Building a Networked Systems Security

Final Report

Group 6 - Sniff_my_packets_420

Christoforos Seas, Ferran Solanes, Eduardo Santos, Gabriel Fetu

Table of Contents:

Final Network Topology

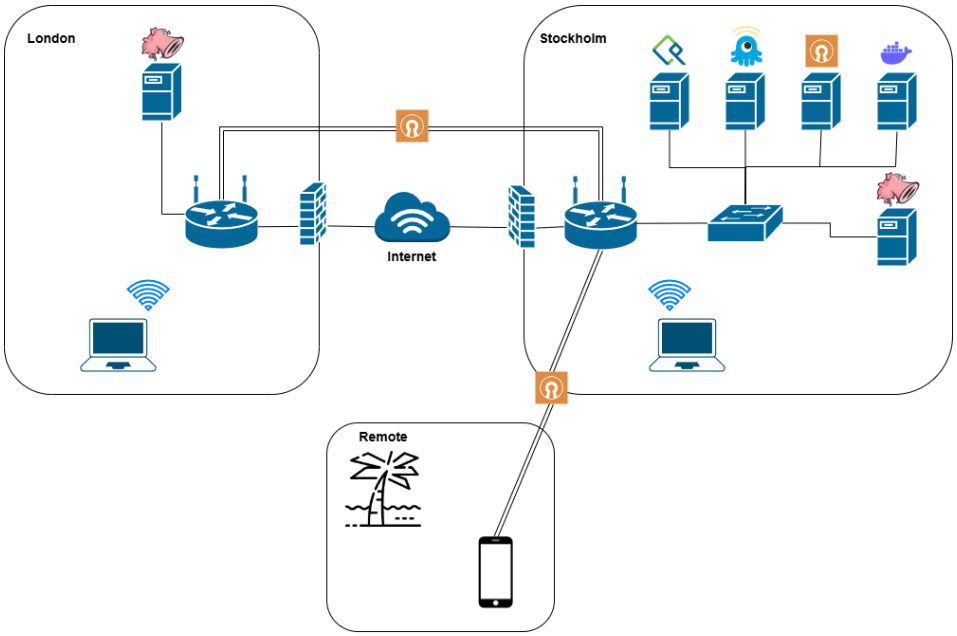

Figure 1: ACME network topology

Local Networks connection

We have connected the two offices from Stockholm and London using OpenVPN, an SSL VPN. We have used OpenVPN because our router's firmware supports it. OpenVPN establishes a tunnel between our two routers in Stockholm and London branches where London's one is seen as a client of Stockholm's one. This tunnel uses SSL/TLS for authenticating the two sides. It means that each side presents its own certificate and private key. Then, the traffic going through the tunnel is encrypted and authenticated. To configure OpenVPN on our routers, we developed one script per branch. They set up the tunnel connection by uploading the keys and certificates to the routers and configuring OpenVPN parameters.

For remote network access, we also used OpenVPN, which has a dedicated server in our Stockholm network. It is in charge of providing connection to the company network and authenticating the users based on the certificates present in the client configuration files. These files have to be handled carefully. User's certificates can be created running a script on the machine running the VPN server called *create_client.sh*. These certificates can also be revoked and a CRL can be updated with the script *revoke_cert.sh*. Then, once connected to our VPN server, a remote user may be distinguished from a user in the office and therefore undergo more authentication steps to connect to ACME's services (We enabled 2-Factor Authentication for these users). We decided on running the VPN server in a docker container and not on the router, because we were scared it would overwhelm it in a realistic scenario where many users have open connections at the same time. The router simply forwards UDP traffic on port 1194 to the VPN server (see Appendix A).

Wireless Access

Both routers (Stockholm and London) use WPA2-Enterprise authentication, and are connected to a freeRadius server which provides authentication. This allows us to authenticate users with our LDAP directory, which is running in FreeIPA. This is convenient for two reasons. First we have a central user management tool, which allows easy and fast addition and revocation of user accounts. And second, we authenticate our users, and consequently, only users from our company have access to the LAN. Furthermore, the authentication is done with Certificates, so we must have previously issued a certificate for each Wi-Fi user, and this gives us even more control. To create the certificates necessary, we have developed a script that has to be run in the FreeIPA server. The script is called *create_certs.sh* and can be found on our github repository, under the *certificates* directory. The script takes the IPA admin user password, the principal for which we need the certificate and the service (only when creating a certificate for a service and not a user). To sync the CRL from FreeIPA with FreeRadius, we have created a script (*freeradius-CRL-Update/freeipa-crl-update.sh*) which fetches the CRL from FreeIPA and creates the correct file format for

Freeradius to use. Then, on the *mods-enabled/eap* module configuration, we set the value *check_crl* to *yes* and the *ca_file* variable to point towards our CRL file (which has the CA certificate and the actual CRL in PEM format). The script also reloads the freeradius service, since that is needed for FreeRadius to use the correct CRL.

Services

Secure Communication

We run most of our services with docker, and we have deployed a reverse proxy in front of those docker services. This reverse proxy is in charge of handling all the TLS communication. FreeIPA issues the certificates used by the reverse proxy, and all the communications with services behind the reverse proxy are protected using TLS. See the *Authentication* Section for further information on how we use Traefik as a reverse proxy to enforce TLS.

Secure Web Server

A secure web server has been implemented and is running inside a docker container behind our reverse proxy. Python is used to run the web server. Only authenticated users can access the website at https://webserver.acme.local.

Secure File Sharing

We have deployed Nextcloud, using Docker. This allows our service to be lightweight and be portable. We have created a new group in our LDAP directory, called nextcloud-users and any user in that group has access to nextcloud. This has been done to avoid the services users access to nextcloud, since these are not human users. Furthermore, the accounts of nextcloud are synchronized with the LDAP directory, thus a user gets an account for nextcloud automatically.

Authentication

Comentado [1]: limit for this report is 5 pages (+ 3 pages Appendix for screens / technical details). Don't detail everything

Comentado [2]: Oof okay

Comentado [3]: Now it's 3 pages with the screenshots

For ACME's networked systems security project, we implemented **Authelia** (https://www.authelia.com), an open-source authentication server from GitHub (https://github.com/authelia/authelia). Authelia provides Single Sign-On (SSO) and multi-factor authentication (MFA), working with reverse proxies like Traefik to secure access to ACME's services.

Operation/Flow

Authelia uses a web portal for centralized authentication. When an employee accesses services such as **Nextcloud** (file-sharing), **Traefik** (reverse proxy), the web server, or others, they are redirected to Authelia's login page. After entering their credentials, SSO grants seamless access to all services. For enhanced security, Authelia enforces **Two-Factor Authentication (2FA)** when required, prompting employees if needed. An example of the flow can be seen in Figures 3 and 4 in Appendix B.

Implementation Details

We configured Authelia to integrate with Traefik, which acts as the entry point for all incoming traffic. Traefik forwards authentication requests to Authelia, which then validates credentials and enforces 2FA when required. This setup provides a robust, scalable authentication layer that can adapt to ACME's future service expansions while maintaining a user-friendly experience for employees.To achieve this integration, we relied heavily on the official resources provided by Authelia. The detailed documentation on their website (https://www.authelia.com) offered step-by-step guidance on configuration, including how to pair Authelia with a reverse proxy like Traefik. Additionally, we consulted the Authelia GitHub repository (https://github.com/authelia/authelia), which provided access to example configurations, issue trackers, and community discussions that helped us troubleshoot specific challenges. Beyond these written resources, we found valuable assistance from a Persian YouTube creator whose tutorial (https://www.youtube.com/watch?v=aimQe3vox6U) clarified the practical steps for setting up Authelia and Traefik in a real-world environment. This combination of official documentation and community-driven support enabled us to implement a secure and efficient authentication system tailored to ACME's needs.

Access Control Rules

Authelia's access control system allows us to define granular rules to secure ACME's services based on network locations, user groups, and authentication policies. Configuring these rules is straightforward and highly customizable, enabling us to tailor

security requirements to the company's operational needs. We began by defining three network groups corresponding to ACME's infrastructure: Stockholm, London, and VPN (outside). For each service, we then specified the authentication policy. Options include one-factor, two-factor, or bypass, long with the permitted networks and user groups allowed to access the service.
To implement this, we used Authelia's configuration file to establish a default "deny" policy, ensuring that access is blocked unless explicitly allowed by a rule. We defined the networks as follows:

- **Stockholm**: Representing devices on the subnet 192.168.10.0/24.
- **London**: Covering devices on the subnet 192.168.11.0/24.
- **VPN**: Including remote connections via the VPN server, identified by the IP 192.168.10.85/32.

Next, we crafted specific rules to enforce security policies for ACME's services. For example, the Traefik dashboard (traefik.acme.local) requires one-factor authentication and is accessible only to users in the admins group, regardless of whether they connect from Stockholm, London, or the VPN. See Appendix C for the actual implementation.

Two-Factor Authentication

To implement 2FA, we developed a script *setup_totp_for_users.sh* with the username as an argument, that generates a QR code displayed to the employee during setup. Employees scan this QR code using an authenticator app, such as Google Authenticator, Microsoft Authenticator, or both, to register their device and generate time-based one-time passwords (TOTPs). This process adds an additional layer of security by requiring a second factor beyond the password, which significantly reduces the risk of unauthorized access.

IDS & SIEM

As IDS our group has decided to use Snort3. In order to check on the traffic running on both branches, 2 instances of Snort3 will be running in parallel. Hence, the following explanation applied to both branches. Since running Snort on the routers could compromise its resources, we have implemented it on a virtual machine that is part of the LAN. This presented the first challenge, since not having the IDS in-line prevents it from checking the branch traffic. To solve this, port mirroring via **iptables** (-TEE) and **ebtables** has been

implemented, see Appendix D for the specific rules. The latest one main target is to redirect the layer-2 LAN traffic to the Snort-VM. With the port mirroring implemented, Snort is able to get a copy of every packet going through the network, from both external and internal sources to the LAN.

The rules set used by Snort is based on the public set **Light SPD**, available at Snort.org. Via scripting this ruleset has been filtered to only apply rules whose objective is the DNS and DoS-attacks detection. Some "home-made" rules have been added for testing purposes.

On top of Snort we have implemented a SIEM system. The tool we have used for it is **Splunk**. The data is transferred via JSON file export. Within Splunk, alarms have been configured to be triggered under certain circumstances.

Appendix A

```
iptables -I FORWARD -p udp -d {{ vpn_server_ip }} --dport 1194 -j ACCEPT
iptables -t nat -A PREROUTING -p udp --dport 1194 -j DNAT --to-destination {{ vpn_server_ip }}:1194
```

Figure 2: iptables rules for VPN port forwarding

Appendix B

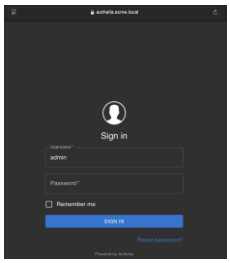

Figure 3: Authelia authentication form.

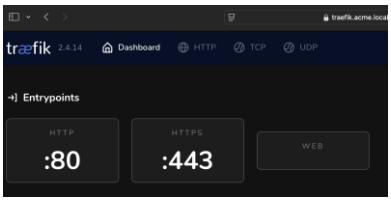

Figure 4: Traefik dashboard.

Figure 5: Example of QR code for 2FA

Comentado [4]: When is this referenced? Just to have it ordered correctly

Comentado [5]: very first part VPN, should be fig 1

Comentado [6]: 👍

Appendix C

```
access_control:
  default_policy: deny
  networks:
    - name: 'Stockholm'
      networks:
        - '192.168.10.0/24'  # IPs in the 192.168.10.0/24 (Stockholm) subnet
    - name: 'London'
      networks:
        - '192.168.11.0/24'  # IPs in the 192.168.11.0/24 (London) subnet
    - name: 'VPN'
      networks:
        - '192.168.10.85/32'  # IPs in the VPN subnet (from VPN server)
  rules:
    - domain: traefik.acme.local
      policy: one_factor
      subject: "group:admins"
      networks:
        - Stockholm
        - London
        - VPN

    # Require two-factor for webserver
    - domain: webserver.acme.local
      policy: two_factor
      networks:
        - Stockholm
        - London
```

Figure 6: Access control for authelia

Appendix D

```
iptables -I PREROUTING -t mangle -j TEE --gateway 192.168.10.31
iptables -I PREROUTING -t mangle -j MARK --set-mark 1
iptables -I POSTROUTING -t mangle -m mark ! --mark 1 -j TEE --gateway 192.168.10.31
```

Figure 7: Iptables rules for port mirroring

```
ebtables -t broute -A BROUTING -p IPv4 --ip-src 192.168.10.1 -j ACCEPT
ebtables -t broute -A BROUTING -p IPv4 --ip-dst 192.168.10.1 -j ACCEPT
ebtables -t broute -A BROUTING -p IPv4 --ip-src 192.168.10.0/24 --ip-dst
192.168.10.0/24 -j redirect --redirect-target DROP
```

Figure 8: ebtables rules for port mirroring

README for Non-expert Users

User Management

When new employees enter ACME, you need to create users for them. To do so, you can access the User Management service in https://freeipa.acme.local, access with the administrator user, and create the new user in the Identity menu. When the account is created, you should also (if needed) add the user in the nextcloud-group, which you can do by clicking the user, and in the groups tab, add him to the nextcloud-users group. To access the Wireless access points, the user will need a PFX certificate. To do so, you can access the FreeIPA server with SSH (the credentials are in the Keeweb in Nextcloud) and run the script *certificates/create_certs.sh* in the github repository.

The same can be said when an employee needs to be deleted. To delete a user, you have to delete the LDAP user, and also their certificates. The user will lose all access to our services, since all need authentication, either with certificates or with LDAP credentials.

Services Management

When you want to add a new service, you will need a set of steps. You will need to create the host - if it doesn't yet exist. You can check the available hosts in the FreeIPA dashboard, in *Identity -> Hosts*. Once we have the host, we must create a DNS entry in the *Network Services -> DNS*, here we have defined the domain ***acme.local*** which can be used to deploy internal services, only accessible from the LAN. Then, you can create the service in the *Identity -> Services* tab of FreeIPA, where you can specify the protocol/service given. This service will be the **Principal**, which is needed if you want to issue certificates for the service (for TLS for example). To create the certificates, you can again use the *certificates/create_certs.sh* in the Github repository. Although you probably won't need the PFX certificate.

Setup VPN Tunnel Between a New Branch and Stockholm HeadQuarters

First of all, set Stockholm's branch as the server by running *deploy_stockholm_ovpn.sh*. Then, to establish a tunnel between a new company branch and Stockholm headquarters, simply run the script deploy_london_ovpn.sh (which is not specific to London's branch, except for its name) on a machine connected to the LAN of the new branch.

Access to the VPN

In order to connect to the company network remotely, an employee must have an OpenVPN client configuration file and the OpenVPN client application installed on its device. Then, using this file, the employee may connect with any device remotely. It will still have to authenticate as any other employee working at either branch before accessing services like the web server.

To create a new VPN user, run the script *create_client.sh* on the VPN server. To revoke the access of a user, run *revoke_cert.sh*.

Two-factor Authentication

Employees will use mobile applications such as Microsoft authenticator or Google authenticator to enable multi-factor authentication. To link the app to our system, you need to run the script *setup_totp_for_users.sh* with an existing username. Then the employee will scan the QR code generated by the script. It will provide TOTP whenever 2FA is required to log in to the services.