockholm, which is perfectly possible since we configured the internal VPN and we added the router of London as a client of FreeRadius.

Appendices

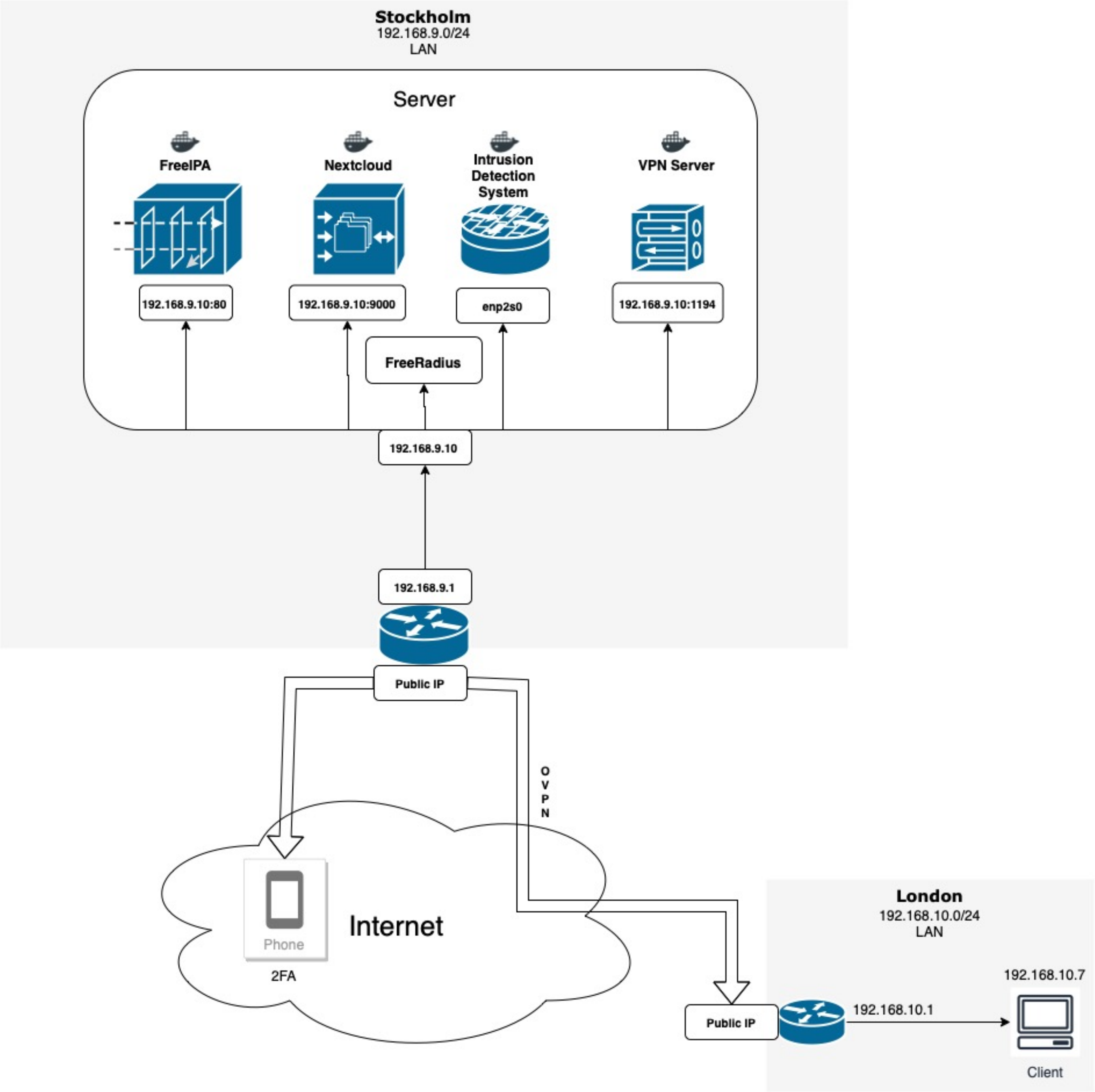
Appendix A

README - How to configure the network

A detailed description of how to set up the network is available at GitHub.

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.

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/keyrings/docker-archive-keyring.gpg

$ 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 tee /etc/apt/sources.list.d/docker.list > /dev/null

$ 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-$(uname -s)-$(uname -m)" -o /usr/local/bin/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 443:443 -p 389:389 -p 636:636 -p 88:88 -p 464:464 -p 88:88/udp -p 464:464/udp -p
123:123/udp -v /sys/fs/cgroup:/fs/cgroup:ro -v /var/lib/ipa-data:/data:Z -e
PASSWORD=Secret123 -sysctl net.ipv6.conf.all.disable_ipv6=0 freeipa/freeipa-
server:centros-8-4.8.7 ipa-server-install -U -r FINAL.TEST --no-ntp
```

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

```
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.conf -i enp2s0
```

NOTE:
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.conf -i enp2s0 >> /var/log/snort/snort.log
```

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

Common 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}
192.168.9.2:55338 -> 192.168.9.26:22
03/04-16:41:28.805635  [**] [1:628:8] SCAN nmap TCP [**] [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 192.168.9.2:55338 -> 192.168.9.26:22
03/04-16:41:56.717214  [**] [1:368:6] ICMP PING BSDtype [**] [Classification: Misc
activity] [Priority: 3] {ICMP} 192.168.9.2 -> 192.168.9.26
03/04-16:41:56.717214  [**] [1:384:5] ICMP PING [**] [Classification: Misc activity]
[Priority: 3] {ICMP} 192.168.9.2 -> 192.168.9.26
03/04-16:41:56.717273  [**] [1:1000001:0] Pinging... [**] [Priority: 0] {ICMP}
192.168.9.26 -> 192.168.9.2
03/04-16:42:15.831005  [**] [1:453:5] ICMP Timestamp Request [**] [Classification:
Misc activity] [Priority: 3] {ICMP} 192.168.9.2 -> 192.168.9.26
03/04-16:42:15.851048  [**] [1:620:11] SCAN Proxy Port 8080 attempt [**]
[Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.9.2:55340 ->
03/04-16:42:16.960237  [**] [1:620:11] SCAN Proxy Port 8080 attempt [**]
[Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.9.2:55362 ->
03/04-16:42:17.064509  [**] [1:1000002:1] FTP connection attempt [**] [Priority: 0]
{TCP} 192.168.9.2:55368 -> 192.168.9.26:21
03/04-16:42:18.397314  [**] [1:1000005:4] Potential SSH Brute Force Attack [**]
[Classification: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.9.2:55860 ->
192.168.9.26:22
03/04-16:42:18.606497  [**] [1:1421:11] SNMP AgentX/tcp request [**] [Classification:
Attempted Information Leak] [Priority: 2] {TCP} 192.168.9.2:56020 -> 192.168.9.26:705
03/04-16:42:18.827199  [**] [1:618:10] SCAN Squid Proxy attempt [**] [Classification:
Attempted Information Leak] [Priority: 2] {TCP} 192.168.9.2:56207 -> 192.168.9.26:3128
```