Assignment #3

EECE 412

October 15

FinalLy, jbauer, maj, maleficent
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

I. Installation Instruction

1. *Requirements*

a. Windows computer to run client and server
b. Access to Github to download repository

2. *Installation*

The program has been compiled to an executable file located in the software repository located here: [https://github.com/fgfl/EECE412_VPN]. Once the repository has been downloaded to your computer navigate to the file named VPN_gui.exe located here https://github.com/fgfl/EECE412_VPN/blob/master/dist/VPN_gui.exe and execute it. Be sure not to remove VPN_gui.exe from the folder it is in as it has dependencies on the files in it. No installation is required.

3. *Operation*

Once the repository has been downloaded execute VPN_gui.exe. You will first be prompted for the shared secret and need to tick a checkbox to either server mode or client mode. IMPORTANT: Make sure to start the VPN server first or the client will not be able to connect.

4. *Server Mode*

In server mode you only need to input which port you would like to run the server on and then connect. The server will wait until the client has connected before opening the messaging screen and no messages will be send until a session key has been negotiated.

5. *Client Mode*

You will be prompted for both the port and ip you wish to connect to. The ip address of the server can be found by running ‘ipconfig’ in the command prompt of the computer hosting the server.

6. *Debug Mode*

By checking the debug checkbox you enter debug mode in either client or server mode. You can then step through the messages being sent by using the Step button. Entering debug on both the client and server will allow you to step through all communications sent.

7. *Switching Modes*

To switch modes you must close the client or server and restart the application

II. VPN Description

1. Sending/receiving and protecting the data

The data is encrypted with AES CFB mode using the negotiated session key before it is sent and it is decrypted using the same method after it is received. AES is secure and it is used widely in the industry. In CFB mode, data is XORed with a value generated from the previous message history. Since the data does not pass the AES engine, its latency is minimized. CFB works similar to a stream cipher, hence the data can be processes as bits or bytes. In addition, the size of the entered plaintext can vary.

In order to check the integrity of the message, in addition to AES encryption, the message is hashed in the *encrypt* method using SHA256 hash function. The decrypt method has the ciphertext and the integrity hash digest as parameters. The ciphertext is then decrypted and it is hashed again. The result is compared to the inputted hash digest to verify the integrity of the message.

2. Mutual authentication and key establishment protocols

Combination of Diffie-Hellman protocol and Challenge-response protocol was implemented to provide both mutual authentication and Perfect Forward Secrecy (PFS) (See Diagram I). These protocols were chosen since they allow for efficient exchange of keys, provide PFS, prevent eavesdropping and are easy to implement.

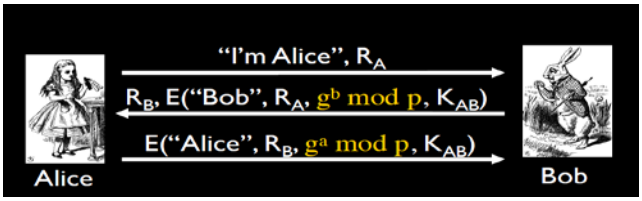
In the VPN system, the client and server both have knowledge of a shared secret key. The key is used in the initial Diffie-Hellman key exchange protocol to create the first session key. Both sides use the implemented *SessionKeyNegotiator* class for the key exchange. In the *SessionKeyNegotiator* class a *generator* number and a large *prime* number is defined. Both sides generate a random 512 bit *a* and *b*, and use them to generate public keys using $g^a \bmod p$ and $g^b \bmod p$. Once they public keys have been transmitted each side calculates the new session key using $g^{ab} \bmod p$.

To provide mutual authentication *identifiers* are set (e.g. Bob and Alice) and they are verified during the negotiation process. Lastly, to prevent man-in-the-middle attack, challenge-response protocol is implemented. A challenge is randomly generated, a response is created using the shared secret key and it is then validated in the negotiation process.

3. Encryption and integrity-protection keys

The Diffie-Hellman key exchange algorithm is implemented to use a shared secret value to negotiate an initial session key. Immediately, when the client connects to the server, the shared secret key is used to negotiate a session key. The shared secret key is then discarded from memory. A new session key is negotiated every 10 seconds in the same way, using the existing session key and the Diffie-Hellman key exchange. Every time a new session key is created, the old one is discarded; ensuring that even if the original shared secret were recovered none of the messages could be decrypted.

Data integrity is achieved using the SHA256 cryptographic hashing algorithm. For every message that is encrypted, a corresponding hash is created and transmitted before the message. When the encrypted message is received, the hash of the decrypted contents is compared with the received integrity hash. If the two hashes do not match, then it is determined that the message did not maintain integrity during transmission.



4. Real world implementations

According to Cisco the cryptographic algorithms used in this VPN, AES in CFB mode and SHA 256, are sufficiently secure and provide 128 bits and 198 bits of security, respectively [1]. SHA 256 is classified as a next generation security algorithm, and AES is widely deployed today by companies such as Apple. According to this article summarizing the ANSI X9.42 specification the sizes of p for 128-bit security should be at least 2084 bits long and q and t should be 224 bits [2].

5. Software language and size of the program

The Secure Vpn program is written entirely in Python and makes use of one external library, Pycrypto, to perform SHA256 cryptographic hashing and AES encryption. Excluding the external library the program consists of 1120 lines of code and the executable is 37 KB in size. The major modules of the code are as follows:

- SecureVpnCrypter
  - Provides encryption and decryption services for the Secure Vpn. Requires a shared secret to be set before encryption or decryption can take place. The shared secret can then be updated as new session keys are negotiated.
  - Important Methods:
    - encrypt
      - Takes a message to encrypt as a parameter and returns both a hash of the plaintext message, hashed using SHA256, and an encrypted message encrypted using AES 256 in CFB mode.
      - A new random initialization vector is generated for each message that is encrypted.
    - decrypt
      - Takes an encrypted message and an integrity hash as inputs and returns the decrypted message and True, if the message hash matches the integrity hash, or False if the integrity hash is not matched.
- SessionKeyNegotiator
  - Provides all functionality needed to negotiate session keys using Diffie-Hellman. This includes creating and parsing

challenges, creating challenge response messages, and the calculating of the session key.
    - Important Methods:
        - get_public_key
            - Generates a new private key and calculates the corresponding public key, returning it to the caller.
        - get_session_key
            - Takes a negotiation response as a parameter and calculates the corresponding session key using its own private key.
        - set_challenge
            - Generates a new random challenge for the negotiator to use.
        - create_challenge_response_message
            - Takes a challenge and the shared secret as parameters and creates a response to the diffie hellman negotiation which was sent with the corresponding challenge.
        - validate_response
            - Takes a negotiation response and shared secret as parameters and determines whether the response is valid (i.e. the response contains the challenge that was sent and the shared secret value)
        - record_challenge
            - Records a challenge sent in a diffie hellman exchange to be used in the negotiation.
- MessageManager
    - Formats and parses all messages that are sent and received. This class includes definitions for message flags which determine which type of message is being received. In addition, this class appends delimiters necessary for communication.
    - Valid message types are: CHALLENGE, INTEGRITY_HASH, NEGOTIATION_MESSAGE, NEGOTIATION_CHALLENGE, NEGOTIATION_CONFIRMATION_MESSAGE, NEGOTIATION_INITIALIZATION_MESSAGE, and DATA_MESSAGE.
    - Important Methods:
        - parse_message
            - Takes a message as input and parses it into an array of tuples containing the message type as well as the message contents.
        - create_xxxx
            - Takes the plain message to send as input and returns the specified type of message by appending message type and delimiters.
- SecureVpnBase
    - Provides all base functionality used by both the SecureVpnServer and SecureVpnClient. Its main functionality is providing methods to send the different types of messages created by MessageManager. This class takes advantage of the Python asynchat.async_chat class to provide asynchronous socket client functionality.
    - Important Methods:
        - send_xxxx
            - Creates, encrypts, and sends a specific type of message. This method will also send the integrity hash created during the message encryption to ensure message integrity on the receiving end.
        - initialize_negotiation and confirm_negotiation
            - Sends a message to the SecureVpnClient signaling that a key negotiation is about to take place. This allows both clients to go into negotiation mode and stop the sending of non-negotiation messages. After the negotiation is completed the Vpn sends a confirmation message so the client and server can leave negotiation mode.
        - toggle_step_through_mode
            - Toggles step through mode which will lock the sending of messages until the step button is pressed.
        - log, debug, debug_raw

        - These methods allow loggers to be passed into the Securevpn which can be used to output information. These are used to send information to the vpn UI.
- SecureVpnClient
    - The Secure VPN Client handles all client communication in the VPN. The client inherits from SecureVpnBase and adds the functionality of processing incoming messages and responding to them.
    - Important Methods:
        - process_message
            - This method takes the raw input from the socket and performs decryption, integrity checking, and message parsing. This method contains all logic for which tasks should be carried out when a specific message is received.
- SecureVpnServerHandler
    - The Secure VPN Server Handler handles all server communication between the client and the server. It adds functionality to SecureVpnBase to handle the processing of incoming messages. Additionally this class is responsible for initiation session key negotiations during VPN operation.
    - Important Methods:
        - process_message
            - Similar to the SecureVpnClient, this method takes the raw input from the socket and performs decryption, integrity checking, and message parsing. All logic for how different messages should be handled is contained here.
        - initiate_key_negotiation
            - Initiates the session key negotiation and re-initiates every 10 seconds for however long the VPN runs.
- SecureVpnServer
    - This class is only responsible for waiting for a connection from the client and forwarding commands to the SevureVpnServerHandler.

REFERENCE

[1] Cisco, "Next Generation Encryption," April 2014. [Online]. Available: http://www.cisco.com/web/about/security/intelligence/nextgen_crypto.html. [Accessed 15 October 2014].

[2] "Information Security," September 2013. [Online]. Available: http://security.stackexchange.com/questions/42558/is-there-a-key-length-definition-for-dh-or-dhe. [Accessed 14 October 2014].