Network Security — Exercise 4: Transport Layer Security

Noah Link, Jan Pfeifer, Julian Weske

June 6, 2024

1 Preliminaries

We used Ubuntu and skipped the VM install.

2 TLS Basics (Time spent: 2h)

The full code can be found in our *solution.zip* under *part2*.

2.1 TLS Client

1. What is the cipher used between the client and the server?
   Between the server and the client, *AES* cipher is used.

```
openssl s_client -connect example.com:443
...
TLS_AES_256_GCM_SHA384
...
```

2. Print out the server certificate in the program:

```
HTTP/1.1 200 OK
Age: 401790
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 04 Jun 2024 10:18:12 GMT
Etag: "3147526947+gzip+ident"
Expires: Tue, 11 Jun 2024 10:18:12 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECAcc (nyd/D16C)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256
```

3. Explain the purpose of /etc/ssl/certs:
   The folder */etc/ssl/certs* is used to store certificates on Linux, enabling users to securely communicate by verifying the authenticity of SSL connections when communicating with servers. Specifically, it includes PEM files of CA (Certificate Authority) certificates.

We found out that *DigiCert_Global_Root_G2.pem* is the certificate used and created a link for it. It is listed in the browser, when visiting *www.example.com*

2.2 Certificate Authority

We followed the given steps and were able to generate a self-signed certificate. A shortened version of the certificates are here:

```
    Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3f:63:fa:ef:b1:28:1e:9d:02:88:f2:c1:03:a6:41:0f:06:7e:c2:3a
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = DE, ST = Hamburg, L = Hamburg, O = UHH, OU = Netsec, CN = PWL, emailAddress = ". "
        Validity
            Not Before: Jun 4 13:08:11 2024 GMT
            Not After : Jul 4 13:08:11 2024 GMT
        Subject: C = DE, ST = Hamburg, L = Hamburg, O = UHH, OU = Netsec, CN = PWL, emailAddress = ". "
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                21:8C:00:1C:4A:DD:7B:EA:69:59:1C:16:51:9A:21:44:D6:64:72:44
            X509v3 Authority Key Identifier:
                21:8C:00:1C:4A:DD:7B:EA:69:59:1C:16:51:9A:21:44:D6:64:72:44
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        ...
```

The content of the *server.csr file*:

```
openssl req -noout -text -in 'server.csr'

Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = DE, ST = Hamburg, L = Hamburg, O = example, OU = Betriebsrat, CN = example.com
        Subject Public Key Info:
```

```
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                ...
            Exponent: 65537 (0x10001)
    Attributes:
        unstructuredName :example2
        challengePassword :challenge
        Requested Extensions:
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
    ...
```

And the new *server.crt*:

```
openssl x509 -noout -text -in 'server.crt'
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = DE, ST = Hamburg, L = Hamburg, O = UHH, OU = Netsec, CN = PWL, emailAddress = ". "
        Validity
            Not Before: Jun 4 13:38:44 2024 GMT
            Not After : Jun 4 13:38:44 2025 GMT
        Subject: C = DE, ST = Hamburg, O = UHH, OU = Betriebsrat, CN = example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                04:0B:C4:C5:0B:C6:7A:63:F3:6A:A2:5E:0F:91:99:AF:CC:42:E6:2A
            X509v3 Authority Key Identifier:
                21:8C:00:1C:4A:DD:7B:EA:69:59:1C:16:51:9A:21:44:D6:64:72:44
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        ...
```

We hope that was all you want to see. Our folder-structure can be found too in *solution.zip* in folder *part2* for you to check. This contains all certificates we generated.

2.3 TLS Server

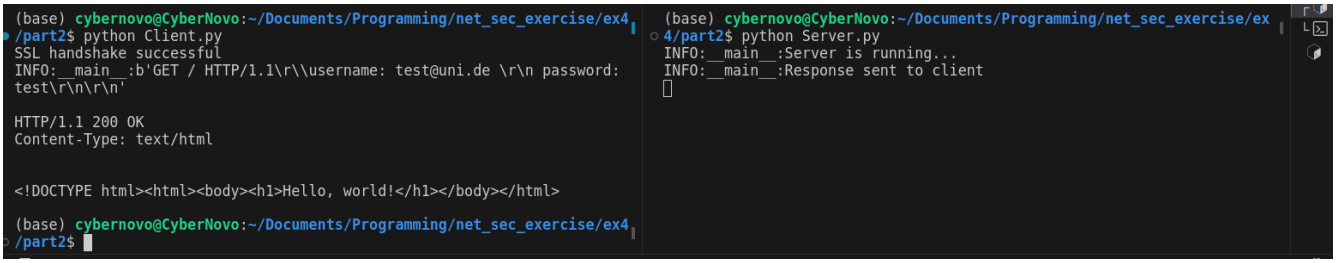
After managing the certificates, we get the following if we use *./certs/ca.crt"*:

```
python Client_template.py
SSL handshake successful

HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html><html><body><h1>Hello, world!</h1></body></html>
```

But if we switch to */etc/ssl/certs/ca-certificates.crt* we get:

```
python Client_template.py
Traceback (most recent call last):
  File "/home/cybernovo/Documents/Programming/net_sec_exercise/ex4/Client_template.py", line 20, in
      <module>
    ssock = context.wrap_socket(sock, server_hostname=hostname)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/cybernovo/anaconda3/lib/python3.11/ssl.py", line 517, in wrap_socket
    return self.sslsocket_class._create(
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/cybernovo/anaconda3/lib/python3.11/ssl.py", line 1108, in _create
    self.do_handshake()
  File "/home/cybernovo/anaconda3/lib/python3.11/ssl.py", line 1379, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get
     local issuer certificate (_ssl.c:1006)
```

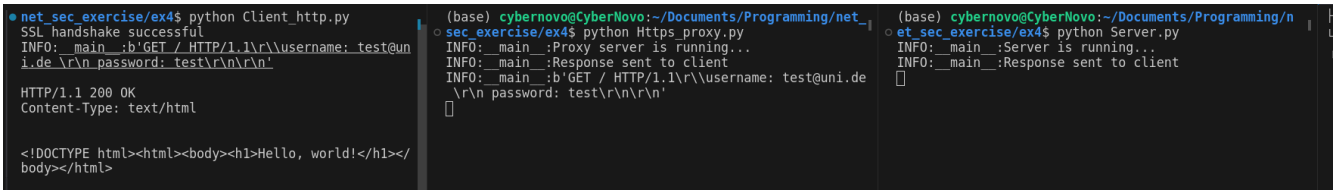
This is expected and happens because the CA certificate used *./certs/ca.crt* is not included in the system's list of trusted CA certificates */etc/ssl/certs/ca-certificates.crt*. Using the right certificates, the Client-Server-Setup works as expected:



The full code for this is found in *part2* in Client.py and Server.py.

3 TLS Security: A Simple HTTPS Proxy (6h because of massive troubleshooting)

We implemented the proxy, which can be found under *part3/test_local/Proxy.py*. It functions as a middleman between Client and Server and does its job if run locally:



The figure above shows that it is possible for the proxy to read credentials in the local setup.

To test it on a real website, we chose *www.mindfactory.de*. We regenerated the certificates for CA and proxy using *CN="www.mindfactory.de*.

```
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:HH
Locality Name (eg, city) []:HH
Organization Name (eg, company) [Internet Widgits Pty Ltd]:evil
Organizational Unit Name (eg, section) []:ca
Common Name (e.g. server FQDN or YOUR name) []:www.mindfactory.de


Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:HH
Locality Name (eg, city) []:HH
Organization Name (eg, company) [Internet Widgits Pty Ltd]:evil
Organizational Unit Name (eg, section) []:proxy
Common Name (e.g. server FQDN or YOUR name) []:www.mindfactory.de
Email Address []:.


Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:HH
Locality Name (eg, city) []:HH
Organization Name (eg, company) [Internet Widgits Pty Ltd]:evil
Organizational Unit Name (eg, section) []:mindfactory
Common Name (e.g. server FQDN or YOUR name) []:www.mindfactory.de
Email Address []:.
```

And generated a new certificate for mindfactory, signed by our CA.

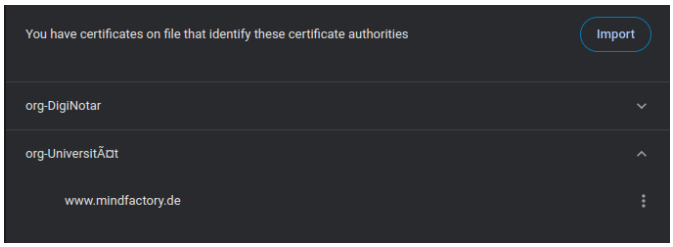
```
openssl genrsa -out mindfactory.de.key 2048
openssl req -new -key mindfactory.de.key -out mindfactory.de.csr
openssl x509 -req -in mindfactory.de.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
    mindfactory.de.crt -days 365
```

```
openssl verify -CAfile ca.crt mindfactory.de.crt
```

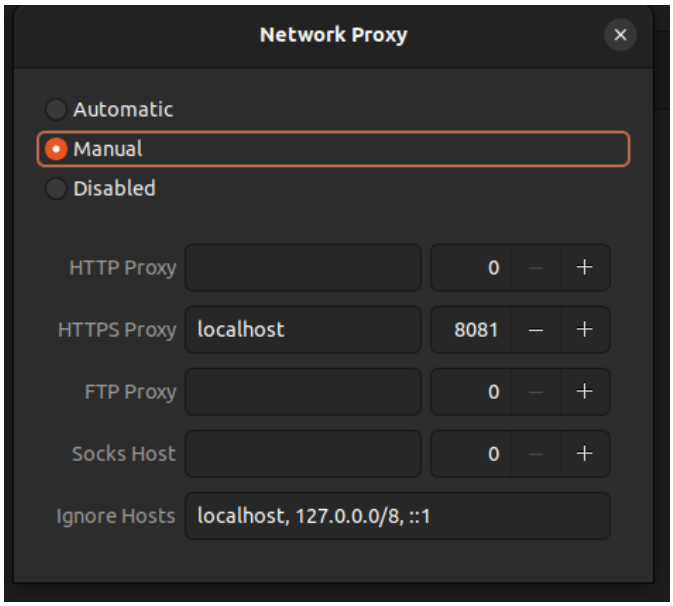
This is how we added Proxy and Mindfactory:

```
openssl x509 -req -in mindfactory.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
    sermindfactoryver.crt -days 365 -sha256
Certificate request self-signature ok
subject=C = DE, ST = HH, L = HH, O = Universit\C3\83\C2\A4t, OU = Mindfactory, CN = www.mindfactory.de

openssl x509 -req -in proxy.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out proxy.crt -days 365 -sha256
Certificate request self-signature ok
subject=C = DE, ST = HH, L = HH, O = Universit\C3\83\C2\A4t, OU = Proxy, CN = www.mindfactory.de
```

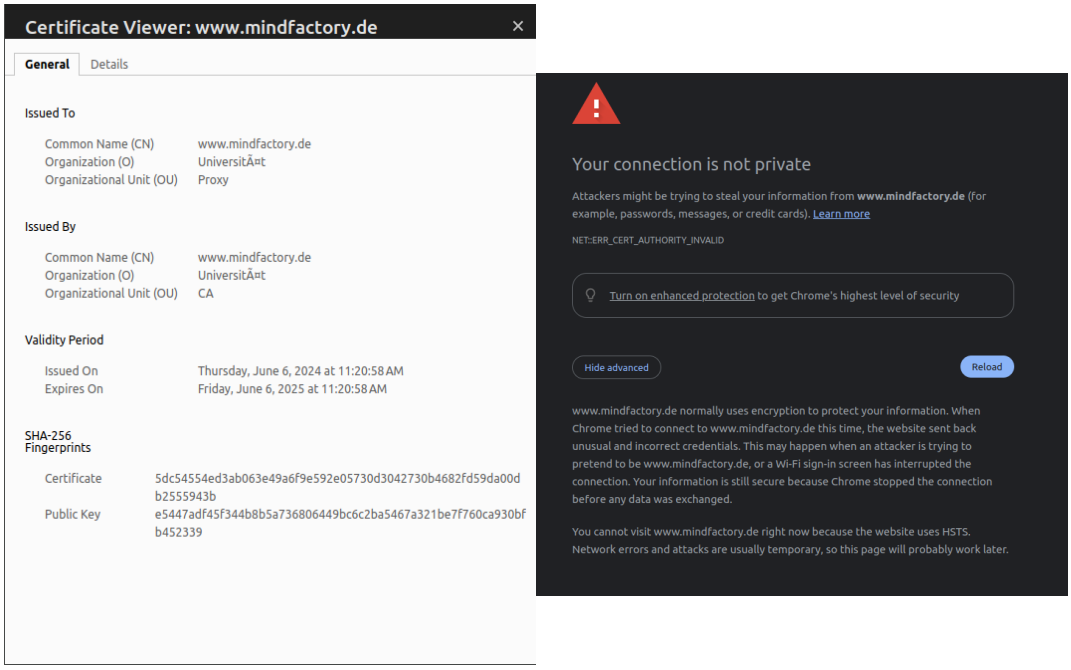
The new ca.crt has to be added in the browser. We tried to use Google Chrome but also tested in Firefox:



Now we started a local proxy to forward the HTTPS traffic to our localhost, so that *Proxy.py* can see the traffic:



After we started *Proxy.py* we tried to connect to the website, but sadly we failed to resolve the following error and are stuck here:



So the correct certificate was loaded by the browser, but we were not able to actually visit the website. Maybe we are missing a security-option that has to be deactivated for it to work, but for now we can only steal credentials locally but not connect to real websites.

4 Certificate Pinning (Time spent: 1h)

Certificate Pinning is a security mechanism to prevent man-in-the-middle attacks by associating a server's SSL/TLS certificate with its expected public key or hash. It ensures trust in certificates beyond relying solely on the system's list of trusted CA certificates. Uses include reducing the risk of trusting compromised CA certificates, preventing SSL stripping and man-in-the-middle attacks, and ensuring communication only with servers whose certificates match pinned values. However, it also has limitations such as requiring careful management of pinned certificates or keys, leaving all clients vulnerable if pinned values are compromised, and potential interoperability issues.

Since we did not get the Browser to work in part3 we implemented certificate pinning locally with our working Client-Proxy-Server-Setup. We implemented a small script to show the basic functionality of HTTPS pinning. Here is the output when connecting to the server:

```
$ python3.9 Cert_pinning.py
SSL established.
Known Public Key:
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAhSvxnQZCLcOQgQ0q74gO
r0W1sSzNK46N/iyNlfGAs0jTZCop0sCr0JF0o9IPdusuF6FeuxIp7KaDGZ9ObHYe
IHMuFB+fUg7UJHEIr4e3oVxe/nYFB5lCLahIUYLiQm99JE6UMxIzjHi57ByyFLy9
cJDJtp2lXdauNmnmCHZ0cpXdGYNnTj8xCF4DG3/Suu5cgiH3Ggd9+Nridbyrvb3k
wPhI6YJJoFSqaqXL+UrmKNmRfcnYp/Aw/8q1Uo7rOS/KjJ0Li6F7Li0EpmLzEpdv
dIFCvGxPUnZ11xPRXaoxLXQV/0DoTIiL1ntIOtVElUiU0P+jQWGBtNkC2+fCP26/
KQIDAQAB
-----END PUBLIC KEY-----
Server Public Key:
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAhSvxnQZCLcOQgQ0q74gO
r0W1sSzNK46N/iyNlfGAs0jTZCop0sCr0JF0o9IPdusuF6FeuxIp7KaDGZ9ObHYe
IHMuFB+fUg7UJHEIr4e3oVxe/nYFB5lCLahIUYLiQm99JE6UMxIzjHi57ByyFLy9
cJDJtp2lXdauNmnmCHZ0cpXdGYNnTj8xCF4DG3/Suu5cgiH3Ggd9+Nridbyrvb3k
wPhI6YJJoFSqaqXL+UrmKNmRfcnYp/Aw/8q1Uo7rOS/KjJ0Li6F7Li0EpmLzEpdv
dIFCvGxPUnZ11xPRXaoxLXQV/0DoTIiL1ntIOtVElUiU0P+jQWGBtNkC2+fCP26/
KQIDAQAB
-----END PUBLIC KEY-----
Server public key is verified.
Response received from server:

HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html><html><body><h1>Hello, world!</h1></body></html>
```

And here when connecting to the MIM proxy:

```
$ python3.9 Cert_pinning.py
SSL established.
Known Public Key:
-----BEGIN PUBLIC KEY-----
```

```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAhSvxnQZCLcOQgQ0q74gO
r0W1sSzNK46N/iyNlfGAs0jTZCop0sCr0JF0o9IPdusuF6FeuxIp7KaDGZ9ObHYe
IHMuFB+fUg7UJHEIr4e3oVxe/nYFB5lCLahIUYLiQm99JE6UMxIzjHi57ByyFLy9
cJDJtp2lXdauNmnmCHZ0cpXdGYNnTj8xCF4DG3/Suu5cgiH3Ggd9+Nridbyrvb3k
wPhI6YJJoFSqaqXL+UrmKNmRfcnYp/Aw/8q1Uo7rOS/KjJ0Li6F7Li0EpmLzEpdv
dIFCvGxPUnZ11xPRXaoxLXQV/0DoTIiL1ntIOtVElUiU0P+jQWGBtNkC2+fCP26/
KQIDAQAB
-----END PUBLIC KEY-----
Server Public Key:
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvM4wJ9c23/qfwjJLDtUN
B23hAAfL/1k4rtlt7vSQ863prAoBBzmfGRAtG/Kkc0o5EH/mQzBVv2l9n8zjAjBo
79+xkkcJUGQnpVW3A/eo+n4o4g+X8Aryerwaiuix5RflEsbl3NefKkMq/GEUHQTL
i0vy+ikfmXSyCtfdXrkEFuoTZNxJtvSUAXX4yvXLi7Ee4DezNdN4fBFoX7OrMA/I
RoAufCH9sCLZIJD3ZnCYBuR4ixbXB086VnQA1td6wZvLsDaTDHmbAC+SYV2zwBdj
palKRM23+Qp3F5Q2R0iKMBCWVZ/Aj7tWMM1IbNJh0zX6JTXNE6MY4eAbQQTBT0WX
5QIDAQAB
-----END PUBLIC KEY-----
Server public key verification failed.
```

So if we at least once connected correctly, we can detect fraud.