Don Heer
Reto Galli
ECE 565 – Project
March, 14 2001

An Encrypted File Transfer Program

Contents

Introduction

For our final project, we decided to explore creating a file transfer program that can send both files in plain format, like most file transfer programs do, and in a packet encrypted method. With the rate that people are moving to the Internet for communications of everything from personal letters and shopping to larger transactions with banks, and credit agencies, the Internet becomes more and more appealing to the criminal element. This as always forces an improvement in security, one method of which is through encryption. This was our motivation behind or selection of this project. We decided on using a well-known encryption algorithm known as the Digital Encryption Standard (DES), and to exchange encryption keys between machines we used the Diffe-Hellman Key Exchange algorithm. Both of these methods are discussed more detailed in the Cryptography section.

Functionality

For our program we wanted to make a ‘test bed’ for future designs. To do this we wrote or program as a dual transceiver system, rather than a receiver and a transmitter program. This allows for either end of the connection to initiate commands to the either end. With little work, we could create an encrypted chat program, Telnet connection, or other dual mode programs. To do this the entire program has to work in a ‘non-blocking’ mode. This means, that we can not wait for any one event to occur, we need to instead check if an event occurs, and if it has not, we go on and check another event, this is sometimes called polling. To create this non-blocking design it was necessary to find a way that we could check the incoming data stream from the TCP/IP socket for the presence of data, without calling the ‘recv()’ c-function which is a blocking instruction. To do this we located an algorithm for checking any UNIX stream without incrementing its pointer. This was ideal for us because it allowed us to also check for data from the keyboard without blocking as well.

As for the communication packet structure, we chose a simple design that relies on the built in functionality of the UNIX socket functions. We simply transmit a single unsigned integer and 64 bytes of data as a single packet. The communications are best exemplified by a simple example.

Control(4)	Data(64)

Alice wants to send a file to Bob. First, she selects the non-encrypted file send option. She is then prompted for the file she wishes to send, if the file cannot be opened, the file transfer ends. If the file can be opened, she is prompted for the name of the file on the receiving side. A data packet with a control code corresponding to ‘file send’ and the filename of the destination in the data is then sent to Bob. Bob’s machine then checks the incoming control code and decides that Alice is trying to send a file to be named whatever is in the data section of the packet. Bob then tries to open the file if he succeeds, he replies to Alice that the file can be written, otherwise he replies that the file cannot be written and returns to ‘watch’ mode. If Alice receives that the file could not be opened, she closes her source file and returns to ‘watch’ mode as well, otherwise she begins to transmit the file packet by packet, placing the number of valid bytes in each packet in the control section of each packet. At the end of transmission each side pauses momentarily to allow for the user to review the results of the command, then the menu is redisplayed.

The encrypted file transfer works exactly the same with the difference that before the transmission starts the two stations exchange their Diffie-Hellmann public keys to create the session key. During the transmission the 64 byte data block is encrypted using DES. The receiver station saves the received data without decrypting it. The session key used for the transmission is saved on the receiving machine under the same name as the file but with a suffix ‘.key’. The received file can be decrypted using the decrypt function in the menu.

Cryptography

The program uses standardized cryptographic functionalities. Before the file transmission can take place the cryptographic key (session key) has to be exchanged between the two parties. This is done using the Diffie-Hellmann Key Exchange Protocol. This is a public key cryptography method that is widely used in data security. It works as follows (again an example with Alice and Bob):

- Both parties have a prime number q and a number a where $a < q$ and a is a primitive root of q. This numbers can be public.
- Alice selects randomly a private key $X_A < q$ and calculates its public key $Y_A = a^{XA} \bmod q$.
- Bob selects randomly a private key $X_B < q$ and calculates its public key $Y_B = a^{XB} \bmod q$.

- They exchange the public keys.
- Alice computes: $K = (Y_B)^{XA} \bmod q$
- Bob computes: $K = (Y_A)^{XB} \bmod q$
- The two K are identical and can be used as session key for the transmission. A prove for this relation can be found in [1].

The cryptographic library we are using in our program implements a 1024 bit version of this protocol. This means an intruder would have to solve a 1024 bit discrete logarithm to obtain the key. The discrete logarithm is a NP-complete mathematical problem. That means the computational power needed to solve the problem grows exponential with the size of the inputs. Cryptographic systems based on the discrete logarithm problem should use at least 512 bit keys to be secure with today's computational power. The time needed for the computation of the session key by the authorized parties takes some 100 milliseconds. This is feasible if it has to be done once per session.

After the session key is exchanged between the two parties the data stream is encrypted using the fast and widely used Digital Encryption Standard (DES). This encryption standard was developed in 1977 by the National Institute of Standards and Technology (NIST) based on an algorithm developed by IBM. The algorithm is a block encryption algorithm that encrypts 64 bit blocks. It is a Feistel Network that uses permutations, lookup tables and XOR functions. Until today no statistical weaknesses could be found in the algorithm. The only disadvantage of it is the relative short key length of 56 bit, which makes it vulnerable against brute force attacks. NIST still recommends using it, but they will replace it by a new standard soon. Although the algorithm is optimized for hardware realization is DES quite fast if executed on microprocessors.

To make a statistical attack on the encrypted data stream harder our program uses an 8-Bit Cipher Feedback (CFB) chaining mode to encrypt the data. That means that parts of the previous encrypted block are used to encrypt the current data block. With this technique two identical data blocks result in two different encrypted data blocks. A detailed description of this mode can be found in [1].

All cryptographic functions in our program are realized using the Crypto Toolkit Library from Cylink. This library can be used free for private and academic purposes. The Crypto Toolkit Library uses the BigNum Math Library, which provides the mathematical functions for numbers up to 1024 bit. This library is free available and can be used for private and academic purposes.

How to run the program

First unzip (*gunzip project.tar.gz*) and untar (*tar xvf project.tar*) the archive *project.tar.gz*. To compile the program run the makefile in the top directory of the structure without any parameter (*make*). To delete all compiled data use the same makefile with the parameter clean (*make clean*). The compilation process produces the file *trans.exe*. Run this program and follow the instructions. The program is tested on HP machines in the ENGR and the ECE network at OSU. Those are the only machines we can guarantee a correct functionality.

Conclusions

We showed with this project how basic cryptographic functions could be added to a simple network program. The code presented implements an easy file transfer program but could easily be modified to another application like a chat or a telnet program.

References

[1] William Stallings, *Cryptography and Network Security – Principles and Practice*, 2nd edition, 1998, Prentice-Hall