Assignment #3

EECE 412

October 15th

Nora Lynn, Heartbeat, BlueMonday, Demios111
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

I. Installation Instructions

In order to run the VPN, the following prerequisites need to installed on the system:

1. Python 2.7
2. Pip for python 2.7

To install python 2.7 and pip in Ubuntu:

```
$ sudo apt-get -y install python2.7 pip
```

All required non-standard python libraries are listed in requirements.txt in project root. They can all be installed by running the following:

```
$ sudo pip2 install -r requirements.txt
```

Once these prerequisites are met, the VPN can be run by running the following:

```
$ python2.7 assignment3.py
```

The radio buttons can then be used to select client and server modes on the respective computers, with the client requiring an ip address to connect to the server.

II. Data Transmission and Prtection

The data is sent via a standard TCP socket. Both clients maintain an internal state, expecting unencrypted information for the authorization and key exchange, and encrypted information following the handshake. The sensitive data is subsequently sent, prefixed with a header specific to our protocol.

The messages are first signed with a SHA-512 HMAC, using the symmetric key, which is appended to the body. Then the signed message is encrypted using AES-128 in CBC mode, using the session key, with the IV prepended. Finally, the encrypted message and IV are signed once again using a SHA-512 HMAC with the symmetric key. The reasoning for signing twice is to provide authentication and integrity whereas only signing once leaves the messages vulnerable to man-in-the-middle and replay attacks. AES with Galois/Counter Mode (GCM) would have instead been employed for this task, but it was not available in the stable libraries at the time of writing.

III. Mutual Authentication and Key establishment

A Diffie–Hellman–Merkle (DHM) key exchange is used for the session key establishment. A DHM key exchange is used because it can create a secure channel over an insecure connection. DHM relies upon the discrete-logarithm problem being computationally infeasible to solve by today's modern supercomputers in a reasonable amount of time[1]. After a secure channel is available, the pre-shared symmetric key is used by each party to perform a challenge-response to provide mutual authentication. Nonces are included in the mutual authentication to prevent replay attacks while HMAC is used to enforce the integrity of the message without leaking the ore-shared secret key.

The following steps are used to establish a key and for mutual authentication:

1. Client & Server: generate a secret integer, *a* and *b*, which is a random number from 2 to 2^{256}
2. Client - send hello message with includes a nonce which is a random integer between 2 and 2^{256}
3. Server - upon receiving the client's hello message the nonce that the client sent is stored. The server will now send an init message to the client. This will contain:
   a. A different random nonce
   b. A SHA-512 hash of the client's nonce and the shared secret value
   c. The value of g^a mod p, where g and p are constants for the program, g being relatively prime to p.

Client - after receiving the server's response, the client computes the session key, and verifies that the server has sent the appropriate hash authentication. If not, it will disconnect. If it is valid, it sends the following: A SHA-512 hash of the server's nonce and the shared secret value and the value of g^b mod p

Server - computes the session key, and verifies that the client has sent the appropriate hash authentication. If not, it will disconnect.

Key exchange and authentication are done at this point, if the connection is still open.

IV. Encryption and Integrity-Protection Keys

The encryption key for the session is derived using the Diffie–Hellman–Merkle key exchange. The encryption key is created by taking the first 32 bytes of the truncated shared secret value, from the DHM key exchange, and passing it to the AES algorithm. A random 16 byte cryptographically secure initialization vector (IV) is generated for each message.

The integrity protection key is derived from the pre-shared secret key by using a SHA-512 HMAC with the message data. The HMAC is appended to the message, to be verified by the other party upon reception, by performing the same operation again using the pre-shared secret key.

V. Real World VPN Security

Given the implicit trust and importance of a VPN's security, values and algorithms should be chosen with a very large margin of safety and be influenced by top industry standards. Following NIST and NSA guidelines for security, and choosing algorithms that would be easy to implement in hardware or software would create the most secure and versatile system.

A Diffie–Hellman–Merkle key exchange is used to generate a session key as it is can create a secure channel over an insecure connection without leaking any private information. In addition to the key exchange, AES-256 encryption should be used. The modulus size should be 300 digits long, with the shared secrets a,b at least 100 digits long to prevent an eavesdropper from being able to solve the discrete-logarithm problem that the DHM relies on for confidentiality[1]. AES in CBC mode is used to provide confidentiality while an SHA-256 HMAC is used for integrity. Using AES-256 bit for encryption and SHA-512 bit for HMAC would be sufficient for key sizes as it exceeds the current NSA and NIST standards for computer security and information privacy[2][3].

VI. Software Architecture, Modules, and Reasoning

This project was written in Python. Python was chosen due to it's simplicity, and numerous libraries for everything from cryptography to GUI design and networking. In total the project consists of 600 lines of code. Python also includes TK (a cross platform widget toolkit) bindings as part of its standard library.

The main modules of the project are the GUI, socket and cryptosystem.

The GUI provides an interface through which users can perform many actions such connect to each other, send messages, view received messages and set the VPN mode. As the name suggests the GUI provides an interface through which the user can pass along information and commands to the underlying modules. The inputs into the GUI come from users and from the socket. The GUI must receive input from the socket so that it can display incoming and outgoing messages and debug information as well as notify the user on connection status. The output from the GUI almost exclusively goes to the socket in the form of commands; whether it is a command to open a socket, send a message or step through to the next command.

The socket itself provides the means through which the two computers communicate. This module is in charge of handing the initialization of the connection, termination of a connection, events and debugging of the connection. The inputs into this module are commands and settings from the GUI and encrypted information from the cryptosystem. The outputs of this module are information on the status of the socket and information received and sent from the socket. Information is also passed to the cryptosystem to be hashed, encrypted or encrypted.

The cryptosystem is in charge of encrypting the information sent over the socket. Inputs into this system are the shared key, session key, and data to be signed/verified & encrypted/decrypted. The outputs are encrypted and signed cipher texts now ready to be sent over the socket or verified decrypted plaintext recovered from a ciphertext received over the socket. The cryptosystem is further broken down into crypto and authentication modules to easily separate concerns. They also handle the nuances of the respective algorithms employed like padding the message lengths, and removing padding.

References

[1] *Diffie Hellman Key Exchange*. Wikipedia. Web. 14 October 2014 < https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange>

[2] *SHA-2*. Wikipedia. Web. 14 October 2014 < https://en.wikipedia.org/wiki/SHA-2>

[3] *Advanced Encryption Standard*. Wikipedia. Web. 14 October 2014 < https://en.wikipedia.org/wiki/Advanced_Encryption_Standard>