Week 10 Practical: Digital certificates & TLS using *asyncio*

Dr. Qublai Ali Mirza

University of Gloucestershire

qalimirza@glos.ac.uk

UNIVERSITY OF GLOUCESTERSHIRE

Recap

- Last week we looked at generating digital certificates and how to use them to provide digital signatures
- This week we will be looking at using digital certificates to establish secure client/server communication
- But before doing so, 10 minutes to complete the post-sessional work from last week.

Environment setup

- For our practical session today, we will be using
  - Setting up a *VM* using VirtualBox
  - *asyncio* module for `Python`

Setting up a Virtual Machine (VM)

- First of all download the Ubuntu 16.04 (64-bit) ISO from the website
- Then open *VirtualBox* and create a virtual machine
- Once imported, fire up the virtual machine and open the `Terminal` application on *Ubuntu*
- Then change the *domain name* of the VM by:
  - `sudo gedit /etc/hosts`
  - `127.0.0.1 test1.example.com`
  - Click on *Save*

Setting up a Virtual Machine (VM)

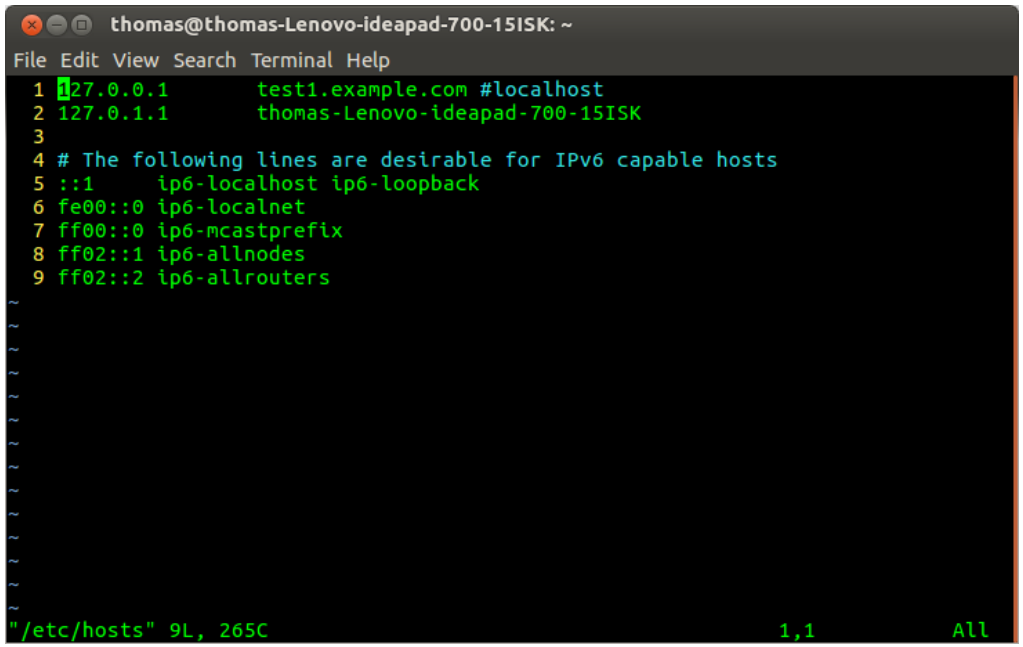

Figure: Updated `/etc/hosts` file content

Installing asyncio

- Next we need to install the *asyncio* module for `Python`
- To that end, type in: `sudo pip3 install asyncio pyopenssl` to install the package

Client-Server using SSL

Figure: What we are going to work on for this lab

Generate Certificate I

```
import random
import sys
from OpenSSL import crypto

def create_root_ca():
    pkey = crypto.PKey()
    pkey.generate_key(crypto.TYPE_RSA, 4096)

    cert = crypto.X509()
    cert.set_version(3)
    cert.set_serial_number(int(random.random() * sys.maxsize))
    cert.gmtime_adj_notBefore(0)
    cert.gmtime_adj_notAfter(60 * 60 * 24 * 365)

    subject = cert.get_subject()
```

Generate Certificate II

```
subject.CN = "example.com"
subject.O = "mycommonname"

issuer = cert.get_issuer()
issuer.CN = "example.com"
issuer.O = "mycommonname"

cert.set_pubkey(pkey)
cert.add_extensions([
    crypto.X509Extension(b"basicConstraints", True,
                         b"CA:TRUE"),
    crypto.X509Extension(b"subjectKeyIdentifier",
    False, b"hash", subject=cert)
])
cert.add_extensions([
```

Generate Certificate III

```
        crypto.X509Extension(b"authorityKeyIdentifier",
         False, b"keyid:always",issuer=cert)
])
cert.sign(pkey, "sha1")

with open("root.pem", "wb") as certfile:
    certfile.write(
    crypto.dump_certificate(crypto.FILETYPE_PEM, cert))
    certfile.close()

with open("root.key", "wb") as pkeyfile:
    pkeyfile.write(
    crypto.dump_privatekey(crypto.FILETYPE_PEM, pkey))
    pkeyfile.close()
```

Generate Certificate IV

```
def create_certificate(cn, o, serverside,
certfilename, pkeyfilename):
    rootpem = open("root.pem", "rb").read()
    rootkey = open("root.key", "rb").read()
    ca_cert = crypto.load_certificate(
              crypto.FILETYPE_PEM, rootpem)
    ca_key = crypto.load_privatekey(
             crypto.FILETYPE_PEM, rootkey)

    pkey = crypto.PKey()
    pkey.generate_key(crypto.TYPE_RSA, 2048)

    cert = crypto.X509()
    cert.set_serial_number(int(random.random() * sys.maxsize))
    cert.gmtime_adj_notBefore(0)
```

Generate Certificate V

```
cert.gmtime_adj_notAfter(60 * 60 * 24 * 365)
cert.set_version(3)

subject = cert.get_subject()
subject.CN = cn
subject.O = o

if serverside:
    cert.add_extensions([crypto.X509Extension(
        b"subjectAltName", False,
        b"DNS:test1.example.com,DNS:test2.example.com")])

cert.set_issuer(ca_cert.get_subject())

cert.set_pubkey(pkey)
```

Generate Certificate VI

```
    cert.sign(ca_key, "sha1")


    with open(certfilename, "wb") as certfile:
        certfile.write(crypto.dump_certificate(
        crypto.FILETYPE_PEM, cert))
        certfile.close()

    with open(pkeyfilename, "wb") as pkeyfile:
        pkeyfile.write(crypto.dump_privatekey(
        crypto.FILETYPE_PEM, pkey))
        pkeyfile.close()


print("Making root CA")
```

Generate Certificate VII

```
create_root_ca()
print("Making server certificate")
create_certificate("server", "my organisation",
True, "server.crt", "server.key")
print("Making client certificate")
create_certificate("client", "my organisation",
False, "client.crt", "client.key")
```

Setting up server I

```
import asyncio
import ssl


@asyncio.coroutine
def client_connected(reader, writer):
    writer.write(b"Hello world! ^_^\n")
    writer.close()


sslcontext = ssl.create_default_context(
             purpose=ssl.Purpose.CLIENT_AUTH)
sslcontext.verify_mode = ssl.CERT_REQUIRED
sslcontext.load_cert_chain(certfile="server.crt",
                           keyfile="server.key")
```

Setting up server II

```
sslcontext.load_verify_locations("root.pem")

print(sslcontext.cert_store_stats())
loop = asyncio.get_event_loop()
asyncio.async(asyncio.start_server(client_connected,
 "test1.example.com", 1234, ssl=sslcontext))

loop.run_forever()
```

Setting up client I

```
#!/usr/bin/python3

import asyncio
import ssl


@asyncio.coroutine
def connect(loop):
    sslcontext = ssl.create_default_context(
                 purpose=ssl.Purpose.SERVER_AUTH)
    #sslcontext.check_hostname = False
    sslcontext.load_verify_locations("root.pem")
    sslcontext.load_cert_chain(certfile="client.crt",
                               keyfile="client.key")
```

Setting up client II

```
    reader, writer = yield from
    asyncio.open_connection("test1.example.com",
    1234, ssl=sslcontext, loop=loop)
    data = yield from reader.read()
    print(data)
    return

loop = asyncio.get_event_loop()

loop.run_until_complete(connect(loop))
```

Bringing it all together

- This week we looked at *Digital signatures & Certificates*
- We also looked at how to generate a digital certificate and use it in verification process
- Next week: *Modern Cryptanalysis & Secure Communication*

Post-sessional work

- Using the in-lab exercise at starting point, create a public key certificate of your own and use it to create a digital signature for both:
  - a *PDF* file
  - a *PNG* file of your choosing.
- **Hint:** you might want to use the `timeit.timeit` function

Q & A