Lab 7: Tunnelling and Web Security

Objective: In this lab we will investigate the usage of SSL/TLS and VPN tunnels.

🕮 **YouTube** Demo: https://youtu.be/ASCDJq4Wy9Y

A Web cryptography assessment

The Ssllabs tool (https://ssllabs.com) can be used to assess the security of the cryptography used on a Web site. Pick three of your favouriate sites to scan. Now perform a test on them, and determine:

Site	**Site 1:**	**Site 2:**	**Site 3:**
What grade does the site get?			
The digital certificate key size and type?			
Does the name of the site match the name on the server?			
Who is the signer of the digital certificate?			
The expiry date on the digital certificate?			
What is the hashing method on the certificate?			
If it uses RSA keys, what is the e value that is used in the encryption (M^e mod N)?			
Determine a weak cipher suite used and example why it might be weak?			
Is SSL v2 supported?			
If SSL v2 was supported, what problems might there be with the site (this will require some research)?			
Outline the usage of TLS 1.0/1.1 and 1.2, and identify a problem if one of these TLS versions were not supported?			
Is the site vulnerable to Heartbleed? Is the site vulnerable to DROWN? Is the site vulnerable to BEAST? Is the site vulnerable to POODLE?			

Research questions:

What does TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 identify?

If a site gets a 'T' grade, what is the problem?

If the site was susceptible to Poodle, what is the vulnerability?

Can you find a site which gets an "A+"? What features does a site need to get an "A+" grade?

A.2 We will now create a Python program which calls up the SSLlabs assessment. First create a CSV file (sites.csv) with your sites in it. The format is Name of site, URL:

```
web,site
Cloudflare,www.cloudflare.com
BBC,bbc.co.uk
```

Next enter the following code and run it:

```
# Code from
https://github.com/TrullJ/ssllabs/blob/master/ssllabsscanner.py
import requests
import time
import sys
import logging

API = 'https://api.ssllabs.com/api/v2/'


def requestAPI(path, payload={}):
    '''This is a helper method that takes the path to the relevant
        API call and the user-defined payload and requests the
        data/server test from Qualys SSL Labs.
        Returns JSON formatted data'''

    url = API + path

    try:
        response = requests.get(url, params=payload)
    except requests.exception.RequestException:
        logging.exception('Request failed.')
        sys.exit(1)

    data = response.json()
    return data


def resultsFromCache(host, publish='off', startNew='off', fromCache='on',
all='done'):
    path = 'analyze'
    payload = {
                'host': host,
                'publish': publish,
```

```
                'startNew': startNew,
                'fromCache': fromCache,
                'all': all
            }
    data = requestAPI(path, payload)
    return data


def newScan(host, publish='off', startNew='on', all='done',
ignoreMismatch='on'):
    path = 'analyze'
    payload = {
                'host': host,
                'publish': publish,
                'startNew': startNew,
                'all': all,
                'ignoreMismatch': ignoreMismatch
            }
    results = requestAPI(path, payload)

    payload.pop('startNew')

    while results['status'] != 'READY' and results['status'] != 'ERROR':
        time.sleep(30)
        results = requestAPI(path, payload)

    return results


import csv
with open('sites.csv') as csvfile:
      reader = csv.DictReader(csvfile)
      for row in reader:

            url = row['site'].strip()

            a = newScan(url)
            with open("out3.txt", "a") as myfile:
                  myfile.write(str(row['web'])+"\n"+str(a)+"\n\n\n")
                  print row['web']
```

Note that it will can take a few minutes to perform a single scan. By reading the out3.txt file, outline your findings:

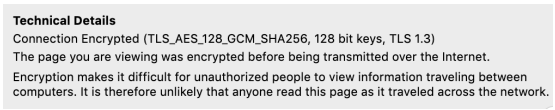
Site name: Site rating:

Other significant details:

Site name: Site rating:

Other significant details:

B Viewing details

No	Description	Result
B.1	On your VM instance (or your desktop), run Wireshark and capture traffic from your main network connection. Start a Web browser and go to **Google.com**. Stop Wireshark and identify some of your connection details:	Your IP address and TCP port: Google's Web server IP address and TCP port: Which SSL/TLS version is used: By examining the Wireshark trace, which encryption method is used for the tunnel (hint: look in the 'Server Hello' response): By examining the Wireshark trace, which hashing method is used for the tunnel (hint: look in the 'Server Hello' response): By examining the Wireshark trace, what is the length of the encryption key (hint: look in the 'Server Hello' response): Using Firefox, and examining the connection details from the site (click on green padlock), can you verify the TLS version, the symmetric key encryption method, the handshaking method and the hashing method used within the tunnel? A sample is shown below. 
B.2	Run Wireshark and capture traffic from your main network connection. Start a Web browser and go to **https://twitter.com**. Stop Wireshark and identify some of your connection details:	Your IP address and TCP port: Twitter's Web server IP address and TCP port: Which SSL/TLS version is used: By examining the Wireshark trace, which encryption method is used for the tunnel:

		By examining the Wireshark trace, which hash method is used for the tunnel: By examining the Wireshark trace, what is the length of the encryption key: Using Firefox, and examining the connection details from the site (click on green padlock), can you verify the TLS version, the symmetric key encryption method, the handshaking method and the hashing method used within the tunnel?

C OpenSSL

No	Description	Result
C.1	On your VM instance (or your desktop), make a connection to the **www.live.com** Web site: `openssl s_client -connect www.live.com:443`	Which SSL/TLS method has been used: Which method is used on the encryption key on the certificate, and what is the size of the public key? Which is the handshaking method that has been used to create the encryption key? Which TLS version is used for the tunnel? Which symmetric encryption method is used for the tunnel: Which hashing method is used for the tunnel: What is the length of the symmetric encryption key: Who has signed the certificate:

D Examining traces

No	Description	Result
D.1	Download the following file, and examine the trace with Wireshark: `http://asecuritysite.com/log/ssl.zip`	Client IP address and TCP port: Web server IP address and TCP port: Determine one of the symmetric key encryption methods, the key exchange, and the hashing methods that the client wants to use (Hint: look at the 'Client Hello' packet)" Which SSL/TLS method has been used: Which encryption method is used for the tunnel: Which hashing method is used for the tunnel: What is the length of the encryption key:
D.2	Download the following file, and examine the trace with Wireshark: `http://asecuritysite.com/log/https.zip`	Client IP address and TCP port: Web server IP address and TCP port: Which SSL/TLS method has been used: Which encryption method is used for the tunnel: Which hashing method is used for the tunnel: What is the length of the encryption key:
D.3	Download the following file, and examine the trace with Wireshark: `http://asecuritysite.com/log/heart.zip`	Client IP address and TCP port: Web server IP address and TCP port: Which SSL/TLS method has been used:

		Which encryption method is used for the tunnel: Which hashing method is used for the tunnel: What is the length of the encryption key:
D.4	Download the following file, and examine the trace with Wireshark: `http://asecuritysite.com/log/ipsec.zip`	Which is the IP address of the client and of the server: Which packet number identifies the start of the VPN connection (Hint: look for UDP Port 500): Determine one of the encryption and the hashing methods that the client wants to use: Now determine the encryption and hashing methods that are agreed in the ISAKMP:
	Download the following file, and examine the trace with Wireshark: `http://asecuritysite.com/log/tor.zip`	Which TCP port does the client use to send to? What is the IP address of the Tor node that the client connects to? What is strange about the packet size? Is SSL/TLS used for the connection? Can you trace any content in the conversation? Can you determine the Web site that is being connected to?

E TLS Connection

E.1 We will now create our own SSL/TLS server and client in Python. First, we need to generate a certificate for our server:

```
openssl req -new -x509 -days 365 -nodes -out mycert.pem -keyout mycert.pem
```

Next we will create a server which will listen on Port 444 (as 443 is likely to be used already for HTTPs), and support two cipher suites (`'AES256+ECDH:AES256+EDH'`):

```
import socket, ssl

context = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
context.load_cert_chain(certfile="mycert.pem")

def handle(conn):
    conn.write(b'GET / HTTP/1.1\n')
    print(conn.recv().decode())

while True:
  sock = socket.socket()
  sock.bind(('', 444))
  sock.listen(5)
  context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
  context.load_cert_chain(certfile="mycert.pem")
  context.options |= ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1  # optional
  context.set_ciphers('AES256+ECDH:AES256+EDH')
  while True:
    conn = None
    ssock, addr = sock.accept()
    try:
      conn = context.wrap_socket(ssock, server_side=True)
      handle(conn)
    except ssl.SSLError as e:
      print(e)
    finally:
      if conn:
        conn.close()
```

Now we will create the client to connect on Port 444. As we have a self-signed certificate, we will disable the checking of the host and certificate (remember to change the IP address to the address of your local host):

```
import socket, ssl

HOST, PORT = '10.10.10.10', 444

def handle(conn):
    conn.write(b'GET / HTTP/1.1\n')
    print(conn.recv().decode())

def main():

    sock = socket.socket(socket.AF_INET)

    context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH)
```

```
    context.check_hostname = False
    context.verify_mode=ssl.CERT_NONE


    context.options |= ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1

    conn = context.wrap_socket(sock, server_hostname=HOST)

    try:
        conn.connect((HOST, PORT))
        handle(conn)
    finally:
        conn.close()

if __name__ == '__main__':
    main()
```

Now run Wireshark (`sudo wireshark &`), and capture from the Ethernet port (a sample run is show in in Figure 1). Now run the server, and then run the client. Stop Wireshark and determine:

The cipher suites sent from client to the server ('Client Hello'):

The cipher suite selected by the server ('Server Hello'):

If we change the code to:

```
context.set_ciphers('HIGH')
```

What are the cipher suites sent from server, and which cipher suite is selected by the client:

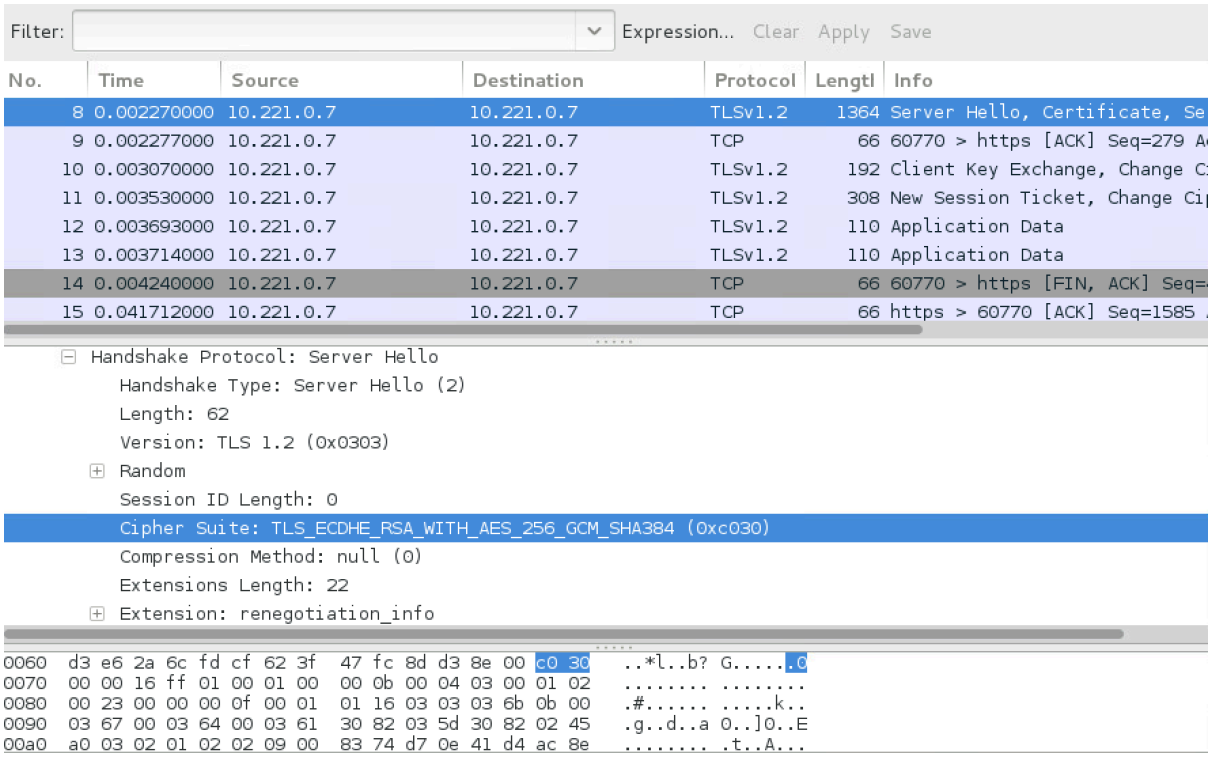


Figure 1: Sample capture

Now select your own cipher suits to accept. The possible settings are given next. You can use the "+" (to add), "-" (to take away), and "!" (for not).

Key exchange:

- **kRSA, aRSA, RSA.** RSA Key exchange.
- **kDHE, kEDH, DH.** Ephemeral DH key agreement.
- **DHE, EDH.** Cipher suites using authenticated ephemeral DH key agreement.
- **kEECDH, kECDHE, ECDH**. Cipher suites using ephemeral ECDH key agreement.
- **ECDHE, EECDH.** Cipher suites using authenticated ephemeral ECDH key agreement.
- **aECDSA, ECDSA**. Cipher suites with ECDSA authentication.

Encryption:

- **AES128, AES256, AES, AESGCM, AESCCM, AESCCM8.**
- **ARIA128, ARIA256, ARIA.**
- **CAMELLIA128, CAMELLIA256, CAMELLIA.**
- **CHACHA20.**
- **3DES, DES, RC4, RC2, IDEA**.

Hashing methods:

- **MD5, SHA1, SHA. SHA256, SHA384**
- **aGOST, kGOST, GOST94, GOST89MAC.**

We can also use: HIGH (256-bit); MEDIUM (128-bit); LOW (56-bit or 64-bit).

G Secure services

G.1 On your VM, determine your IP address with ipconfig, and then using nmap, show the running servers on the server:

ifconfig
nmap <*ip*>

What are the servers that are running:

Open a Web browser on your server, and open up the home page with:

https://<ip>

What is contained on the home page:

G.2 Now to the /var/www/html folder and show that there is a file named index.html. Connect to the sftp service by determining your IP address (<ip>) and use the command:

```
sftp sftpuser@<ip>
```

With this we run the normal FTP service, but integrate with the SSH service (and which runs on Port 22). Now run the following commands, and determine the output:

pwd
ls
cd napier
put index.html

G.3 Now exit from sftp and try and locate the file you have copied. Go back to sftp, and now see if you can copy a file to the /home/napier folder.

Now start wireshark (with **sudo wireshark &),** and capture your session. Now login into your local host with the ssh server:

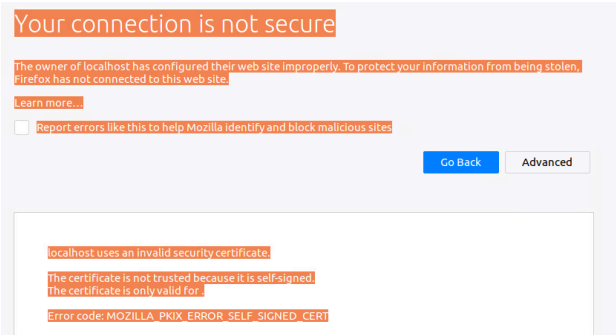
ssh napier@localhost

What observations can you make on the creation of the secure connection:

G.4 Now, let's repeat the lab question from last week. Let's enable HTTPs:

```
sudo a2enmod ssl
service apache2 restart
openssl genrsa -out ca.key 2048
sudo openssl req -nodes -new -key ca.key -out ca.csr
sudo openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt
sudo mkdir /etc/apache2/ssl
sudo cp ca.crt ca.key ca.csr /etc/apache2/ssl/
sudo nano /etc/apache2/sites-enabled/000-default.conf
sudo /etc/init.d/apache2 restart
```

HTTPs should now be enabled with a self-signed certificate. If you try https://localhost, you will have to add an exception to view the page, as we are using a self-signed certificate:



What I should have learnt from this lab?

The key things learnt:

- How do perform a cryptography assessment on a Web site (using ssllabs) and in how to spot weaknesses.
- Able to interpret an SSL/TLS session, and identity the important elements of the Client Hello, and the Server Hello.