Assignment #3

EECE 412

October 15, 2014

Adam Berg, Kevin Wagner, Jonathan Machel and Christopher Tan
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

Installation Instructions:

1. To install the VPN, download the code from https://github.com/adamjberg/VPN/tree/master on a computer with 64-bit Linux OS.
2. The computer will require the following libraries: libevent, libgtk3.0-dev, libssl-dev and OpenSSL v1.01f.
3. Run the make file with "make -f Makefile" which will compile the code and create an executable file called main.

VPN Instructions:
When running the executable, a GUI is presented where the server IP address, the server port number and shared key are inputs for the user. The port numbers and shared key must be exactly the same for both the client and the server for a successful connection. For the server, use the combo box to select the "Server" mode. If you are running in server mode you should use the server's external IP or localhost IP. When ready, press the "Start" button. For the client, use the combo box to select "Client" mode, and ensure that the server's IP address is entered correctly. When finished with the inputs, press the connect button to connect to the server. Make sure the server is started before the client tries to connect.

The GUI has a continue and auto-continue button. When the client and server are connected, pressing the continue button progresses to the next step of the authentication process. If this is used instead of the automatic progression, both client and server must alternate using the continue button in order to progress.

The auto-continue button allows the client or server to progress through the authentication process without the user's prompt to continue.

I. Problem #1

In our VPN design data is sent to and from the client and server using the linux sys/socket library. While this socket library exists on multiple operating systems, if we were using windows we would need to use another library such as winsock. Using the linux library we first create sockets which we use to send and receive data. Events are then made using the libevent library to listen for data that is waiting to be read. When those events are triggered, an event handler is called which reads the data that was sent. The sent messages are encrypted and decrypted using Blowfish encryption and later on encrypted using a session key which is determined using the protocol explained in problem 2.

II. Problem #2

difficult to perform a brute force attack on the encryptions.nore
For this assignments we chose to use the Diffie-Hellman key exchange to establish mutual authentication. We chose to use the Diffie-Hellman key exchange because it allows us to perform perfect forward secrecy. In order to prevent man in the middle attacks we encrypt the messages using a symmetric key that has been agreed upon by both client and server prior to authentication. The encryption is done using a Blowfish encryption function given in OpenSSL. We chose the Blowfish encryption function because contrary to many of the other encryption function we found it allowed for an input of variable length. It also takes more time to generate a key to be used for Blowfish encryption which means it would be more difficult to perform a brute force attack on the encryptions.

The authentication protocol first involves the client generating a nonce value (an arbitrary number used only once) and sending it to the server. The server then needs to generate its own nonce value as well as encrypt the clients nonce value and (g^b mod p) using a Blowfish encryption function and sends all of this to the client. The client then encrypts the nonce sent by the server as well as (g^a mod p) and sends all of this to the server. Now the server and the client both calculate (g^ab mod p) and use this key for future encryptions.

III. Problem #3

Initially the only shared secret value is one that has been agreed upon ahead of time by the client and server in order to be used as a key during authentication. As explained in problem 2, the shared secret key encrypts messages sent to and from the client and server in order to protect against man in the middle attack during authentication. After authentication has been completed, both the client and the server no longer have use for shared secret value used during authentication. Instead the client and server will have securely determined a shared secret session key using the Diffie-Hellman method which can be used from then on to communicate. The Diffie-Hellman method involves the server sending (g^b mod p) to the client and the client sending (g^a mod p) to the server. In both calculations ‘p’ is a relatively large prime number, ‘g’ is prime relative ‘p’ and ‘a’ and ‘b’ are private integers know only by one of the parties involved. After these values have been sent both client and server can then calculate (g^ab mod p); this value is then used as the session key and is shared between the client and server.

IV. Problem #4

There are several changes that would have to be made if this VPN were a real-world product for sale. From a usability perspective, there are UI and functionality modifications that would be required, such as removing the step-through “Continue” button and improving the user interface. One of the major changes we would make if this were a real-world VPN would be to use a more modern encryption function such as AES or Twofish (the direct successor to Blowfish). While Blowfish is less susceptible to brute force attacks because of the slow initial key setup, it is known to be susceptible to attacks on certain classes of keys. Another change we would need to make is to the modulus size. Currently we use the value of 11 for large prime number ‘p’ in the Diffie-Hellman method, as we were instructed to use a small value. However, this is unsecure for a real VPN as it is too easy to guess. Ideally we would use the largest number that did not impact performance of the program.

V. Problem #5s

operating systems.t only widely used but it worked on most

The executable of the program is about 65 KB, which was compiled from 9 files; 4 header files and 5 C files that include server code, client code, encryption code, helper function code and the GUI code. The entire project is just over 1,200 lines of code. The main GUI, server and client files have approximately 300 lines of code each as these are the main components of the VPN. The GUI server code is where the main function is located and it consists of creating GTK widgets, where each widget allows input for the server IP address, the port number, the shared key; the output displays authentication text, encrypted text and plain text messages. Lastly the GUI has a combo box input to select between client and server states. The GUI client code also creates GTK widgets for server IP address, port, and key inputs, and sets up the display for output of the messages. The encryption code and helper function code each have less than 100 lines of code since they only contain functions used in the main GUI, client and server.