CS 572: Network Security **Fall 2019**

Project: A Secure Client-Server System

Assigned: Nov. 19, 2019 *Due: Dec. 6, 2019*

1 General Description

In this project, you will implement a secure client-server system using a simplied version of the Kerberos protocol. There are two key pieces to this project: (i) authentication of the client and server using a simplified Kerberos protocol; and (ii) file transfer from server to client, with transmission encrypted by the server.

2 The header file

We will provide you with a header file `packet_header.h` that contains the format information of all the packets, and defines the types of all the fields. You should include this header file in all your C programs, and carefully read the information present in the comments. There is also a README file that contains other important implementation details. These two files define all details you need to complete the assignment, and the instructions will guide your design. Please download the file `project.tar` from Canvas.

3 Client – Server Application

3.1 Requirements

There are three pieces of code you have to implement in three different files – a client, an authentication server (AS) and a server (AP). In this handout, we will give you the high level details of the protocol. For detailed packet type information, please refer the `packet_header.h` file and README file provided as part of the assignment.

3.2 Authentication

In the description that follows, K_c and K_s are secret keys shared by the Authentication Server (AS) with the client and the server (AP), respectively. $K_{c,s}$ is a secret key generated by the Authentication Server, and shared between the client and the server. Furthermore, the packet types that contain the information for each step of the authentication protocol will be presented with the description below (Refer to the figure):

a. The client sends a message to the AS of the form (AS_REQ message):

$ClientID||ServerID||TimeStamp1$

b. The AS responds to the client with a message of the form (AS_REP message):

$E_{K_c}[K_{c,s}||ServerID||TimeStamp2||Lifetime||Tkt]$

c. The Tkt is of the form:

$E_{K_s}[K_{c,s}||ClientID||ServerID||TimeStamp2||Lifetime]$

d. The client sends the server a message of the form (AP_REQ message):

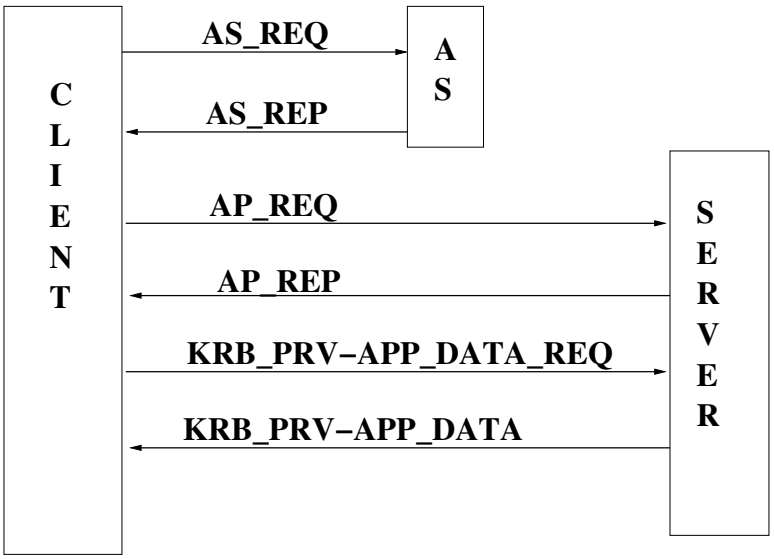
$Tkt||Authenticator$

e. The Authenticator is of the form:

$E_{K_{c,s}}[ClientID||TimeStamp3]$

f. The server returns a message to the client of the form (AP_REP message):

$E_{K_{c,s}}[TimeStamp3 + 1]$



3.3 Data Transmission:

The server transmits data to the client, encrypted with the secret key $K_{c,s}$ that was established between them during the authentication process. The packet types that contain the information for each step of the data transmission protocol will be presented with the description below (Refer to the figure). Please note that each of the packets mentioned below is encrypted and stored in a KRB_PRV packet type and then transmitted through the network:

a. The client transmits an APP_DATA_REQUEST packet to the server. The content sent from the client to the server is string "One sentence".

b. Once the server verifies the packet type to be a APP_DATA_REQUEST, it sends a sentence "Finally I got to send the data to the client. Succeed!" stored in a APP_DATA packet. This packet is then encrypted by AES-CBC using the secret key $K_{c,s}$, and using an IV of 0 (the IV is a 16-byte character array, all bytes of which are set to 0).

After the transmissions and print out the strings (both ciphertext and plantext), the client and the server should exit.

3.4 Error Packets

Two extra packets should be included as part of your implementation:

a. AS_ERR: Will be sent from the Authentication Server to the client if the message type is not AS_REQ. In case the client receives this message, it should gracefully exit the protocol. The Authentication Server should also exit the protocol.

b. AP_ERR: Will be sent by either the client (to the AP) or the AP (to the client) in two situations:

1. If the client received a wrong AP_REP message from the AP, it will return a AP_ERR to the AP. Recall that the AP_REP message contains the value (timestamp3 + 1), where timestamp3 was initially sent from the client to the AP.
2. If the AP receives a client id from the Authenticator that does not match with the client id that was sent with the ticket by the AS.

Notes:

a. You need to include the header file that we have provided in your C-files appropriately. You will also be able to access information regarding all packet types, and formats of fields there.

b. You may assume that the server handles only one client at a time and need not address issues associated with supporting multiple simultaneous clients. Further, there is no need for you to handle error recovery with UDP. However, in the event that a packet loss does occur, the code must exit gracefully.

c. For all encryptions, use AES-CBC. We provide sample code that implements the encrypt and decrypt functions. For all encryption/decryption with AES-CBC, you may assume an Initialization Vector (IV) of 0.

3.5 Command Line Arguments:

Your authentication server code must be executed from the command line as follows:

```
./authserver <authserverport> <clientID> <clientkey> <serverID> <serverkey>
```

<clientID> and <serverID> are strings not to exceed a length of 40 characters. <clientkey> is the secret key shared between the authentication server and the client, and <serverkey> is the secret key shared between the authentication server and the server. The keys must be represented as a string comprised only of ASCII characters.

The server code must be executed from the command line as follows:

```
./server <server port> <serverkey>
```

<authserverkey> is the secret key shared between the server and authentication server. The keys must be represented as a string comprised only of ASCII characters.

The client code must be executed from the command line as follows:

```
./client <authservername> <authserverport> <clientkey> <server name> <server port>
<clientID> <serverID>
```

The <clientkey> is the secret key shared between the client and the authentication server. The keys must be represented as a string comprised only of ASCII haracters. <clientID> and <serverID> are strings not to exceed a length of 40 characters. <authservername> is the host name of the machine running the authentication server (AS) and <servername> is the host name of the machine running the server (AP). The current sample code only work with IP address, instead of host names. If all three programs are running on the same machine, both <authservername> and <servername> should be `127.0.0.1`. If any of the arguments is invalid, your code should return an appropriate message to the user. Be sure to consider the case when the keys are invalid. Output messages: You must print to the screen (STDOUT) the following messages in case your application finishes correctly or terminate unexpectedly:

a. Print the message OK in case your application finishes correctly. This is the case when the server is able to completely send the sentence to the client. The client should print out both the encrypted sentence and decrypted sentence.

b. Print the message "ABORT" in case an error occurs, followed by a description of why the error occurs, Example erroneous situations are: (i) the client or server receives an unexpected packet; and (ii) any security issue occurs.

There could be other situations where it becomes necessary to abort the protocol. You should be able to detect this situation and print out the correct message.

4 Deliverables

The final submission includes the programs and related documents. You need to provide a makefile that will compile and link your programs. Comments are required. Also write a README file to give a general description about your programs and instructions to run them, and state any limitations of the implementation. Submit your programs and related documents to CS portal at `https://www.cs.uky.edu/csportal/login.php`

If necessary, you will also be required to give a demo to show your implementation.