**TLS ATTACKS**

**Task 1: TLS Client**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| Code Snippet:  Task1a:  Handshake.py  Task1b:  Handshake.py but *cadir = ‘./client-certs’*  Task1c:  Handshake.py but context.check\_hostname flag will change from True to False for experimentation  Task1d ():  Screenshot:  Running handshake.py with google.com:      TCP 3-way handshake:    TLS 4-way handshake:    Running the normal handshake.py to see CA certificate name:      Adding google.com root CA cert to client-certs directory    Run handshake.py where *cadir = ‘./client-certs’* with 2 different websites:      Getting IP address of [www.google.com](http://www.google.com) using dig command in the VM:  In /etc/hosts file and added the IP address of [www.google.com](http://www.google.com) and with another hostname:    When flag is **context.check\_hostname = True** in handshake.py:    When the flag is **context.check\_hostname = False** in handshake.py:    Data of the website:    Data of the website’s image:    Explanation and observation Task1a:  1. **What is the cipher used between the client and the server?**   Ans: Cipher used is TLS\_AES\_256\_GCM\_SHA384. TLS version 1.3 with AES encryption with a 256-bit key in GCM mode, SHA384 for hashing.   1. **Please print out the server certificate in the program.**   Ans: Here is the server certificate for google.com:    This shows when the server cert is valid from Mar 4th to May 27th 2024, the CA who issued the cert and the issuer’s details. It also gave the url and alternative url (hostname) for the website.   1. **Explain the purpose of /etc/ssl/certs.**   Ans: It is the default directory where certs are located in.   1. **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.**   Ans: There are 3 packets sent for TCP handshake:     1. **SYN**: client initiates the connection by sending a SYN (synchronize) packet to the server. 2. **SYN, ACK**: server responds to the client with a SYN-ACK packet. Indicating that the server has received the client's SYN request and is responding with its own SYN message, along with ACK the client's SYN packet. 3. **ACK**: client completes the TCP handshake by sending an ACK packet back to the server. This acknowledges receipt of the server's SYN-ACK packet. TCP connection is established.   There are 4 packets for TLS for handshake:     1. client starts the handshake by sending a "Client Hello" message. It includes the client’s supported versions of TLS, list of cipher suites. 2. server responds with "Server Hello," selecting the TLS version and cipher suite from the options provided by the client and sending its own parameters for the session. 3. "Server Hello," we should see the server's certificate, certificate verification. 4. client and server confirm that the handshake is complete and begin to exchange encrypted application data.   The relationship TLS and TCP handshakes are that TCP handshakes creates a reliable connection between client and server for communication. TLS requires a reliable connection between two parties (client and server) and a way to communicate between them, in this case was done by TCP handshake. TLS handshake establishes secure and encrypted a communication between client and server. Basically:   * TCP offers a connection between A and B. * TLS offers a secure encrypted between A and B.  Task1b: In this task we created a directory in which we will hold a root CA certificate and will instead be used of /etc/ssl/certs, therefore we need to create a connection to /client-certs:    Here we copy the root cert from /etc/ssl/certs and add them into /client-certs and then we use OpenSSL to create a hash value which will be used to create a symbolic link to the copied root certificate. As I understand it, this last step is important as it allows us to map the issuer of the server certificate to the CA certificate file based on the hash value. Task1c: In this task we got the IP address of a google.com and added it into the /etc/hosts file but the hostname was of a different website. Then we ran handshake.py and we ran it twice and changed the flag: context.check\_hostname. From my observation when the context.check\_hostname flag is true and causes the hostname to be checked. The client will verify if the server's certificate hostname matches the hostname you are connecting to. What I understand is that the server certificate's Common Name (CN) or Subject Alternative Name (SAN) did not match **www.udst.edu.qa**. This is what should be expected and shows that hostname verification. This flag can stop man-in-the-middle attack. But when the flag is False, TLS handshake will work regardless of whether the server's certificate hostname matches the hostname. Task1d: There are two codes for this task. The first code is added into the handshake.py from the lab pdf file and it basically takes an input of an hostname of a server and the HTTP GET request is sent to the server and the program reads it and gives a HTTP response and prints each line of the response.  The other code is basically the same but with only give the data of an image from a server. The code takes 2 command lines inputs: server hostname and url for an image. The program will then output the image’s data. |

**Task 2: TLS Server**

|  |
| --- |
| **Task 2a**  A screenshot of a computer  Description automatically generated  Initially, we made ca.crt and ca.key . These files were obtained from Lab 4 PKI. Then we ran the server.py file to host the server with the domain “www.netsec.com”.  Then we implemented the handshake.py and server.py files. Now if we run the handshake to our domain “www.netsec.com” we get a successful reply:    **Task 2b**  Now we add the certificates to our browser by importing it.  Then We go to the client docker, and in its “/etc/hosts” file, we add the ip and the domain name of our website:  We also add the website to our /etc/hosts in our main VM as well:  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generatedOnce a domain has been assigned, we can access the server by going to our browser and entering “www.netsec.com” in the searchbar:  As we can see, our server is **successfully** **running** and is reachable outside the dockers as well.  **Task 2c**  Due to a different approach in task 2a, instead of mentioning alternative domains in a config files, we mentioned it in the initial cert request for server:  *openssl req -newkey rsa:2048 -sha256 \*  *-keyout server.key -out server.csr \*  *-subj "/CN=www.netsec.com/O=netsec Inc./C=US" \*  *-passout pass:dees \*  *-addext "subjectAltName = DNS:www.netsec.com, DNS:www.netseca.com, DNS:www.netsecb.com"*  A computer screen with a browser window  Description automatically generatedThis way, our certificates got directly mapped to different domain names:  A computer screen shot of a website  Description automatically generated  In this way, we have successfully mapped the same certificate to multiple domains mapping to the same server file. The handshake with the new domain “www.netseca.com” : |

**Task 3: A Simple HTTPS Proxy**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Proxy Container:**    In this task we combine task 1 and task 2 in this task. Our objective is to create simple HTTPS proxy server that can be used to intercept and forward HTTPS traffic, showing how a Man-In-The-Middle (MITM) attack could be performed if the Public Key Infrastructure (PKI) is compromised. To do this we establishes a connection to the intended destination server (hostname). Wraps the socket with SSL to initiate a secure connection. Receives the HTTP request from the client. Forwards the request to the destination server. Receives the response from the server. Forwards the response back to the client. Closes the connection once all data is forwarded. A c ode sets up an SSL context for the proxy server with its own certificate and private key. s ocket that listens on all interfaces (0.0.0.0) on port 443, which is the standard port for HTTPS traffic. proxy server accepts connections from clients. each new connection, it wraps the connection with SSL to secure it, spawns a new thread, and calls process\_request to handle the connection threading allows the proxy to handle multiple connections simultaneously. Re irecting traffic to your proxy server and using a fake certificate signed with a stolen private key from a trusted CA, you have created the conditions for a successful MITM attack. Any data sent to or from www.example.com could be intercepted and, if unencrypted, read or modified by the proxy server. |