Lab 10: TLS

50.020 Security

Hand-out: December 1
Hand-in: December 8, 11am

1 Objective

By the end of this lab, you should be able to:

- Create simple key-management system
- Create certificate for TLS/SSL connection
- Wrap socket using TLS/SSL
- Use Wireshark to capture packets transmission

2 Part 0: Do this before you come to class

- Read Python SSL module: https://docs.python.org/2.6/library/ssl.html

3 Part I: Key Management System Client-Server

- We are going to implement a simple protocol for key management system similar to Kerberos. If A wants to send a secure message to B, A would request the shared key from a trusted server. The server would respond by generating the shared key between A and B, and encrypt them for both A and B. The protocol can be written as follows.

$$A \rightarrow S \quad : \quad A, B \tag{1}$$
$$S \rightarrow A \quad : \quad \{T_S, L, K_{AB}, B, \{T_S, L, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \tag{2}$$
$$A \rightarrow B \quad : \quad \{T_S, L, K_{AB}, A\}_{K_{BS}}, \{A, T_A\}_{K_{AB}} \tag{3}$$
$$B \rightarrow A \quad : \quad \{T_A + 1\}_{K_{AB}} \tag{4}$$

  The above protocol means that Alice asks the server for a shared key with Bob. The server then creates the ticket $\{\{T_S, L, K_{AB}, A\}_{K_{BS}}\}$ containing the shared key K_{AB} and given to Alice to use. Alice also gets a copy of the key in a form readable by her encrypted under K_{AS}. She now sends the ticket and verifies the ticket by sending a timestamp T_A to Bob. Bob responds by sending back the timestamp incremented by 1. This means that Bob can decrypt the ticket correctly and extract the key K_{AB}.

- Create studentserver.py:

- Create a socket connection using `socket.socket()`.
- Bind socket to localhost with port 10023 or any other unused port number.
- Listen to the port using `socket.listen()`.
- Create a loop to accept socket using `socket.accept()`.
- Complete `deal_with_client`:
  - Prompt user with a menu: 0) Exit 1) Create user account, 2) Request Keys. Use `socket.socket.send(s)` to send the prompt.
  - Read reply from client using `socket.socket.recv(buffsize)`.
    - If client send '0', then server will reply with a string '%' to quit.
    - If client send '1', then create a new user account. Request for a username and password. Ask client to enter the password one more time. Check whether username is already in a file called 'passwd'. If not, create username and hash password using md5. Everytime a new user is created, generate an 80-bit shared key between the client and the server. Store the new username, hashed password, and shared key in a file called 'passwd'. Send the shared key to the client in PEM format. Send '%' for client to quit.
    - If client send '2', then create shared keys for client and destination user. Ask client for its username and password. The client must be found in the file 'passwd'. Prompt for the destination username. Find the usernames in the file 'passwd'. If found, generate a shared key between the client and the destination user. Send the following data back to the client:

      $$\{T_S, L, K_{AB}, B, \{T_S, L, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}. \quad (5)$$

      Where, T_S is the current timestamp, L is the lifetime. K_{AB} is the shared key between the client (A) and the destination user (B), K_{BS} is the shared key between the destination username and the server, K_{AS} is the shared key between the client and the server. Use '$$' to separate between different fields in the message. Note that the shared key is encrypted two times, one is for the destination user using the key K_{BS}, and the other one is for the client using the key K_{AS}. Send the message in PEM format. Use PRESENT in ECB mode to encrypt.
    - If client send others, prompt the menu again.

- Create studentclient.py:
  - Use `localhost` with port 10023 (or any other port number) as server to connect to.
  - Create a socket connection using `socket.socket()`.
  - Connect to your `server.py` using `socket.socket.connect()`.
    - Display the common name, email, and date for the valid certificate of the server.
    - Read prompt from server using `socket.socket.recv(buffsize)`.
    - Display prompt, and if the last character is '%', then exit and terminate the program. If not exit, then read from keyboard.
    - If the server is sending the requested shared key, store it into a file.
    - Send the characters read from keyboard to server.

- Create openkey.py that enables you to do the following:
  - Read shared key between A and server from a file.
  - Read received encrypted message from the server containing the shared key.
  - Decrypt message from the server using PRESENT in ECB mode with the shared key between A and server.
  - Parse the message to obtain: 1) timestamp, 2) lifetime, 3) shared key between A and B, 4) B username, 5) encrypted message to be send to B.
  - Read encrypted message sent by A. The message is encrypted by the shared key between B and server.
  - Parse the message to obtain: 1) timestamp, 2) lifetime, 3) shared key between A and B, 5) A username.

4 Part II: Sniffing Packet using Wireshark

- Run wireshark from the terminal:

```
$ sudo wireshark
```

- Run studentserver.py in one terminal:

```
$ python studentserver.py
```

- Run studentclient.py in another terminal:

```
$ python studentclient.py
```

- Use studentclient.py to request key. Go to wireshark capture to find the username entered and the password.
- Refer to appendix for some screenshot for wireshark capture.

5 Part III: Creating Certificate X509

- Use your pyrsa.py from Lab 9 to generate a new RSA private key in PEM format.
- Create a certificate request from your private key using OpenSSL. Use your student ID as your *Common Name*, and key in your student email address.

```
$ openssl req -new -key yourprivkey.pem -out req.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
```

```
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:SG
State or Province Name (full name) [Some-State]:Singapore
Locality Name (eg, city) []:Singapore
Organization Name (eg, company) [Internet Widgits Pty Ltd]:XXX
Organizational Unit Name (eg, section) []:YYY
Common Name (e.g. server FQDN or YOUR name) []:1000001
Email Address []:1000001@sutd.edu.sg

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

- Submit to eDimension for CA to sign your certificate by Wednesday 12pm. TA will notify you once the certificate is ready for you to download from eDimension.

6 Part IV: TLS/SSL

- We are going to wrap the socket connection using SSL.

```
import ssl
```

- Inside studentserver.py and studentclient.py:
  - Wrap the accepted socket using ssl with `ssl.wrap_socket()`. For server side, set `server_side` to True, and specify the private key you have generated as well as the signed certificate from CA. For client side, specify the location of CA certificate, and set `cert_reqs` to `ssl.CERT_REQUIRED`.
  - Change `socket.socket.recv(buffsize)` to `ssl.SSLSocket.read()`.
  - Change `socket.socket.send(s)` to `ssl.SSLSocket.write(s)`.
- Try to capture the data transferred using Wireshark.

7 Checkoff

- Show the tls/ssl data sniffed by wireshark.
- How would you verify the certificate of the root CA?
- Explain what's the use of certificate in this protocol.
- How can the client check on the identity of the server?
- How does server authenticate the users requesting for keys?
- What's the purpose of timestamp and lifetime?

8 Test Scenario

- Beginning Prompt of client.py:

```
$ python studentclient.py
Connecting to:
commonName =  1000001
email =  1000001@sutd.edu.sg
certificate valid until =  Nov 12 06:56:44 2015 GMT
=====================================================

0) Exit
1) Create account
2) Request Key

>
```

- If user enter '0':

```
0) Exit
1) Create account
2) Request Key

> 0

Quitting
```

- Create first user, enter '1' to create new user:

```
0) Exit
1) Create account
2) Request Key

> 1
Enter username:
> 1000003
Enter password:
> 1234
Enter password one more time:
> 1234
MHZ2ntBUCYNxeQ==
Quitting
```

  Copy the shared key in PEM format to a text file.

- Create second user:

```
0) Exit
1) Create account
2) Request Key

> 1
Enter username:
> 1000004
Enter password:
> 5678
Enter password one more time:
> 5678
tQJAjThb96LfDg==
Quitting
```

- Request key between 1000003 and 1000004 from server:

```
0) Exit
1) Create account
2) Request Key

> 2
Enter your username:
> 1000003
Enter password:
> 1234
Enter destination username:
> 1000004
Receiving key at: receivedkey.txt

Quitting
```

- Running openkey.py to parse receivedkey.txt:

```
========================================
Timestamp in second:  1416972723.92
Lifetime in second:  86400
Shared key with B in PEM:  t2jOSf8LlanDGQ==
B username:  1000004
========================================
Timestamp in second:  1416972723.92
Lifetime in second:  86400
Shared key with A in PEM:  t2jOSf8LlanDGQ==
A username:  1000003
```

9 Appendix: Screenshot for Wireshark capture

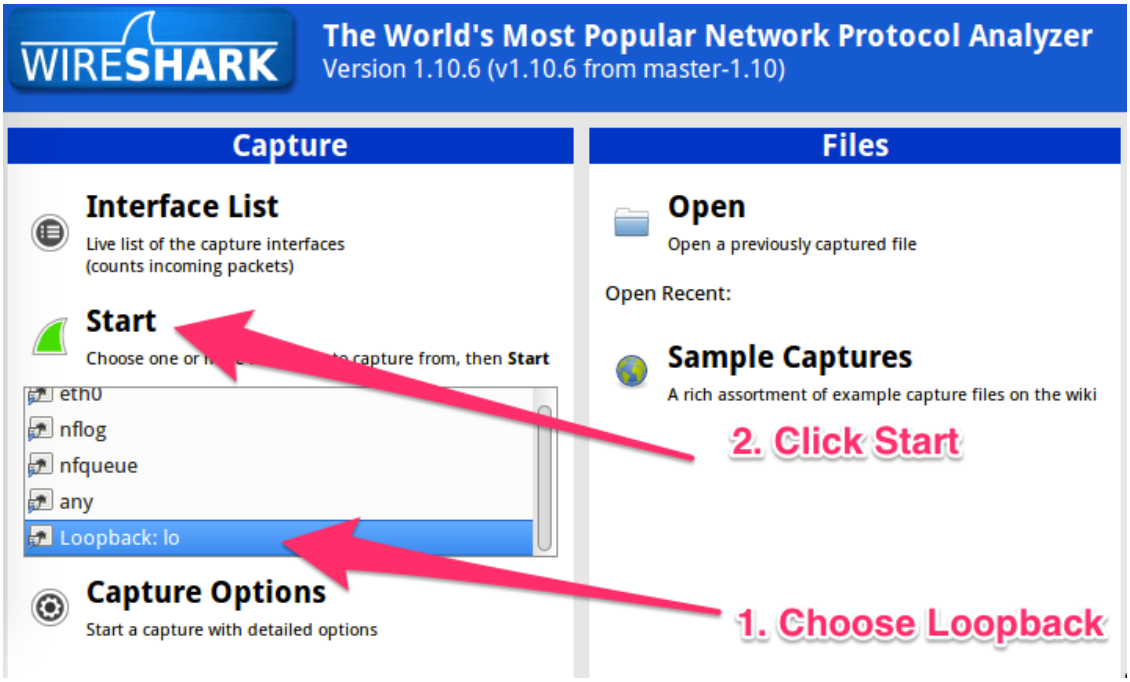

Figure 1: Start capturing localhost in wireshark.

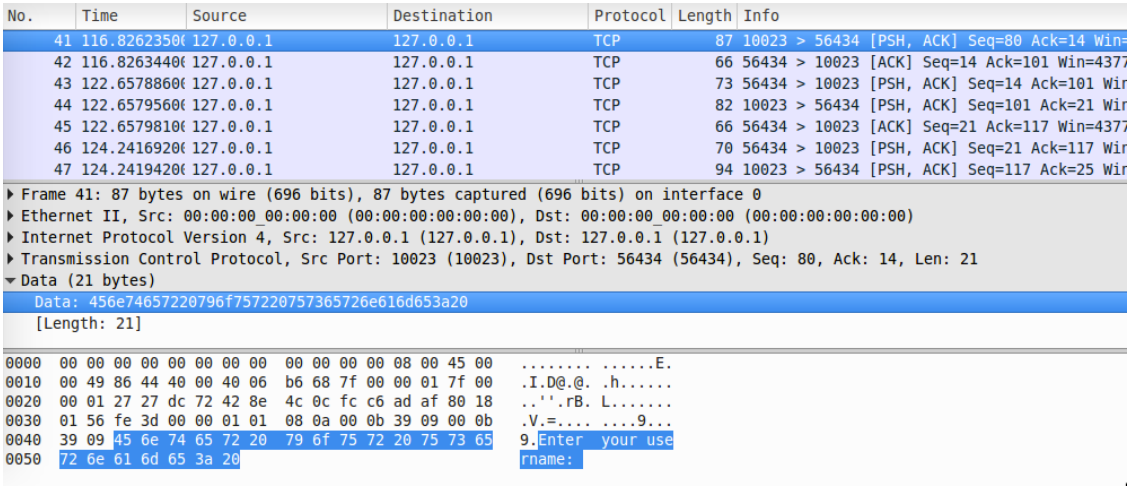

Figure 2: Captured enter username prompt.

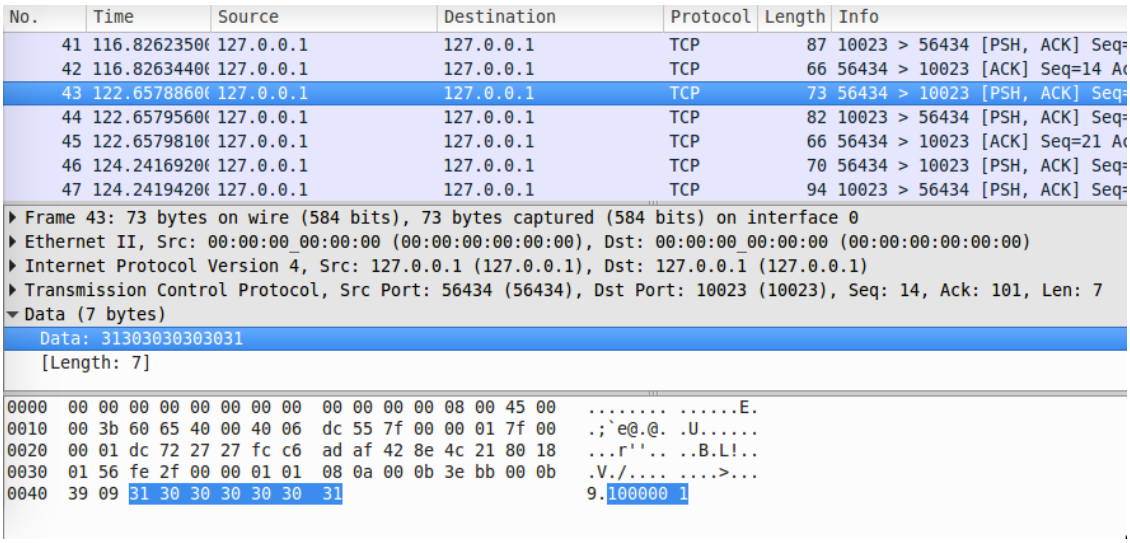

Figure 3: Captured username entered.

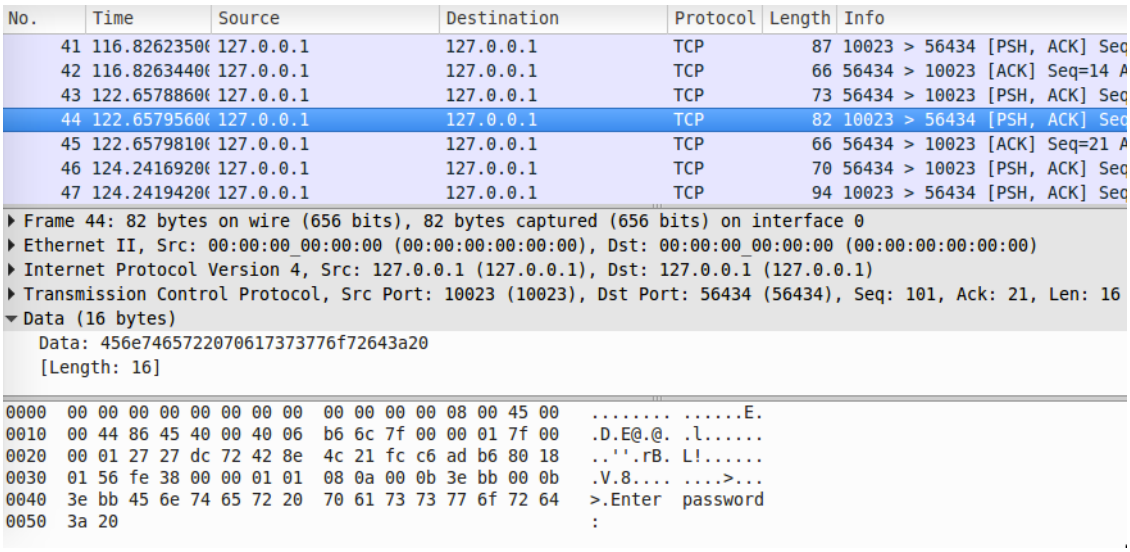

Figure 4: Captured enter password prompt.

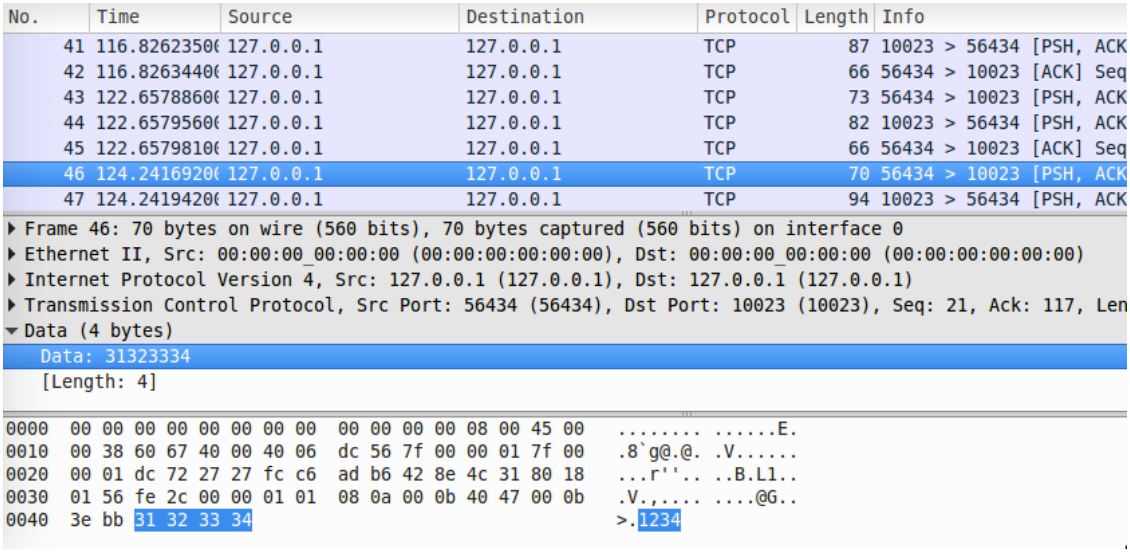

Figure 5: Captured password entered.