COMSW 4181: Security I **(Due: 11/05/21, 11:59 PM EST)**

Homework Assignment #2

Instructor: Suman Jana *Name: , UNI:*

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- You have a total of three late days that you may choose to use in any of the assignments without any penalty. Any submission after the deadline will be considered as a late day and if it exceeds 24 hours after the deadline, then it will be considered as two late days. You may use the late days in separate assignments or together. Once you have used all of your late days, you will not receive any credit for late submissions.
- Due date and time: 11/05/21 11:59 pm. After that, submissions will be counted late. It does not make any difference if it's only 10 minutes or 23 hours.
- You are not allowed to share the solutions to homework problems/programming assignments with your fellow students. You are able to discuss the high level idea how to approach this problem.

Environment: This will be tested on a generic Ubuntu 20 LTS VM. Please use VM machines (e.g., Virtualbox, VMWare) or your Google Cloud Account. For all the coding questions, please use python3.

Disclaimer: No support for alternative environments (ie. Windows, *BSD, etc) will be provided. If you choose to develop in those environments please verify it works on a recommended system before submission.

Instructions: For this assignment, you will implement your own SSL connection based on our sample code `client.py` and `server.py` on your local machine. Then you need to use `Wireshark` to check and analyze the SSL handshake process in details. Finally, you need to use `mitmproxy` to host an attacker's server on your localhost and simulate the man-in-the-middle attack with a vulnerable client `client_vul.py`. Eventually, the attacker will be able to intercept and modify the request sent from `client_vul.py`.

Problem 1: SSL Connection (35 points)

In class, you have learned the basic process of secure communication through an SSL connection between servers and clients. In this exercise, you will be implementing such a communication link.

We have provided you with a client-server skeleton code communicating with each other through http. Specifically, you can run `python server.py` to host a basic server listening at `localhost:80`. Then `python client.py` will allow you to send a get-request to the server. If the communication works correctly, the client will receive a status code `200` indicating the successful connection. However, such a connection is insecure as no certificate is involved to verify the true identity of the server. Therefore, any attacker can spoof to be the server (`localhost:80`) and send malicious packets to the client.

You should edit the `client.py` and `server.py` such that they can establish a SSL handshake and communicate securely with https. Provided client and server use the library `requests` (requests.kennethreitz.org/en/master/) and `http.server` (docs.python.org/3/library/http.server.html). Besides these two libraries, you might also need `ssl` (docs.python.org/3/library/ssl.html). In specific, you should first generate the server's certificate with `openssl`. Then the client should be able to check the server's certificate every time the client sends a get-request. To do that, the client should manually add the server's certificate as the trusted CA with `requests` APIs. Note that there is (for simplicity) no need for the server to check the certificate of the client's. Please submit your edited `client.py`, `server.py`, and the server's certificate and key in `UNI/p1/`.

Problem 2: Wireshark (20 points)

Now that we have a simple SSL client and server (your edited `client.py` and `server.py`), we will take a close look at the connections with Wireshark (wireshark.org/). Wireshark is a network protocol analyzer, which lets you see what's happening on your network by inspecting the packets going out of your machine. Please familiarize yourself with the interface as far as necessary for this problem.

You need to capture all the packets involved in a complete https get-request sent by your `client.py` and the corresponding response from the `server.py`. Save the log into `UNI/p2/packets.pcapng`. In `UNI/p2/summary`, please report the ID numbers of the packets that are used to build up the SSL connection as well the ones used to communicate through get-requests. Briefly explain the handshake process as well as the communicate process with packets that you mentioned. Please submit `UNI/p2/packets.pacpng` and `UNI/p2/summary`.

Problem 3: Man-in-the-middle attack (35+10 points)

The man-in-the-middle attack in communications between a client and a server describes the interception of the client's communications with the server through a third party, without the knowledge of the client. For example, assume you are connected to a public wi-fi hotspot. An attacker might intercept the transmitted data, decrypt and modify it.

In this homework you will be conducting a man-in-the-middle attack (MITM) with `mitmproxy` (`mitmproxy.org`). Specifically, you will use a vulnerable `client_vul.py` provided by us to send a get request to `https://www.google.com` querying about `cat`. Then mitmproxy will host an attacker server at `localhost:8080`, trying to intercept and modify the client's get-request to query about `dog`. Therefore, the client will eventually get the information of `dog` instead of `cat` from google.

Usually, the attacker will hijack a router and intercept all the http and https packets. Since we are simulating the MITM attack on localhost, you just need to set rules to forward all the http/https packets to `localhost:8080` (attacker's server). Note that the attacker server should be transparent (the client must not notice the attacker has intercepted and modified the packets). `mitmproxy` has a very descent document. Please check the document (`docs.mitmproxy.org/stable/`) thoroughly.

Sub-problem1. The client in `client_vul.py` has a severe vulnerability when doing https, such that the MITM attacker can easily intercept and modify the client's request. In particular, the vulnerable client will send a get-request, querying `cat` to `https://www.google.com`. Now, you, as an attacker, need to apply a filter with mitmproxy and intercept the client's get-request. You can use any key words to filter and intercept the get-request packet. Having done so, you should change the query message from `cat` to `dog` and forward the new get-request to `https://www.google.com`. Please submit (1) one screenshot `UNI/p3/cat.png` showing the intercepted client's get-request in `mitmproxy` console (with filter and intercept key words), (2) one screenshot `UNI/p3/dog.png` showing the modified get-request querying about dog in `mitmproxy` console that has been successfully forwarded to google. Note that you should not edit `client_vul.py` in this subproblem.

Sub-problem2. Please fix the vulnerability in `client_vul.py` and submit as `UNI/p3/client_vul.py`. Make sure the MITM attack does not work any more. Submit one screenshot `UNI/p3/safe.png` showing the MITM attack fails in `mitmproxy` console. Briefly explain what's the vulnerability of `client_vul.py` and why your small fix will prevent the MITM attack in `UNI/p3/remedy`.