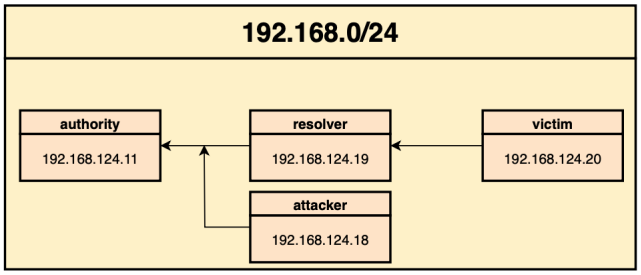
Network Security

Assignment: Application Layer

DNS Hijacking (50 points)

This assignment concerns an application server that regularly checks for updates. However, the update server is not properly secured, so if we can manage to redirect the update to our own malicious update server, we can exploit this process. In this exercise, we will target the DNS infrastructure to accomplish this.

The vulnerable DNS resolver will be running on 192.168.124.19. It periodically resolves the domain name `update-server.updateserver.corp` from its authoritative name-server at 192.168.124.11. The attacker has a man-in-the-middle position between the DNS resolver and the authoritative resolver and can observe all traffic passing through that link.



The responses from the authoritative name-server are served with a TTL of 15 seconds, and are therefore also cached for 15 seconds on the local name-server. In other words, every 15 seconds, the local server will forward the request to the authoritative DNS server.

Set up the environment using the commands below.

```
docker compose build
docker compose up -d
```

The *attacker* container has a mount configured, such that you can edit files directly on your computer in an editor of your choice and then access them within the container: files in the `solution` directory are mirrored in the container under `/solution`. To connect to the attacker container, execute the command below.

```
docker exec -ti assignment_5_dns_attacker bash
```

After finishing the assignment, you can destroy the environment using the command below.

```
docker compose down
```

Goal

You should produce a Python3 script that poisons the DNS cache of the local DNS server, by intercepting the DNS query and spoofing a response. After successfully poisoning the DNS cache, the script should exit. The cache should remain poisoned for a long period after the attack ends.

The script should accept 3 arguments: the DNS server we are targeting, the domain name of the DNS request, and the IP address we want the answer to be.

For instance:

```
python3 dns_hijack.py 192.168.124.19 update-server.updateserver.corp 1.2.3.4
```

You can monitor the application by following the logs of the victim:

```
docker logs --follow assignment_5_dns_victim
```

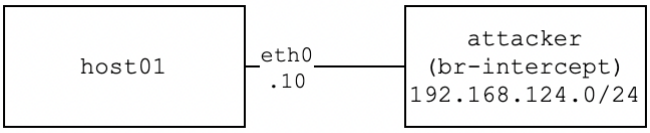
Hint: You can time your DNS request correctly by making a DNS request to the resolver and inspecting the TTL field of the response, or you can wait for the TTL to expire and intercept the query to the authoritative DNS resolver.

Submission Instructions

Your script should run on the *attacker* container. You may not use external libraries or tools different from those installed on the provided *attacker* container.

TLS Interception [50 points]

In this assignment, the attacker has established a MITM position between host01 and the Internet. Moreover, the attacker has installed a malicious root certificate on the host.



When host01 requests a web page, the request and response traffic is forwarded through the attacker. We're going to use this presence to intercept encrypted HTTPS traffic. Such a setup is sometimes also used in a benign way to set up a transparent proxy for detecting malicious traffic. Set up the environment using the commands below.

```
docker compose build
docker compose up -d
```

To connect to the attacker container, execute the command below.

```
docker exec -ti assignment_5_tls_attacker bash
```

To launch a connection from the client to a website using HTTPS, you can use `curl`. For example, to connect to Google, you can use the command below. During debugging, you can use curl's `-k` flag to ignore SSL warnings and the `-v` flag to show additional information.

```
docker exec assignment_5_tls_host01 curl https://google.com\?q\=verysecret
```

The root certificate allows the attacker to generate and sign certificates that are trusted by the client. This certificate and its corresponding key are available on the attacker machine under `/certificate`. Files in the `solution` directory are mirrored under `/solution`.

After finishing the assignment, you can destroy the environment using the command below.

```
docker compose down
```

Goal

Write a Python3 script which takes as an argument the port number on which you want to activate your program. Within the context of the assignment, this should be 8443.

The script should run continuously (until Ctrl+C is pressed), intercept all HTTPS connections and output the requests and responses. For every domain, it should generate a new valid certificate using the provided root certificate, such that it's trusted by the victim host. Example output is shown below.

```
$ python3 tls_intercept.py 8443

Certificate request self-signature ok
subject=C = NL, CN = google.com

GET /?q=verysecret HTTP/1.1
Host: google.com
User-Agent: curl/7.88.1
Accept: */*


HTTP/1.1 301 Moved Permanently
Location: https://www.google.com/?q=verysecret
Content-Type: text/html; charset=UTF-8
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src
'nonce-DpOLm4pUvTt8C6QRVsU8Mg' 'strict-dynamic' 'report-sample' 'unsafe-eval'
'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp
Date: Fri, 21 May 2025 08:10:23 GMT
Expires: Sun, 20 Jun 2025 08:10:23 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 233
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="https://www.google.com/?q=verysecret">here</A>.
</BODY></HTML>
```

Hint: In this task, you aren't expected to use Scapy. You may find `openssl` useful (e.g., by executing commands via `os.system` or using the provided `pyopenssl` Python library) as well as the built-in Python `ssl` module. To generate certificates on demand, you can set up an SNI callback.

Submission Instructions

Your script should run on the *attacker* container. You may not use external libraries or tools different from those installed on the provided *attacker* container.