Assignment #3 - Group 14

EECE 412

October 14th, 2014

<rlzh, Justin Timberlake, Jay Z, tubobo>
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

Project Github link: https://github.com/rlzh/EECE-412-VPN

I. Data Transition and Protection

The VPN created sends and receives data through a standard TCP connection. The presence of a VPN server instance is required before a VPN client instance can be created. When the VPN server instance is first created, it will listen on a user specified port number for an incoming client connection. When the VPN client instance is created, it will automatically attempt to connect to the user specified IP address and port number. If this process is successful, both instances will spawn a thread to maintain the TCP connection that has been established; thus, allowing data to be sent and received on both instances. All data sent and received across this connection are in the form of a byte array. When a new message string is sent, the sender will first convert the data into a byte array. Then, the server calculates and sends an integer metadata message notifying to the receiver the length of the incoming data. Following this, the receiver will prepare a byte array of appropriate length and receive the data sent from the sender. Upon receiving the entire byte array, the receiver will proceed to regenerate the original message string.

Protection of the data being sent and received is done through the use of mutual authentication, shared symmetric key encryption, signatures, and hash sum calculation. When the TCP connection is first established between server and client, a mutual authentication process is performed. This process helps to ensure the identities of the client and server to one another. The confidentiality of the data being sent across the network is achieved through encrypting the data with the symmetric key. Moreover, the integrity of the data is achieved through hash sum calculation. Finally, the authenticity of the data is guaranteed with the use of signatures. The cryptographic algorithms involved in data protection include AES-128, RSA-2048, MD5, and SHA-1.

II. Mutual Authentication Protocol

The mutual authentication protocol is achieved through a challenge-response process. Before the protocol begins, both instance are expected to establish a AES-128 symmetric key and a randomly generated RSA-2048 asymmetric key pair. The symmetric key is used to encrypt and decrypt all data passed through the connection. The asymmetric key pair generated here will be used to sign all data sent between the two instances. The protocol begins by having each instance send each other their respective public keys.

1. *Challenge*

Upon receiving the public key of the client, the server instance will continue the mutual authentication protocol as the challenger first. The challenge involves sending a session token (i.e. a randomly generated string of random length) to the responder as a challenge.

2. *Response*

The responder is expected to respond to this challenge by using the symmetric key to encrypt the value of the challenge concatenated with the shared secret value. Following this, the responder should return the encrypted result to the challenger for validation.

3. *Validation of Response*

Validation of the response begins by having the challenger decrypt the responder's response. If the decrypted result does not match the combination of the session token and the shared secret value, the authentication of the responder fails and the challenger disconnects from the responder. On the other hand, if the result matches, the challenger proceeds to notify the responder that they are authenticated.

After the first iteration through the above three steps, the challenger and responder switch roles to achieve mutual authentication.

The above mutual authentication protocol was chosen because it provides the simplest mechanism for ensuring authenticity of the VPN instance. For instance, brute force attacks on the protocol is difficult, as the session tokens and symmetric keys provide defense in depth. The protocol also effectively prevents replay attacks with the use of randomly generated session tokens every time a connection is established. Furthermore, the protocol is also able to fend off reflection attacks. The reason for this is due to the fact that any VPN server instance will only accept a maximum connection of one client at any given time; therefore, the bi-directional challenge-response process in the protocol cannot be reflected. It should also be noted all challenge and response data sent during the protocol is prepended with a signed hash sum of the data using each instance private keys; thus, providing data authenticity during the protocol.

III. Encryption and Integrity-Protection Key Generation

Encryption and decryption keys in the VPN instances are generated from the shared secret value by first generating a 128-bit MD5 hash of the shared secret value. The 128-bit MD5 hash is then used to generate a AES-128 symmetric key. In result, server and client instances will share identical encryption and decryption keys when given identical shared secret values.

The integrity protection keys in the VPN instances are in the form of RSA-2048 asymmetric keys. These keys are generated randomly in our implementation. Our reasoning behind this is based on the principle of least shared mechanism, as we feel using the shared secret value to also generate the integrity-protection keys did not provide any additional security benefits. The integrity-protection keys are utilized in the VPN instances through using the private key to sign SHA-1 hash sums of the encrypted data. This signed hash sum is then prepended to each message sent across the network. Receivers of each message will then extract the signed hash sum, unsign the hash sum, and recalculate the hash sum to ensure data integrity.

IV. Real-World Implementation

If the VPN instances were implemented as a real-world product for sale, one possible improvement would be to use 256 bit encryption keys to encrypt data. The reason our VPN does not utilize AES-256 currently is because Java is restricted to 128-bit encryption by default. However, removal of this restriction is possible with the use of the Java Cryptography Extension file. We would also modify the current asymmetric key pairs, used as integrity keys, to be RSA-4096. In turn, these changes will decrease the likelihood of successful brute force attacks significantly.

Depending on the number of allowed connections at a given instance, a real-world implementation of the VPN instances may also apply a modified mutual authentication protocol. In particular, the challenge-response process may be modified differ between the two directions.

V. Program Description

The software for this assignment is written in Java. The project contains 1498 lines of code, and the size of the runnable jar generated is around 19-20 MB depending on the operating system. The graphical user interface of the project was created with the utilization of Swing toolkit and Standard Widget Toolkit. Main components of the program include the VPNEntity class, VPNUserInterface class, and VPNManager class. Using the above main components, our program was modelled architecturally to follow MVC design conventions with the VPNEntity class as the model, VPNUserInterface class as the view, and VPNManager class as the controller. Our project also utilizes multi-threaded programming to prevent user interface freezes while background processing occurs. The tasks handled by individual threads include facilitation of TCP connection, mutual authentication, and message handling.

1. *VPNEntity*

   The VPNEntity class is largely responsible for all the calculations of the program. This includes generating keys, performing encryption, decryption, signing, and unsigning. At the end of each stage in the calculation, the VPNEntity class will return the computed result. This class also contains methods to receive and send messages through input and output streams of the network. The VPNEntity class is run on the main thread.

2. *VPNUserInterface*

   The VPNUserInterface class is designed to allow the user to control as well as view the actions of the VPNEntity instance. It notifies the VPNManager of user actions, and provides methods to log the results of those actions. Furthermore, the class also provides methods for classes executing in different threads to log their actions. The VPNUserInterface class is run on the main thread, and is the entry point to the program.

3. *VPNManager*

   The VPNManager class manages an instance of the VPNEntity class. This class is utilized to receive inputs from the user interface and modify the VPNEntity instance accordingly. It is also responsible for relaying the calculation results of the VPNEntity instance to the log in the user interface. The VPNManager class is also run on the main thread of the program.

4. *ConnectionSetupHelper*

   The ConnectionSetupHelper class is run following the creation of the VPNEntity instance. This class is responsible for establishing the TCP connection between VPNEntity instances. It is executed on a separate thread to prevent freezes in the user interface while server instance wait for client connection.

5. *AuthenticationHandler*

The AuthenticationHander class is executed immediately after a TCP connection is established. It is used to carry out all stages of the mutual authentication protocol. The AuthenticationHandler class is initialized in the ConnectionSetupHelper class. After initialization, the AuthenticationHandler class is executed on an individual thread to prevent user interface freeze while the mutual authentication protocol occurs

6. *MessageHandler*

The MessageHandler class is run following the mutual authentication protocol. This class acts as a listener for incoming messages on established TCP connection. The MessageHandler class is initialized in the AuthenticationHandler class. It also runs on an individual thread to prevent user interface freezing while it awaits the next message on the TCP connection.

Note: See “Installation Instructions.pdf” in Github repository for details on running the program