Assignment #3

CPEN 442

17 October 2018

Jack Ma 29930147, Sawyer Payne 36625144, Valerian Ratu 18420142, Michelle Yang 11233146

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

I. Setup Instructions

A. Installation and Execution

Unzip the turnitin zip file.
Install Crypto for python.
Run the windows executable ui.py in the dist folder.
(Alternatively, run "python ui.py")

B. Set-Up and Execution

If stepping through the setup and communication process is desired: ensure to toggle the debug mode to "Debug Mode ON" at the bottom before continuing. This will activate step-by-step logging and the step "continue" button under the "Debug Mode" button.

The UI can be toggled between server mode and client mode on the top left corner. First toggle computer A to SERVER mode, enter a port number and a shared secret key in the respective text boxes, and click the "Start Server" button. Next toggle computer B to CLIENT mode, enter the server's IP address, the same port number entered in computer A, and the shared secret key in the respective text boxes, and click the button "Connect to Server".

Once client and server have been connected, messages can be sent by entering text into the "Data to be Sent" text boxes and clicking "SEND". Received data will appear in the "Data Received" text boxes.

To exit the program, either click the "QUIT" button at the bottom, or the 'X' in the top right corner of the application.

II. How the VPN Works

A. Discussion on How Data is Sent, Received, and Protected

Our program communicates through unsecured TCP sockets. First one side of the program runs in server mode, then the server creates a socket, binds the entered port number to it, and listens for connections from clients to establish connections. In client mode the client creates a socket and connects to the server with the entered IP and port number. Once it's connected to the client, the mutual authentication and session key establishment process begins. The program verifies the shared secret between client and server and terminates the connection if authentication fails.

Once authentication is completed, both client and server set up Sender and Receiver threads which have queues to send and receive messages. The messages are protected with the established session key and AES encryption. The encrypted message is sent through the insured TCP channel, and are decrypted after being received. Messages also use HMAC using SHA-256 and the shared secret key for an integrity check.

Individual components of messages are combined/separated when sent and received by using python Pickle. We initially separated components of messages with a ',' character, but quickly realized that Ra/Rb or any result of encryption, since sent as strings, could end up having a ',' character within them.

B. Mutual Authentication and Key Establishment Protocol

We used the mutual authentication protocol shown in Figure 1 to authenticate using the shared private key that is manually entered on both the client and server. On top of the authentication protocol, we also randomly generate a session key during the authentication process using Diffie-Hellman. All the communication after authentication is encrypted with this computed session key.

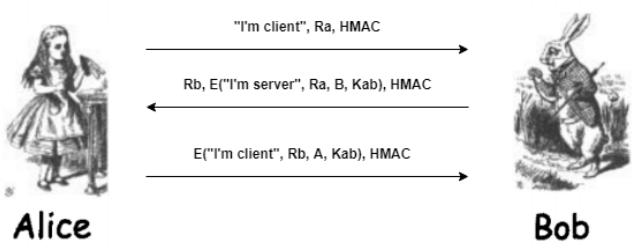

Figure 1. Protocol to provide mutual authentication, perfect forward secrecy, and establish a session key.

The values used in this protocol are as follows:

- Ra: random 64-bit nonce generated by the client

- Rb: random 64-bit nonce generated by the server
- a: random 256-bit number generated by the client
- b: random 256-bit number generated by the server
- g: hard-coded generator value for Diffie-Hellman
- p: hard-coded prime number for Diffie-Hellman
- A: is the result of g^a mod p
- B: is the result of g^b mod p
- Kab: the shared private key already established between client and server
- HMAC: is the HMAC of all previous bytes of the message, created using Kab and SHA-256
- E() is AES encryption

We chose to use this as it combined a strong mutual authentication protocol, which included perfect forward secrecy, with a Diffie-Hellman key exchange. The Diffie-Hellman exchange is protected from man in the middle attacks through the encryption using the previously established private key.

The protocol consists of 4 phases:

- Client challenge:
  - Generates Ra
  - Computes HMAC
  - Sends message in the form: "I'm client", Ra, HMAC
- Server check and response:
  - Verifies HMAC
  - Checks if the client string is “I’m client”
  - Generates b and computes B = g^b mod p
  - Generates Rb
  - Computes ciphertext = E("I'm server", Ra, B, **Kab**)
  - Computes HMAC = H(Rb, ciphertext, **Kab**)
  - Sends response in the form: Rb, E("I'm server", Ra, B, **Kab**), HMAC
- Client check and response
  - Verifies HMAC
  - Decrypts ciphertext
  - Checks if the server string is “I’m server”
  - Checks if Ra returned is the same as the Ra the client generated
  - Generates a and computes A = g^a mod p
  - Computes session key dh = a^B mod p
  - Sends response in the form: E("I'm client", Rb, A, Kab), HMAC
- Server check
  - Verifies HMAC
  - Decrypts ciphertext
  - Checks if the client string is “I’m client”
  - Checks if Rb returned is the same as the Rb the server generated
  - Computes session key dh = b^A mod p

The server and client are now mutually authenticated and have established a session key.

C. Deriving Encryption and Integrity Keys from Shared Value

The session key that has been established after authentication is now used both as the encryption key for AES and as the integrity protection key for the HMAC with SHA-256.

D. Real World Requirements of VPN

Implementation of this VPN as a real-world product would use the same algorithms, ciphers, and hash functions (Diffie-Hellman, AES, HMAC, and SHA256) for large messages like emails. However, in the case of severely resource constrained devices, or extremely high data rate systems, a steam cipher may be more appropriate.

The AES block cipher may use a larger key than the one we’ve implemented (128-bits) depending on device resources, and time (longer key means more number of rounds).

For Diffie-Hellman key exchange, the use of a prime modulus p with a size of 1024-bits has depreciated in level of security. It is currently recommended to use at least a size of 2048-bits. The a and b values generated by the client and server are recommended to be at least 256-bits.

For HMAC the encryption key size does not impact security as long as it is larger than the hashed output length. Extra length will not considerably increase security, but a longer key is advisable if the randomness of the key generation is weak.

E. Language of the VPN

We decided to use Python 3 as the language with which to develop our VPN. It had some useful tools built in to use AES and SHA-256. As well, we were familiar with socket programming using Python. The VPN software consists of ____ lines and is a ____ executable.

1) Modules

a) Sender

Inputs: connection socket, sender queue
Outputs: Encrypted messages to connection socket from sender queue, or error
The Sender module is a thread that abstracts outgoing socket communication in the client and the server. Both client and server have their own respective Senders which take outgoing messages off the sending queues, encrypts them, and sends them through the connection socket.

b) Receiver

Inputs: connection socket, receiver queue
Outputs: Decrypted messages from connection socket into receiver queue, or error
The Receiver module is a thread that abstracts incoming socket communication in the client and the server. Both client and server have their own respective Receivers which take incoming messages from the connection socket, decrypts them, and puts them into the receiver queue.

c) Authenticate

Input: shared secret key, receiver queue, sender queue, mode, Diffie-Hellman value

Output: successful authentication with session key, or authentication error
The Authenticate module authenticates and generates a session key between the client and server described in section II B.

d) Encrypt

Input: plaintext, session key
Output: ciphertext
Encrypts plaintext using an AES block cipher. To create the AES block cipher, the AES key is generated by hashing the session key using SHA256 to create a 128-bit key. Next the initialization vector of 16 randomly generated bytes is created, and the AES block cipher is set to CBC (cipher block chaining mode). The plaintext is then padded if necessary (to be a multiple of 16 bytes), encrypted, and appended to the initialization vector to generate the ciphertext. Finally, for message integrity, the ciphertext HMAC is generated and appended to the ciphertext.

e) Decrypt

Input: ciphertext, session key
Output: plaintext
Decrypts ciphertext using an AES block cipher. To create the AES block cipher, the AES key is generated by hashing the session key using SHA256 to create a 128-bit key. Next the initialization vector of 16 randomly generated bytes is taken from the first 16 bytes of the ciphertext, and the AES block cipher is set to CBC (cipher block chaining mode). This creates the same AES block cipher that was used to encrypt the message. Next, the remaining ciphertext after the initialization vector is decrypted using the AES block cipher and any padding is removed to generate the plaintext. Finally, integrity is checked by verifying the HMAC.

f) Get HMAC

Input: message, key
Output: hashed message authentication code
Generates HMAC using the SHA256 Hash Algorithm

g) Verify HMAC

Input: message, HMAC, key
Output: True if HMAC matches the message HMAC, else False
Generates the messages expected HMAC and compares it to the received HMAC. Uses SHA256 Hash Algorithm.

2) Architectural Components

a) User Interface (UI):

Tkinter is used as a Python binding to the Tk GUI Toolkit. This is Python's standard GUI package. The UI displays all the required buttons and textboxes and allows the user to toggle, and input data to set up the program. The UI also acts as the main application. It generates Sender and Receiver queues, starts Sender, Receiver, Server Listener, and Text Handler threads, starts up the client/server, initiates authentication, and writes to/ reads from the Sender and Receiver queues for sending and receiving messages.
For client mode: creates a socket and initiates TCP connection to the given IP address and given port number.
For server mode: creates a socket with given port number and starts Server Listener.

b) Server Listener

Thread generated for the Server to handle socket listening of incoming TCP connections from clients to the port specified in the user interface. It is configured to handle one client at a time and then stop listening. This prevents the UI from freezing while the server listens for connections.

c) Text Handler

Implements logging to write to UI and log each step during the setup and communication process. This is enabled when the program is in debug mode.

REFERENCES

Mark Lutz., *Python Pocket Reference, 4th Edition*. O'Reilly Media, Inc., 2009.