SIL765:NetworksandSystemSecurity

Assignment-5

2024JCS2040

Problem-1: Transport Layer security (TLS)

Given client side code establishes a TLS (SSL) connection to a given hostname over port 443, performs a handshake, prints the server's certificate, and then closes the connection.

Task 1: TLS Handshake

Updated the client code to print cipher used and the full server certificate.

These are the certificates and the cipher used:

```
[+] Connecting to www.google.com on port 443...
TCP connection established. Press Enter to continue...
[+] Starting TLS handshake...

[+] Server Certificate:
{'OCSP': ('http://o.pki.goog/wr2',),
 'caIssuers': ('http://i.pki.goog/wr2.crt',),
 'crlDistributionPoints': ('http://c.pki.goog/wr2/GSyT1N4PBrg.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Google Trust Services'),),
            (('commonName', 'WR2'),)),
 'notAfter': 'Jun 23 08:56:20 2025 GMT',
 'notBefore': 'Mar 31 08:56:21 2025 GMT',
 'serialNumber': '3FAF85F6FCC85CEA0A84F8C7E6988F7B',
 'subject': ((('commonName', 'www.google.com'),),),
 'subjectAltName': (('DNS', 'www.google.com'),),
 'version': 3}

[+] Cipher used:
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
Handshake complete. Press Enter to close the connection...
[+] Connection closed.
```

Here we can see the cipher that is used is : TLS_ AES_256_GCM_SHA384 and the server certificate is also seen.

➔ the purpose of /etc/ssl/certs certs.
this is a directory that contains trusted CA (Certificate Authority) certificates on most Linux systems.
When the client connects to a server, it uses these certificates to verify that the server's certificate is signed by a trusted CA.
Without this, the handshake would fail unless the certificate is explicitly trusted.

This network traffics are captured from Wireshark during the execution of the program.

81	4.841349	10.184.61.134	142.250.206.132	TCP	74 50108 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=251448084 TSecr=0 WS=128
82	4.848495	142.250.206.132	10.184.61.134	TCP	74 443 → 50108 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=536 SACK_PERM TSval=1191090414 TSecr=251448084 WS=256
83	4.848973	10.184.61.134	142.250.206.132	TCP	66 50108 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=251448091 TSecr=1191090414
84	7.168367	2603:1063:10::5d	2001:df4:e000:3fd2:…	TLSv1.2	1098 Application Data
85	7.171002	2001:df4:e000:3fd2:…	2603:1063:10::5d	TLSv1.2	837 Application Data
86	7.201244	2603:1063:10::5d	2001:df4:e000:3fd2:…	TCP	74 443 → 53861 [ACK] Seq=1025 Ack=764 Win=501 Len=0
87	7.201374	2001:df4:e000:3fd2:…	2603:1063:10::5d	TLSv1.2	105 Application Data
88	7.230759	2603:1063:10::5d	2001:df4:e000:3fd2:…	TCP	74 443 → 53861 [ACK] Seq=1025 Ack=795 Win=501 Len=0
89	8.085323	10.184.61.134	142.250.206.132	TLSv1.3	583 Client Hello (SNI=www.google.com)
90	8.093630	142.250.206.132	10.184.61.134	TCP	66 443 → 50108 [ACK] Seq=1 Ack=518 Win=268288 Len=0 TSval=1191093659 TSecr=251451328
91	8.095889	142.250.206.132	10.184.61.134	TLSv1.3	590 Server Hello, Change Cipher Spec
92	8.095889	142.250.206.132	10.184.61.134	TCP	590 443 → 50108 [ACK] Seq=525 Ack=518 Win=268800 Len=524 TSval=1191093661 TSecr=251451328 [TCP segment of a reassembled PDU]
93	8.095889	142.250.206.132	10.184.61.134	TCP	590 443 → 50108 [PSH, ACK] Seq=1049 Ack=518 Win=268800 Len=524 TSval=1191093661 TSecr=251451328 [TCP segment of a reassembled PDU]
94	8.095889	142.250.206.132	10.184.61.134	TCP	590 443 → 50108 [ACK] Seq=1573 Ack=518 Win=268800 Len=524 TSval=1191093661 TSecr=251451328 [TCP segment of a reassembled PDU]
95	8.095889	142.250.206.132	10.184.61.134	TCP	590 443 → 50108 [ACK] Seq=2097 Ack=518 Win=268800 Len=524 TSval=1191093661 TSecr=251451328 [TCP segment of a reassembled PDU]
96	8.095889	142.250.206.132	10.184.61.134	TCP	590 443 → 50108 [PSH, ACK] Seq=2621 Ack=518 Win=268800 Len=524 TSval=1191093661 TSecr=251451328 [TCP segment of a reassembled PDU]
97	8.095889	142.250.206.132	10.184.61.134	TCP	590 443 → 50108 [ACK] Seq=3145 Ack=518 Win=268800 Len=524 TSval=1191093661 TSecr=251451328 [TCP segment of a reassembled PDU]
98	8.095889	142.250.206.132	10.184.61.134	TLSv1.3	503 Application Data
99	8.096498	10.184.61.134	142.250.206.132	TCP	66 50108 → 443 [ACK] Seq=518 Ack=1573 Win=63616 Len=0 TSval=251451339 TSecr=1191093661
100	8.096582	10.184.61.134	142.250.206.132	TCP	66 50108 → 443 [ACK] Seq=518 Ack=3145 Win=62080 Len=0 TSval=251451339 TSecr=1191093661
101	8.096601	10.184.61.134	142.250.206.132	TCP	66 50108 → 443 [ACK] Seq=518 Ack=4106 Win=61184 Len=0 TSval=251451339 TSecr=1191093661
102	8.100073	10.184.61.134	142.250.206.132	TLSv1.3	146 Change Cipher Spec, Application Data

In this we are establishing a connection with www.google.com where
client : 10.184.61.134
server : 142.250.206.132

→ This step sock = socket.create_connection((hostname, port)) triggers the tcp handshake
1) client send the SYN packet request to the server to connect with the server → 81 number
2) server replied with SYN, ACK packets means accepts the request → 82

3) client replied with ACK to the server. → 83

Now TCP connection has established.

➔ This step ssock = context.wrap_socket(sock, server_hostname=hostname) triggers the TLS handshake

TLS v1.3 handshake starting
1) client send “Client Hello (SNI = www.google.com)” → 89

2)server send “Server Hello, Change Cipher Spec” → 91

3) the certificate is exchanged which is in encrypted form.

4) client send “Change Cipher Spec, Application Data” → 102

Now the TLS handshake is completed. After this the communication is secure that is the encrypted messages are only shared.

101	8.096601	10.184.61.134	142.250.206.132	TCP	66 50108 → 443 [ACK] Seq=518 Ack=4106 Win=61184 Len=0 TSval=251451339 TSecr=1191093661
102	8.100073	10.184.61.134	142.250.206.132	TLSv1.3	146 Change Cipher Spec, Application Data
103	8.110352	142.250.206.132	10.184.61.134	TCP	66 443 → 50108 [ACK] Seq=4106 Ack=598 Win=268800 Len=0 TSval=1191093676 TSecr=251451342
104	8.608137	10.184.61.134	52.187.79.109	TCP	55 52652 → 443 [ACK] Seq=1 Ack=1 Win=255 Len=1 [TCP segment of a reassembled PDU]
105	8.680100	52.187.79.109	10.184.61.134	TCP	66 443 → 52652 [ACK] Seq=1 Ack=2 Win=251 Len=0 SLE=1 SRE=2
106	9.019596	2001:df4:e000:3fd2:…	2603:1040:a03:9::195	TLSv1.2	125 Application Data
107	9.053098	2603:1040:a03:9::195	2001:df4:e000:3fd2:…	TLSv1.2	114 Application Data
108	9.097578	2001:df4:e000:3fd2:…	2603:1040:a03:9::195	TCP	74 51922 → 443 [ACK] Seq=52 Ack=41 Win=251 Len=0
109	10.176661	10.184.61.134	142.250.206.132	TCP	66 50108 → 443 [FIN, ACK] Seq=598 Ack=4106 Win=64128 Len=0 TSval=251453419 TSecr=1191093676
110	10.186413	142.250.206.132	10.184.61.134	TCP	66 443 → 50108 [FIN, ACK] Seq=4106 Ack=599 Win=268800 Len=0 TSval=1191095752 TSecr=251453419
111	10.186766	10.184.61.134	142.250.206.132	TCP	66 50108 → 443 [ACK] Seq=599 Ack=4107 Win=64128 Len=0 TSval=251453429 TSecr=1191095752
112	14.541030	2606:4700:4400::681…	2001:df4:e000:3fd2:…	TLSv1.2	113 Application Data
113	14.541764	2001:df4:e000:3fd2:…	2606:4700:4400::681…	TCP	74 53874 → 443 [FIN, ACK] Seq=1 Ack=40 Win=255 Len=0
114	14.542630	2606:4700:4400::681…	2001:df4:e000:3fd2:…	TCP	74 443 → 53874 [FIN, ACK] Seq=40 Ack=1 Win=37 Len=0
115	14.542841	2001:df4:e000:3fd2:…	2606:4700:4400::681…	TCP	74 53874 → 443 [ACK] Seq=2 Ack=41 Win=255 Len=0
116	14.597295	2606:4700:4400::681…	2001:df4:e000:3fd2:…	TCP	74 443 → 53874 [ACK] Seq=41 Ack=2 Win=37 Len=0
117	16.486517	2606:4700:4400::681…	2001:df4:e000:3fd2:…	TLSv1.2	113 Application Data
118	16.487268	2001:df4:e000:3fd2:…	2606:4700:4400::681…	TCP	74 53890 → 443 [FIN, ACK] Seq=1 Ack=40 Win=255 Len=0
119	16.487950	2606:4700:4400::681…	2001:df4:e000:3fd2:…	TCP	74 443 → 53890 [FIN, ACK] Seq=40 Ack=1 Win=32 Len=0
120	16.488193	2001:df4:e000:3fd2:…	2606:4700:4400::681…	TCP	74 53890 → 443 [ACK] Seq=2 Ack=41 Win=255 Len=0
121	16.541554	2606:4700:4400::681…	2001:df4:e000:3fd2:…	TCP	74 443 → 53890 [ACK] Seq=41 Ack=2 Win=32 Len=0

Data is transmitted securely and after completion
->the client send FIN, ACK to server
->the server responded with FIN, ACK

->the client send ACK to the server
this is the end of the connection.

➔ Relation between TLS and TCP handshake
TLS runs over TCP
TCP ensures a reliable byte-stream channel.
Once TCP is established, the TLS handshake begins on top of it to negotiate encryption, ciphers, and authentication.

Task 2 : CA's Certificate

Created an empty folder ./certs and assigned it to the cadir

The client program gave certificate verifying error:

```
[+] Connecting to www.google.com on port 443...
TCP connection established. Press Enter to continue...
[+] Starting TLS handshake...
Traceback (most recent call last):
  File "/mnt/c/Users/VADDI JHANCY/desktop/client2.py", line 27, in <module>
    ssock = context.wrap_socket(sock, server_hostname=hostname)
  File "/usr/lib/python3.10/ssl.py", line 513, in wrap_socket
    return self.sslsocket_class._create(
  File "/usr/lib/python3.10/ssl.py", line 1100, in _create
    self.do_handshake()
  File "/usr/lib/python3.10/ssl.py", line 1371, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get
 local issuer certificate (_ssl.c:1007)
```

This is happened because the TLS handshake requires the client to verify the server's certificate using a trusted Certificate Authority (CA).
The empty ./certs folder contains no trusted CA certificates.
So, Python's ssl module fails the certificate verification.

To fix this issue we need to add the trusted CA certificates into the ./certs folder
so I copied the certificates in /etc/ssl/certs into ./certs folder

after coping, the issue is resolved. TLS handshake succeded and the certificate is also verified.

Task 3: Hostname

--First using dig command got the IP address of www.google.com
i.e; dig www.google.com +short
-- Modified the /etc/host file by adding www.google.com ip address and naming it as

fakegoogle.com

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY/desktop$ python3 client2.py fakegoogle.com
[+] Connecting to fakegoogle.com on port 443...
TCP connection established. Press Enter to continue...
[+] Starting TLS handshake...
Traceback (most recent call last):
  File "/mnt/c/Users/VADDI JHANCY/desktop/client2.py", line 27, in <module>
    ssock = context.wrap_socket(sock, server_hostname=hostname)
  File "/usr/lib/python3.10/ssl.py", line 513, in wrap_socket
    return self.sslsocket_class._create(
  File "/usr/lib/python3.10/ssl.py", line 1100, in _create
    self.do_handshake()
  File "/usr/lib/python3.10/ssl.py", line 1371, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: Hostname mism
atch, certificate is not valid for 'fakegoogle.com'. (_ssl.c:1007)
```

When the context.check_hostname = True
it gave an error. This is because the certificate Google serves is for www.google.com, not fakegoogle.com.

after changing context.check_hostname = False

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY/desktop$ python3 client2.py fakegoogle.com
[+] Connecting to fakegoogle.com on port 443...
TCP connection established. Press Enter to continue...
[+] Starting TLS handshake...

[+] Server Certificate:
{'OCSP': ('http://o.pki.goog/wr2',),
 'caIssuers': ('http://i.pki.goog/wr2.crt',),
 'crlDistributionPoints': ('http://c.pki.goog/wr2/GSyT1N4PBrg.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Google Trust Services'),),
            (('commonName', 'WR2'),)),
 'notAfter': 'Jun 23 08:56:20 2025 GMT',
 'notBefore': 'Mar 31 08:56:21 2025 GMT',
 'serialNumber': 'A0FD88AE8BB9F4B510C880F1D48C6A0C',
 'subject': ((('commonName', 'www.google.com'),),),
 'subjectAltName': (('DNS', 'www.google.com'),),
 'version': 3}

[+] Cipher used:
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
Handshake complete. Press Enter to close the connection...
[+] Connection closed.
```

The TLS handshake is success this time, even though the hostname in the cert does not match.

--When context.check_hostname = True:

- Ensures the server's certificate matches the hostname the client is connecting to.
- Prevents man-in-the-middle (MITM) attacks where an attacker presents a valid certificate for a different hostname.

--When context.check_hostname = False:

- The client skips hostname validation.
- It might accept any certificate signed by a trusted CA, even if it belongs to another website.
- This defeats the purpose of TLS, allowing attackers to impersonate websites if they have any valid cert.

--Security Consequences of when we don’t perform the hostname check:
Allows MITM attackers to impersonate a trusted website if they get a cert for any domain.
Clients(we) may trust the wrong server, leading to:
-Credential theft
-Data leaks
-Injection of malicious content

Therefore, Hostname checking is crucial for TLS security. Without it, certificate validation becomes incomplete and insecure.

Task 4: Communicating Data

To send http request and read the response from the server, we are adding the some lines of http codes to handle http request and response.
After the TLS handshake I am adding this line to the code.

```
input("Handshake complete. Press Enter to close the connection...")
# sending http request
request = (
    "GET /images/branding/googlelogo/2x/googlelogo_color_272x92dp.png HTTP/1.1\r\n"
    "Host: www.google.com\r\n"
    "Connection: close\r\n\r\n"
).encode('utf-8')

ssock.sendall(request)
print("[+] Sent HTTP GET request. Receiving response...")

# read the response
response = ssock.recv(2048)
while response:
    pprint.pprint(response.split(b"\r\n"))
    response = ssock.recv(2048)
```

Therefore, After the handshake, the client sends an HTTP GET request
The server responds with headers and HTML content. The response is printed in chunks as split lines (\r\n).

```
[+] Cipher used:
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
Handshake complete. Press Enter to close the connection...
[+] Sent HTTP GET request. Receiving response...
[b'HTTP/1.1 200 OK',
 b'Accept-Ranges: bytes',
 b'Content-Type: image/png',
 b'Cross-Origin-Resource-Policy: cross-origin',
 b'Cross-Origin-Opener-Policy-Report-Only: same-origin; report-to="static-on-bi'
 b'gtable"',
 b'Report-To: {"group":"static-on-bigtable","max_age":2592000,"endpoints":[{"ur'
 b'l":"https://csp.withgoogle.com/csp/report-to/static-on-bigtable"}]}',
 b'Content-Length: 13504',
 b'Date: Sun, 20 Apr 2025 12:42:13 GMT',
 b'Expires: Sun, 20 Apr 2025 12:42:13 GMT',
 b'Cache-Control: private, max-age=31536000',
 b'Last-Modified: Tue, 22 Oct 2019 18:30:00 GMT',
 b'X-Content-Type-Options: nosniff',
 b'Server: sffe',
 b'X-XSS-Protection: 0',
 b'Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000',
 b'Connection: close',
 b'',
 b'']
[b'\x89PNG',
 b'\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02 \x00\x00\x00\xb8\x08\x06'
 b'\x00\x00\x00\xda#W\x1b\x00\x004\x87IDATx\x01\xec\xdd\x03\x90%?\x1e'
 b'\xc0\xf1\x9cm\xdb\xb6}\xaf\x93\xc1\xd9(\x9dY:\x9b\xafn\x92\xf4z\xff\xb6m'
 b'\xdb\xb6\xed\xfb\xdbw\x83$=\xeb\xd7\x97\x9c\xd7\xbb\xf3^\xbf\x99\xe9'
 b'\xfeN\xd5g\x8d\x9e\xd2|\xab\x93_"\xf8X\xfd\xa3\xd5.\x1f\xfd\xe9v\xf9\xc4'
 b'\xc1\xb9\xa3O\x1bZ\xd0y\xd2\xbbv*\x1f#\xca\xf2\x11\x91\x00\x80'
 b'\x1e\x01\xd0\x94O4\x85\x84\x1c\xf1oR\xd6}Y\xe6\xe1\x97\xca\x86'
 b'\xed\x94\xf5\xc7f\xd6_\x94i\x7f[\xa6\xc3xf\xdd*eC\xb9nnIt\x9f\xb4\xe1\xea'
 b'\xe8\x0c\xa9\xc3\x01\xca\xf89\xca\x16?\x94y\x18n\xcd-^\xdc\xe0P\x01\x00\x80'
 b'\x00i\xb7\xcbG\xa6\xd8\x906|_Z\xbf\x872\xee\xca\xcc\xbae)$*\xa5\xc3ht\xa6'
 b'2a\xb1\xb4\xee\x8b\xca\xfa\xe7Ebu\x00\x00\xa06\x9f\xc8\xe0\xbc\xe2\x852\x0f'
 b'\xdfV6\x1c\x9cB \x05\xc1L\x90\xe9pS\xfc~\xab\xf4\x96\xe4\xe3[u\x1e'
 b'\x17\x89\xa6\x03\x00`V?\xfc\xe0\xdc\xc9\x97*\xeb\x7f\x91\x96Q\xba'
 b'\x8f\x85\xeae\xda\x87\xf8\xfd\xfe\xca\x14\x9f\xf9J\xbb|l$\x9a\x08'
 b'\x00\x80Y\xf7\xc0i\x83\xa84\xe1\x1b\x99\x0egW\x1c\x0c\xd5/\xd7'
 b'\xd8\xb0\x95\xb2\xee\x8d\x91h\x12\x00\x00f\xcd\x83\xb6\xf2\xf1W+\x13'
 b'\xb6N\x9bE\x95\re\xad\xe8p\xa6\xcc\x8b\xcf\xa6\xbd+\x91\xd88\x00\xc0\xbb\x0f'
 b'\xfcR\xd9#\x02\xfd7\xe3\x1fP\xea\x89\x0fJ\xe3\x8fP\xdau\x94\re\x9de'
 b':\\\x97\xde\xee\xa4Q\xe0H\xac\x07\x00\xa0\xaa\x00\x01\x012\x90O~ \xd3\xe1'
 b'deC\xd98:\xdc\x90\x19\xf7\xa5\xf5\x8d\xf5\x02\x00z\x1c  @\x06G\xdc'
```

For saving an image, I extracted the bytes data and saved in an png file.

```python
# Read and write image data
response = b""
while True:
    data = ssock.recv(4096)
    if not data:
        break
    response += data

# Separate headers and body
header_end = response.find(b'\r\n\r\n')
headers = response[:header_end]
body = response[header_end+4:]

# Save image
with open("google_logo.png", "wb") as f:
    f.write(body)

print("Image saved as google_logo.png")
```

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY/desktop$ python3 client2.py www.google.com
[+] Connecting to www.google.com on port 443...
TCP connection established. Press Enter to continue...
[+] Starting TLS handshake...

[+] Server Certificate:
{'OCSP': ('http://o.pki.goog/wr2',),
 'caIssuers': ('http://i.pki.goog/wr2.crt',),
 'crlDistributionPoints': ('http://c.pki.goog/wr2/GSyT1N4PBrg.crl',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Google Trust Services'),),
            (('commonName', 'WR2'),)),
 'notAfter': 'Jun 23 08:56:20 2025 GMT',
 'notBefore': 'Mar 31 08:56:21 2025 GMT',
 'serialNumber': '3FAF85F6FCC85CEA0A84F8C7E6988F7B',
 'subject': ((('commonName', 'www.google.com'),),),
 'subjectAltName': (('DNS', 'www.google.com'),),
 'version': 3}

[+] Cipher used:
('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
Handshake complete. Press Enter to close the connection...
[+] Sent HTTP GET request. Receiving response...
Image saved as google_logo.png
[+] Connection closed.
```

Since I requested the logo of the google, I got this image after extracting the data.

TLS only encrypts the communication channel; the content is still regular HTTP protocol over it (i.e., HTTPS = HTTP over TLS).

Fetching binary data (like images) works the same as text — just we need to handle bytes properly.

the python file named client2.py
run this for every task as mentioned in the above and for every task I commented in the code what I have added in each step
to run: python3 client2.py < website name>
example : python3 client2.py www.google.com

Problem 2: Website Security Analysis

For the Vulnerability Assessment and Penetration Testing (VAPT) process different tools can be used like Nmap(Network Mapper), Burp Suite, Metasploit, etc.
Nmap:
Nmap is a powerful open-source network scanning tool used for network discovery and security auditing. It was used in this assessment to identify open ports, detect services running on those ports, and check for known vulnerabilities. The specific mode used was nmap -sV --script=vuln, which enables service version detection (-sV) and executes a collection of vulnerability detection scripts from the Nmap Scripting Engine (--script=vuln). This help in identifying services that might be vulnerable to common exploits or misconfigurations.

Burp Suite:
Burp Suite is a widely-used web application security testing tool developed by PortSwigger. It functions as a web proxy that allows interception and modification of traffic between the browser and web server. We are using the Community Edition of Burp Suite was used in Intercept and Active Scan modes.

Tools Functionalities

1) Nmap
   Nmap was used to perform port scanning and version detection (-sV) on the target server, followed by built-in vulnerability scanning using NSE scripts (--script=vuln). This allowed enumeration of open ports, services, and potential server-side vulnerabilities.
   Command to test the vulnerabilities:
   sudo nmap -sS -sV <website_name> -oN nmap_basic.txt

   -sS : is for SYN scan (Stealth scan)- sends TCP SYN packets to detect open ports.

   -sV : for Service version detection – tries to find the version of services running on open ports.
   Tested on website testphp.vulweb.com

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY$ sudo nmap -sS -sV testphp.vulnweb.com -oN nmap_basic.
txt
[sudo] password for jhancy:
Starting Nmap 7.80 ( https://nmap.org ) at 2025-04-20 21:40 IST
Nmap scan report for testphp.vulnweb.com (44.228.249.3)
Host is up (0.023s latency).
rDNS record for 44.228.249.3: ec2-44-228-249-3.us-west-2.compute.amazonaws.com
Not shown: 999 filtered ports
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.19.0

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 29.97 seconds
```

   Port 80(HTTP) is open → means website is accessible over unsecured Http.
   The server is running Nginx 1.19.0, which is an older version.
   No other ports (SSH, FTP, MySQL, etc.) are accessible — all others are filtered, likely by a firewall.
   →here potential vulnerabilities based on old version of Nginx 1.19.0
   the Nmap itself doesn't directly list vulnerabilities, it gives us service version info, which we can now use to check for known CVEs (Common Vulnerabilities and

Exposures).

known issues in Nginx 1.19.0:

→CVE-2021-23017 – [Critical] – 1-byte memory overwrite

Affects older versions of Nginx before 1.21.0

Occurs when parsing specially crafted DNS responses.

Can lead to remote code execution (RCE) under certain conditions.

→Insecure configuration risks (if default Nginx configs used):

Directory listing enabled.

Missing security headers (X-Frame-Options, CSP, HSTS, etc.).

No rate limiting or WAF (Web Application Firewall).

Potential for HTTP request smuggling if behind certain proxies.

Going into deep to probe port 80 more thoroughly

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY$ sudo nmap -p 80 --script http-enum,http-headers,http
title,http-server-header testphp.vulnweb.com -oN nmap_http_enum.txt

Starting Nmap 7.80 ( https://nmap.org ) at 2025-04-20 21:55 IST
Nmap scan report for testphp.vulnweb.com (44.228.249.3)
Host is up (0.0034s latency).
rDNS record for 44.228.249.3: ec2-44-228-249-3.us-west-2.compute.amazonaws.com

PORT   STATE    SERVICE
80/tcp filtered http

Nmap done: 1 IP address (1 host up) scanned in 1.70 seconds
```

Using Nmap sent probes to port 80, but got no response back.

here 80/tcp filtered http this means:

-A firewall is blocking the request.

-Rate limiting or intrusion detection system (IDS) is in place and blocking scanning.

-the server may have seen the first scan (earlier command) and temporarily blocked my IP.

Tested on zero.webappsecurity.com

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY$ sudo nmap -sS -sV zero.webappsecurity.com -oN nmap_basic.txt
Starting Nmap 7.80 ( https://nmap.org ) at 2025-04-20 21:53 IST
Nmap scan report for zero.webappsecurity.com (54.82.22.214)
Host is up (0.036s latency).
rDNS record for 54.82.22.214: ec2-54-82-22-214.compute-1.amazonaws.com
Not shown: 997 filtered ports
PORT     STATE SERVICE    VERSION
80/tcp   open  http       Apache Tomcat/Coyote JSP engine 1.1
443/tcp  open  ssl/https?
8080/tcp open  http       Apache Tomcat/Coyote JSP engine 1.1

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 36.20 seconds
```

This tells us that three open port are there: 80,443 and 8080

in these 80 and 8080 uses http protocol and 443 uses https protocol.

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY$ sudo nmap -p 80,8080 --script http-enum,http-title,http-methods,http-headers zero.webappsecurity.com

[sudo] password for jhancy:
Starting Nmap 7.80 ( https://nmap.org ) at 2025-04-20 22:14 IST
Nmap scan report for zero.webappsecurity.com (54.82.22.214)
Host is up (0.070s latency).
rDNS record for 54.82.22.214: ec2-54-82-22-214.compute-1.amazonaws.com

PORT     STATE SERVICE
80/tcp   open  http
| http-enum:
|   /admin/: Possible admin folder
|   /admin/index.html: Possible admin folder
|   /login.html: Possible admin folder
|   /manager/html/upload: Apache Tomcat (401 Unauthorized)
|   /manager/html: Apache Tomcat (401 Unauthorized)
|   /README.txt: Interesting, a readme.
|   /docs/: Potentially interesting folder
|_  /errors/: Potentially interesting folder
| http-headers:
|   Date: Sun, 20 Apr 2025 16:44:48 GMT
|   Server: Apache-Coyote/1.1
|   Access-Control-Allow-Origin: *
|   Cache-Control: no-cache, max-age=0, must-revalidate, no-store
|   Content-Type: text/html;charset=UTF-8
|   Content-Language: en-US
|   Content-Length: 12471
|   Connection: close
|
|_  (Request type: HEAD)
| http-methods:
|   Supported Methods: GET HEAD POST PUT DELETE TRACE OPTIONS PATCH
|_  Potentially risky methods: PUT DELETE TRACE PATCH
|_http-title: Zero - Personal Banking - Loans - Credit Cards
```

```
8080/tcp open  http-proxy
| http-enum:
|   /admin/: Possible admin folder
|   /admin/index.html: Possible admin folder
|   /login.html: Possible admin folder
|   /manager/html/upload: Apache Tomcat (401 Unauthorized)
|   /manager/html: Apache Tomcat (401 Unauthorized)
|   /README.txt: Interesting, a readme.
|   /docs/: Potentially interesting folder
|_  /errors/: Potentially interesting folder
| http-headers:
|   Server: Apache-Coyote/1.1
|   Cache-Control: no-cache, max-age=0, must-revalidate, no-store
|   Content-Type: text/html;charset=UTF-8
|   Content-Language: en-US
|   Content-Length: 12471
|   Date: Sun, 20 Apr 2025 16:44:48 GMT
|   Connection: close
|
|_  (Request type: HEAD)
| http-methods:
|   Supported Methods: GET HEAD POST PUT DELETE TRACE OPTIONS PATCH
|_  Potentially risky methods: PUT DELETE TRACE PATCH
|_http-title: Zero - Personal Banking - Loans - Credit Cards

Nmap done: 1 IP address (1 host up) scanned in 638.47 seconds
```

Open port 80 (http)

Potential Issues:

- Some potentially interesting folders like /admin/, /login.html, and /manager/html/ were detected. These could be entry points for admin access or contain sensitive files.
- PUT, DELETE, TRACE, and PATCH methods are supported, which can be risky because they allow potentially dangerous operations on the server.

Open port 8080(http proxy)

Similar results to port 80:

- Potentially interesting paths (/admin/, /login.html, /manager/html/).
- Risky HTTP methods like PUT, DELETE, TRACE, and PATCH are supported.

HTTP Header Information:

- Server: Apache-Coyote/1.1
- Cache-Control: No-cache and no-store directives, which are typically used to prevent caching of sensitive data.
- Content-Length: 12471 bytes
- Content-Type: text/html;charset=UTF-8
- Access-Control-Allow-Origin: * – This could allow cross-origin requests from any domain, potentially a security concern.

HTTP Methods:

- Supported Methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, PATCH
- Some of these methods (like PUT, DELETE, TRACE, and PATCH) can be risky as they can allow for file uploads, deletion, or manipulation of server resources.

HTTP Title:

- The page on both port 80 and port 8080 appears to be related to personal banking services, with titles indicating features such as "Loans" and "Credit Cards."

Now done testing for college website uceou.edu

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY$ sudo nmap -sS -sV uceou.edu
[sudo] password for jhancy:
Starting Nmap 7.80 ( https://nmap.org ) at 2025-04-20 22:24 IST
Nmap scan report for uceou.edu (202.63.117.90)
Host is up (0.047s latency).
Not shown: 994 filtered ports
PORT     STATE  SERVICE    VERSION
21/tcp   closed ftp
22/tcp   closed ssh
80/tcp   open   http       Apache httpd
443/tcp  open   ssl/ssl    Apache httpd (SSL-only mode)
3306/tcp open   mysql      MySQL (unauthorized)
9090/tcp closed zeus-admin

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 22.05 seconds
```

→Open ports that we can connect to are:

Port 80 (HTTP): A standard web server (Apache httpd) is running.

Port 443 (HTTPS): Secure web server, again Apache but in SSL-only mode, meaning the server only accepts secure connections.

Port 3306 (MySQL): MySQL database server is running. unauthorized, which means the server responded but didn't allow access (no credentials or IP restrictions).

→Filtered Ports (Not shown individually - 994 total)

These are ports that didn’t respond at all or were blocked by a firewall. Nmap couldn’t determine if they’re open or closed, that means:

- Firewall/IDS is actively filtering them.
- Could also be dropped packets or blackholing (intentional ignoring).

therefore this is happening because:

The server has a firewall configured (blocking most ports).

- Only specific services are allowed (like web and database).
- Security best practices are followed by closing/limiting access to FTP, SSH, and admin interfaces.

"unauthorized" on MySQL because:

- We’re not allowed to connect from our IP or without credentials.

- MySQL is bound to public IP, but access is restricted internally.

Now we ran the command nmap --script http-enum,http-vuln* -p 80,443 202.63.117.90
--script http-enum,http-vuln*: Runs two categories of NSE scripts:

- http-enum: Enumerates directories and common files on web servers.
- http-vuln*: Runs all HTTP-related vulnerability scripts.

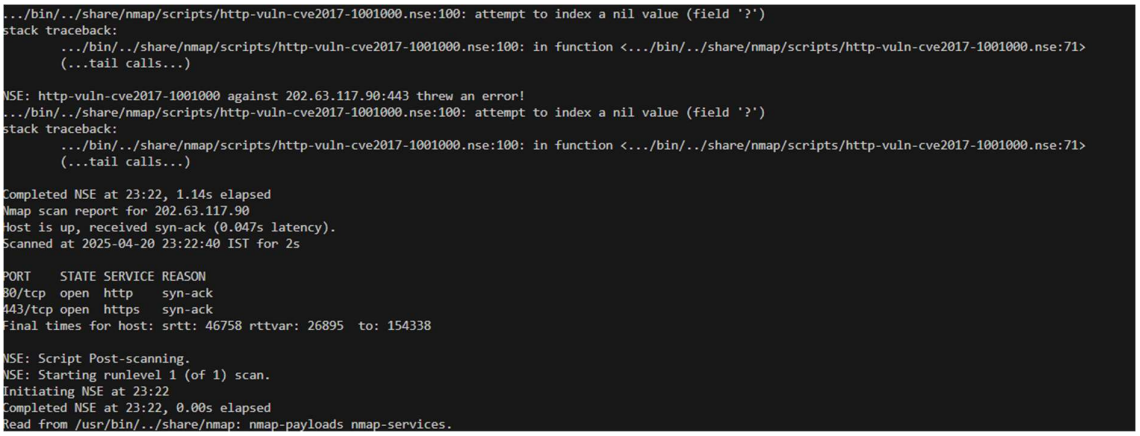
-p 80,443: Scans ports 80 (HTTP) and 443 (HTTPS).
The ip address is uceou.edu university ip address.

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY$ nmap --script http-enum,http-vuln* -p 80,443 202.63.117.90
Starting Nmap 7.80 ( https://nmap.org ) at 2025-04-20 22:30 IST
Nmap scan report for 202.63.117.90
Host is up (0.040s latency).

PORT    STATE SERVICE
80/tcp  open  http
| http-enum:
|   /icons/: Potentially interesting folder w/ directory listing
|_  /manual/: Potentially interesting folder
|_http-vuln-cve2017-1001000: ERROR: Script execution failed (use -d to debug)
443/tcp open  https
| http-enum:
|   /icons/: Potentially interesting folder w/ directory listing
|_  /manual/: Potentially interesting folder
|_http-vuln-cve2017-1001000: ERROR: Script execution failed (use -d to debug)

Nmap done: 1 IP address (1 host up) scanned in 28.60 seconds
```

_http-vuln-cve2017-1001000: ERROR: Script execution failed (use -d to debug)

means This script checks for a specific CVE vulnerability from 2017 related to HTTP servers (e.g., PHP libraries with command injection flaws). But it failed → this doesn't mean the system is safe, just that:
The script couldn't run successfully (maybe due to permissions, a timeout, or a missing dependency).

```
jhancy@DESKTOP-UPF9MGV:/mnt/c/Users/VADDI JHANCY$ nmap -p 80,443 --script=http-vuln-cve2017-1001000 -d 202.63.117.90

Starting Nmap 7.80 ( https://nmap.org ) at 2025-04-20 23:22 IST
--------------- Timing report ---------------
  hostgroups: min 1, max 100000
  rtt-timeouts: init 1000, min 100, max 10000
  max-scan-delay: TCP 1000, UDP 1000, SCTP 1000
  parallelism: min 0, max 0
  max-retries: 10, host-timeout: 0
  min-rate: 0, max-rate: 0
---------------------------------------------
NSE: Using Lua 5.3.
NSE: Arguments from CLI:
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
NSE: Starting runlevel 1 (of 1) scan.
Initiating NSE at 23:22
Completed NSE at 23:22, 0.00s elapsed
Initiating Ping Scan at 23:22
Scanning 202.63.117.90 [2 ports]
Completed Ping Scan at 23:22, 0.05s elapsed (1 total hosts)
Overall sending rates: 42.04 packets / s.
mass_rdns: Using DNS server 10.255.255.254
Initiating Parallel DNS resolution of 1 host. at 23:22
Completed Parallel DNS resolution of 1 host. at 23:22, 0.47s elapsed
Performing system-dns for 1 domain names that were deferred
Initiating System DNS resolution of 1 host. at 23:22
Completed System DNS resolution of 1 host. at 23:22, 0.41s elapsed
DNS resolution of 1 IPs took 0.88s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 0, SF: 0, TR: 1, CN: 0]
Initiating Connect Scan at 23:22
Scanning 202.63.117.90 [2 ports]
Discovered open port 80/tcp on 202.63.117.90
Discovered open port 443/tcp on 202.63.117.90
Completed Connect Scan at 23:22, 0.05s elapsed (2 total ports)
Overall sending rates: 42.10 packets / s.
NSE: Script scanning 202.63.117.90.
NSE: Starting runlevel 1 (of 1) scan.
Initiating NSE at 23:22
NSE: Starting http-vuln-cve2017-1001000 against 202.63.117.90:80.
NSE: Starting http-vuln-cve2017-1001000 against 202.63.117.90:443.
NSE: http-vuln-cve2017-1001000 against 202.63.117.90:80 threw an error!
.../bin/../share/nmap/scripts/http-vuln-cve2017-1001000.nse:100: attempt to index a nil value (field '?')
```



The script http-vuln-cve2017-1001000 checks for a **command injection vulnerability** in certain PHP applications using the **phpmailer library**, which was vulnerable to remote code execution (RCE) if improperly configured.

attempt to index a nil value (field '?') → we got this
This usually happens when the **expected HTTP response or header is missing or malformed**, and the script doesn't handle it gracefully.

→ Two Critical Vulnerabilities Not Found & Their Mitigations
Nmap-Based Not Found Vulnerabilities
1) CVE-2017-1001000 → The target site didn’t respond with vulnerable headers or inputs. Likely patched PHPMailer version or no PHPMailer used.
2) SSL/TLS Weakness → HTTPS server properly configured; did not support deprecated protocols like SSLv2 or weak ciphers.

Mitigations for this:

- Up-to-date software stack
- Secure server configurations with modern TLS

→ Two Critical Vulnerabilities Found & Potential Attacks
Nmap Vulnerabilities
→Open Directory Listing

- /icons/ and /manual/ folders are publicly accessible.
- Impact: Attackers can download files, guess technology stack, or find sensitive documentation.

→Open HTTP Port (80)

- HTTP exposed without redirecting to HTTPS.
- Impact: Allows credential theft or session hijacking via MITM if user is tricked into HTTP.

Mitigation techniques that could be deployed by the website

1) Disable Directory Listing on the Web Server
2) Restrict Access Using Permissions

3) Remove Unused Directories or Move Them Outside Web Root
   Move internal documentation or developer files to a secure, non-public path.
4) Use Web Application Firewall (WAF)
   Prevents exploitation of exposed paths and adds a monitoring layer.
5) Automatically redirect all HTTP traffic to HTTPS.
6) Enable HTTP Strict Transport Security (HSTS)
7) Use SSL Certificates from Trusted Authorities
   Ensure SSL/TLS is valid and up to date to avoid browser trust issues.