CSE471: DATA COMMUNICATIONS AND COMPUTER NETWORKS

YEDITEPE UNIVERSITY

FALL 2020

TERM PROJECT – DUE DATE JANUARY 6TH, 2021

As a term project you are expected to develop a socket tunnelling system that will support both UDP and TCP connections. This system should be useful for (a) encrypting non-secure protocols such as POP3, HTTP and IMAP and (b) bypassing deep packet inspection enabled firewall and routers. In order to achieve this, the proposed project requires the utilization of client-server model, where the node behind the firewall acts as a client and the node on the open Internet acts as a server – see Figure 1.

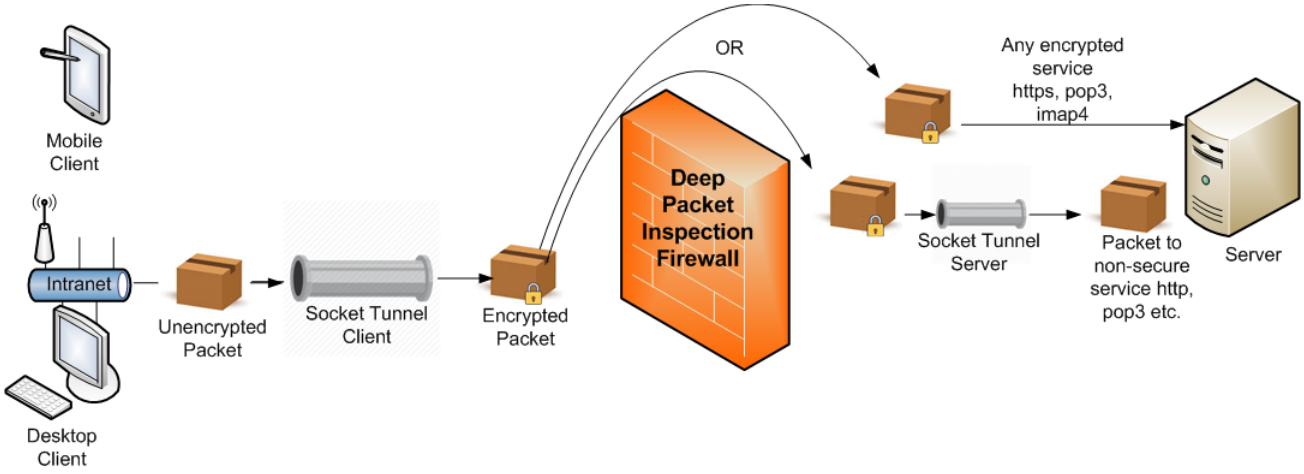

Figure 1: Socket Tunnel Architecture

Your system should eventually create an encrypted socket tunnel between the sender device and receiver device (Figure 1).

- On the client side, there should be a process that listens to one or more TCP/UDP ports based on the settings given by the user. The incoming connection can be encrypted or plain data.
- This process (socket tunnel client) should accept the incoming communication and encrypt the data using SSL/TLS – even if they are already encrypted as a result of their protocol requirements.
- In order for you to achieve this, you will need to create your own private key and certificates pair using OpenSSL, which is freely available for Linux, Windows and Macs. Furthermore, you will be required create a PEM (Privacy-Enhanced Mail) file combining the newly generated private key and certificate.
- Your client application should then be able to encrypt any incoming data stream and relay it to the socket tunnel server's listening port (see Figure 1)

- Your socket tunnel client/server pair should be able to bypass any deep packet inspection router of firewall pretending to be an HTTPS connection. Hence, if the socket tunnel server is set to be listening on HTTPS port (443) on an open Internet, socket tunnel client should be able to relay data and communicate with the server without any issues and firewall interruptions.
- On the server side, the socket tunnel server should be listening to a specific port and relaying any incoming data to a predefined local port or to any third party service (target service in Figure 2).

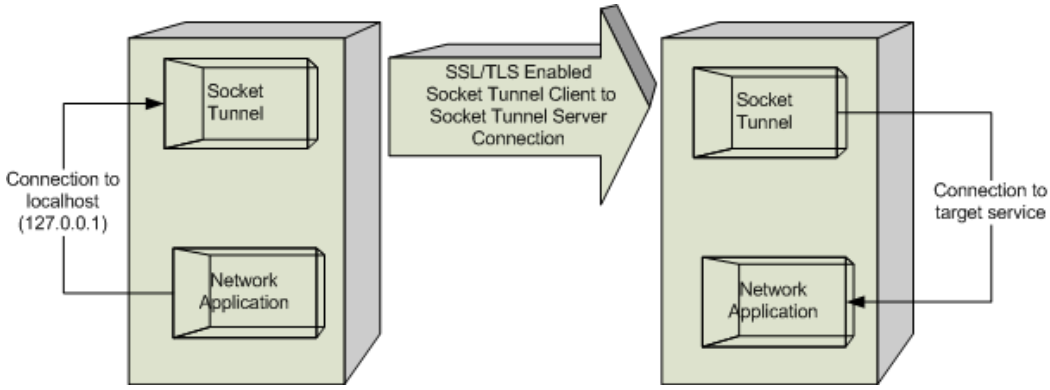

Figure 2: Socket Tunnel Process Communications

- Since to provide this functionality the client and the server application do not need an interface, you are ***not*** required to create a frame for your application.
- However, since it would be easier monitor active connections and the current status of the sockets that are being served, it would be really helpful to have a system tray icon for the application. Accordingly, you are required to create a small icon for system tray that shows the current activity with its changing color (https://docs.oracle.com/javase/tutorial/uiswing/misc/systemtray.html).
- The icon should be clickable and contain a menu. In the system tray icon menu, you are required to have an *About* item which will pop up a message box with the developer student's name, surname and number.
- The same system icon menu should also contain an Exit item which by selecting this item the socket tunnel service should terminate.

From the application logic point of view every node should satisfy the following requirements:

- Both peers – socket tunnel client and server – are actually acting as both client and server. The client peer is required to accept incoming local communication (acting as server) and relaying that data by encrypting it to the socket tunnel server. On the other hand, the socket tunnel server is required to accept incoming communication from the client, but also relay that data to local or remote service (acting as a client).
- To achieve this, both clients and server peers need a configuration file. This file is would contain essentially the services provided by each socket tunnel peer:

```
[Service Name]
client = yes/no
ListenPort = 3500
DestionationIP = 192.168.0.2
DestinationPort = 443
Proto = TCP/UDP
Key = /home/user/xyz.pem
```

- All communication between the tunnel nodes – client and server - should be encrypted using SSL/TLS.
- There will be no need for low level networking programming; hence all implementation could be done using Java. No C or C++ is required to implement this service.
- Java has SSL Socket class which can be used to create the encrypted communication (*javax.net.ssl.SSLSocket*)
- In the case of UDP relay, your application will have to stream them within TCP packets by adding sequence numbers so that they can be sorted/ordered on the server and sent out to the target service/port in their original order.
- [BONUS] In order to make use of the SSL Socket, you will be required create a Keystore and import your key into it. You can automate this process so that the application can directly access the PEM file and make the required conversion and import it to the Java keystore.
- [BONUS]. You can create a mini editor which will enable the user to add/edit the configuration file, mentioned above, which will set the service port numbers and destination addressed etc. The editor should be accessible from the system tray icon.

Submit your project source code in a zip file, which has your student number as name, using COADSYS by the end of Wednesday, January 6th, 2021. All submitted source files will be check for plagiarism - among classmates and with any existing open source code available on the Internet. Furthermore, all students will be required to demonstrate their work for 15 minutes. DO NOT submit somebody else's work.