University Göttingen
Department of Computer Science
Jonathan Decker

Contents

Required tools:

- openssl
- Internet Connection
- python3
- ca-certificates
- Cluster-manager with public ipv4 address
- bind-utils
- certbot

Task 1: HTTPS Server with Self-Signed Certificate (10 min)

The goal of this exercise is to run a very simple HTTPS server in Python.

First generate the self-singed certificate:

```
openssl req -new -x509 -nodes -keyout key.pem -sha256 -days 365 -out cert.pem
```

Set CN to `localhost`. The other fields do not matter.

Notice how both are PEM files.

Verify the key:

```
openssl rsa -noout -text -in key.pem
```

Verify the certificate:

```
openssl x509 -noout -text -in cert.pem
```

Create a file called **https-server.py** and copy the listing below into it. Adjust the PORT value as needed to an unused port.

Listing 1: https-server.py

```python
from http.server import HTTPServer, BaseHTTPRequestHandler
import ssl

HOSTNAME = "localhost"
PORT = 8000

class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
  def do_GET(self):
    self.send_response(200)
    self.end_headers()
    self.wfile.write(b'Hello, world!')

httpd = HTTPServer((HOSTNAME, PORT), SimpleHTTPRequestHandler)

httpd.socket = ssl.wrap_socket (httpd.socket,
  keyfile="key.pem",
  certfile='cert.pem', server_side=True)

print("Server starting at https://" + HOSTNAME + ":" + str(PORT))

httpd.serve_forever()
```

Run the server with

```
python3 https-server.py
```

You might have to install Python3 via yum.

Move the server to a background job in your terminal by pressing CTRL + Z and typing `bg`.
You can move the job back into foreground with `fg`.

Send a request to the server via

```
curl https://localhost:8000
```

Notice the error message.

Disable certificate chain validation to see our **Hello, world!** message.

```
curl -k https://localhost:8000
```

Now to trust our own certificate, we need to add it to the systems trust store.

```
sudo cp cert.pem /etc/pki/ca-trust/source/anchors/
sudo update-ca-trust
```

Try again

```
curl https://localhost:8000
```

You should see the same response as when using the `-k` option.

You now have a working local HTTPS server. This is useful for testing when your production systems also run HTTPS.
To become a real CA, all you need to do is put your `cert.pem` file on every internet-facing system on the planet.
This is left as a homework.

Task 2: Creating CA-Signed Certificates (5 min)

Imagine you have multiple sites and each should have its own valid HTTPS cert without having to install a root CA for every single one.
The goal is now to create another certificate and to sign it using our first CA certificate, while including all subdomains that development might use.

First generate a new key:

```
openssl genrsa -out test.key 2048
```

Then create a CSR from it:

```
openssl req -new -key test.key -out test.csr
```

Fill out the questions, the answers do not matter and leave the challenge password empty.

Create the X.509v3 certificate extension config file, which enables us to use SANs (Subject Alternative Names). Paste the code below into a file called `test.ext`.

Listing 2: test.ext

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage =digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = localhost
DNS.2 = test.dev
```

Here `localhost` and `test.dev` will be our development domains.
For a more complete documentation see `https://www.openssl.org/docs/man1.1.1/man5/x509v3_config.html`

Create a new signed certificate:

```
openssl x509 -req -in test.csr -CA cert.pem -CAkey key.pem -CAcreateserial \
  -out test.crt -days 365 -sha256 -extfile test.ext
```

Inspect your new certificate:

```
openssl x509 -noout -text -in test.crt
```

Now edit the Python code for the HTTPS server to use the newly generated certificate and key instead. Afterward, restart the HTTPS server and try to `curl` again.

By running a local DNS server you can attach these development domains within your local network and via the setup shown above, you can successfully serve HTTPS traffic over them.

Task 3: HTTPS via the Internet (5 min)

The next goal is to serve HTTPS traffic via the public internet.

First find the public IP address of your cluster-manager. It is the same address you used to connect to the VM via SSH.

Use a reverse IP lookup tool to find domains belonging to that IP address.
For this you can use the tool `dig` from the `bind-utils` package.

For a reverse IP lookup use `dig -x YOUR_IP`.

In the `ANSWER SECTION` you should see a domain of the form `cXXX-XXX.cloud.gwdg.de`, where X is a number. This is the domain that was automatically assigned by the GWDG for your public IP.

Set the certificate extension config to include this DNS and create a new certificate.
Set the Python HTTPS server to use this new certificate and set `HOSTNAME` to `0.0.0.0`. You might need to run the server with `sudo` now as it attempts to serve on all interfaces.

On your local machine open `https://cXXX-XXX.cloud.gwdg.de:PORT` where X is replaced with the numbers from your domain and PORT is the port set in the Python HTTPS server.

You should get a certificate error saying that the certificate is not secure.
This is what we expect as you have not installed the CA on your workstation but only on the server itself.
Verify this by doing a `curl` from the cluster-manager to your domain.

To remove the warning, the certificate in use must be signed by a known CA.
This will be **Let's Encrypt** in the next task.

Task 4: Getting a Certificate from Let's Encrypt (10 min)

The goal is to have a valid HTTPS certificate installed in our Python HTTPS server that is accepted by the browser on our workstation without having to modify the trust store on our workstation.

First install the `certbot` package via `yum`.

Make sure your port 80 is open as the next step requires running the HTTP challenge over this port.

Stop the Python HTTPS server and run:

```
sudo certbot certonly --standalone
```

Give a valid email address, agree to the terms of service, reject the newsletter and then give your domain.

This will perform a HTTP challenge on port 80 and create a new certificate at
`/etc/letsencrypt/live/cXXX-XXX.cloud.gwdg.de/fullchain.pem`
and the private key for it at
`/etc/letsencrypt/live/cXXX-XXX.cloud.gwdg.de/privkey.pem`

Inspect the certificate that was generated for you using `openssl`.

Edit the Python HTTPS server and point it to `fullchain.pem` for the certificate file and to `privkey.pem` for the keyfile and restart the HTTPS server.

Again open `https://cXXX-XXX.cloud.gwdg.de:PORT` on your workstation and observe how the certificate is now accepted.

Optional Task 5: Installing Certificates (20 min)

This is a difficult **additional** task which will support your understanding in the topic.

Pick one of the applications you have installed over the course of this week, which exposes any service via a webpage.
Look up the documentation for that application and attempt to install the Let's Encrypt certificates into it.

Further Reading

- https://jamielinux.com/docs/openssl-certificate-authority/index.html
- https://github.com/trimstray/the-book-of-secret-knowledge#tool-openssl