SIL765: Networks and System Security

Semester II, 2021-2022

Assignment-3

February 28, 2022

Problem-1: Understanding Transport Layer Security (40 Marks)

In this assignment, we focus on the TLS handshake protocol implemented on the client side.

```
#!/usr/bin/python3
import socket, ssl, sys, pprint
hostname = sys.argv[1]
port = 443
cadir = '/etc/ssl/certs'

# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True

# Create TCP connection
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((hostname, port))
input("After making TCP connection. Press any key to continue ...")

# Add the TLS
ssock = context.wrap_socket(sock, server_hostname=hostname,
do_handshake_on_connect=False)
ssock.do_handshake() # Start the handshake
pprint.pprint(ssock.getpeercert())
input("After handshake. Press any key to continue ...")

# Close the TLS Connection
ssock.shutdown(socket.SHUT_RDWR)
ssock.close()
```

Listing 1: Basic client program for TLS handshake with a server.

Task-1: TLS Handshake

The above client program initiates a TLS handshake with a TLS server (the name of the server needs to be specified as the first command line argument). In this task, you can use this code to

communicate with a real HTTPS-based web server and perform the following. Note that some additional code may need to be added to complete the task.

1. What is the cipher used between the client and the server?
2. Please print out the server certificate in the program.
3. Explain the purpose of `/etc/ssl/certs`.
4. Use Wireshark to capture the network traffics during the execution of the program, and explain your observation. In particular, explain which step triggers the TCP handshake, and which step triggers the TLS handshake. Explain the relationship between the TLS handshake and the TCP handshake.

Task-2: CA's Certificate

In the client program, we use the certificates in the `/etc/ssl/certs` folder to verify server's certificates. In this task, you will create your own certificate folder, and place the corresponding certificates in the folder to do the verification as follows.

1. Create a folder called `certs`, and assign the `cadir` to `./certs`.
2. Run the client program. Since the folder is empty, the program will throw an error. Report your observation and the potential cause.
3. It may be possible to resolve the previous error by placing the corresponding CA's certificate into your `certs` folder. Please use the client program to find out what CA certificate is needed to verify the server's certificate, and then copy the certificate from the `/etc/ssl/certs` to the `./certs` folder.
4. Run the client program again and report the observation. If you have done everything correctly, your client program should be able to talk to the server.

It should be noted that copying the CA's certificate to the `./cert` folder is not enough. When TLS tries to verify a server certificate, it will generate a hash value from the issuer's identify information, use this hash value as part of the file name, and then use this name to find the issuer's certificate in the `./cert` folder. Therefore, we need to rename each CA's certificate using the hash value generated from its subject field, or we can make a symbolic link out of the hash value. You can use the following commands where we use openssl to generate a hash value, which is then used to create a symbolic link.

```
$ openssl x509 -in someCA.crt -noout -subject_hash
4a6481c9

$ ln -s someCA.crt 4a6481c9.0
$ ls -l
total 4
lrwxrwxrwx 1 ... 4a6481c9.0 -> someCA.crt
-rw-r--r-- 1 ... someCA.crt
```

```
$ dig www.example.com
...
;; ANSWER SECTION:
www.example.com. 403 IN A 93.184.216.34
```

Task-3: Hostname

The objective of this task is to make you understand the importance of hostname checks at the client side. Please conduct the following steps using the client program.

1. Get the IP address of the server using the `dig` command, such as the following:
2. Modify the `/etc/hosts` file, add the above entry at the end of the file (the IP address is what you get from the `dig` command).

```
93.184.216.34 www.example2020.com
```

3. Switch the following line in the client program between `True` and `False`, and then connect your client program to `www.example2020.com`. Describe and explain your observation.

```
context.check_hostname = False  #try both True and False
```

Based on the above experiment, explain the importance of hostname check. If the client program does not perform the hostname check, explain the security consequences.

Task-4: Communicating Data

In this task, we will send data to the server and get its response. Since we choose to use HTTPS servers, we need to send HTTP requests to the server; otherwise, the server will not understand our request. The following code example shows how to send HTTP requests and how to read the response.

```
#Send HTTP Request to Server
request = b"GET / HTTP/1.0\r\nHost: " + \
    hostname.encode('utf-8') + b"\r\n\r\n"
ssock.sendall(request)
# Read HTTP Response from Server
response = ssock.recv(2048)
while response:
    pprint.pprint(response.split(b"\r\n"))
    response = ssock.recv(2048)
```

1. Please add the data sending/receiving code to your client program, and report your observation.
2. Please modify the HTTP request, so you can fetch an image file of your choice from an HTTPS server (there is no need to display the image).

Submission

- `tls_client:` This should contain the client's code.
- `readme-problem-1:` This should be the pdf file containing all the necessary details about your solution. For instance, it should explain the steps to execute your code. It should have the screenshots of terminals to demonstrate that your code works as desired. It should contain discussions about the security and efficiency of the protocol.

Problem-2: Implementing Customized TLS (60 Marks)

In this problem, you will prototype the transport layer security (TLS) protocol to securely send a message from the server to the client. The message to be sent is "The OTP for transferring Rs 1,00,000 to your friend's account is 256345." The solution of this problem expects the following functionalities:

- *Trusted third party (TTP):* The TTP issues digital certificates to both the client and the server.
- *Server:* The server generates a public/private key pair, obtains a digital certificate from the TTP, performs the TLS handshake protocol with the client (including the protocol negotiation, authenticated key exchange and key transcript confirmation), and finally performs the TLS record protocol to securely send the message to the client.
- *Client:* The client generates a public/private key pair, obtains a digital certificate from the TTP, performs the TLS handshake protocol with the server (including the protocol negotiation, authenticated key exchange and key transcript confirmation), and finally performs the TLS record protocol to securely receive the message sent from the server.

The TTP, the server and the client have access to the following cipher suite.

- *Asymmetric key algorithm:* ECDSA or RSA,
- *Symmetric key algorithm:* AES or CHACHA20, and
- *Hashing algorithm:* SHA256 or SHA384

You are free to select the specific parameters of the algorithms. However, make sure that the three entities utilizes three different specifications from the available cipher suit (wherever it is possible). For instance, for creating a digital signature, the TTP may utilize RSA with SHA256, the server may utilize ECDSA with SHA384, and the client may utilize ECDSA with SHA256.

The three entities should be prototyped in three files namely: `my_ttp`, `my_server` and `my_client`. Your source codes should present the complete functionality (as discussed above) when the three files are executed in three different terminals in the following order: `my_ttp`, then `my_server`, and finally `my_client`.

Submission

- `my_ttp:` This should contain the trusted third party's source code.
- `my_server:` This should contain the server's source code.
- `my_client:` This should contain the client's source code.

- `readme-problem-2:` This should be the pdf file containing all the necessary details about your solution. For instance, it should explain the steps to build and execute your code. It should have the screenshots of terminals to demonstrate that your code works as desired. It should contain discussions about the security and efficiency of the protocol.