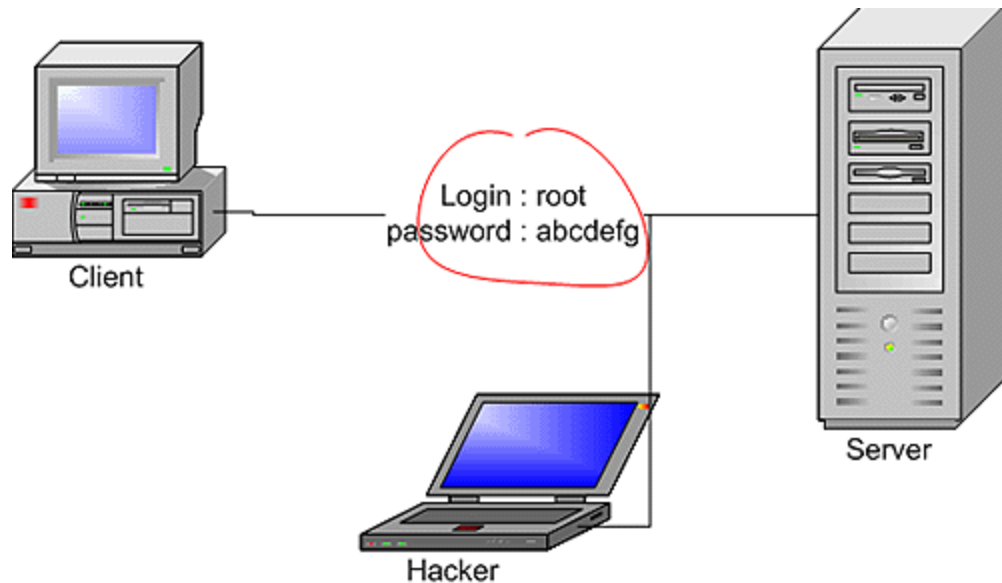


CS588: Final Project
(Implementing Encryption on Telnet using Symmetric Keys)



Submitted by:

Abhishek Kanchan
(akanch4@uic.edu)
Abhishek Singh Rathore
(aratho3@uic.edu)
Prashant Tripathi
(ptripa4@uic.edu)

Introduction

Telnet is a network protocol used on the Internet or LAN's to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection. User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the (TCP). Typically, it establishes a TCP connection at port number 23 where the information is exchanged between the server and client.

The main problem with Telnet is its inherently an insecure protocol. Telnet was created when network security was not as much of a concern as it is today. Both protocols were designed with simplicity, versatility and flexibility in mind, but definitely not security. In modern communication which is marred with frequent eavesdropping,

Motivation

Telnet suffers from some serious security flaws, the most critical of which are:

- **Clear-text transmission:** All communications are done in clear text, including usernames and passwords.
- **Weak client authentication:** Telnet authenticate users through usernames and passwords, which, time and time again, have proven to be unreliable authentication methods. There is no support for more advanced authentication methods such as public/private key, Kerberos or digital certificates.
- **No server authentication:** This means that users have no way to be sure that the host they are communicating with really is the FTP server and not an attacker impersonating the server.
- **Absence of data integrity:** Problem here is that, assuming the same scenario as above, anyone could alter and corrupt the data being transmitted between the server and the client without being noticed.

Encryption

In cryptography, encryption is the process of encoding messages (or information) in such a way that third parties cannot read it, but only authorized parties can. Encryption doesn't prevent hacking but it prevents the hacker from reading the data that is encrypted. In an encryption scheme, the message or information (referred to as plaintext) is encrypted using an encryption algorithm, turning it into an unreadable cipher text. This is usually done with the use of an encryption key, which specifies how the message is to be encoded. Any adversary that can see the cipher text should not be able to determine anything about the original message. An authorized party, however, is able to decode the ciphertext using a decryption algorithm, that usually requires a secret decryption key, that adversaries do not have access to. For technical reasons, an encryption scheme usually needs a key-generation algorithm to randomly produce keys. In this case a fixed key which has been hard-coded is being used.

The existing encryption algorithms can be classified into two main categories:

- Symmetric key algorithms
- Public key encryption algorithms

The current project implements only the symmetric key algorithms. The following algorithms are supported by the code:

- ✓ Blowfish
- ✓ DES
- ✓ Triple DES
- ✓ Threeway
- ✓ Ghost
- ✓ Safer-sk64
- ✓ Safer-sk128
- ✓ Cast-128

- ✓ Xtea
- ✓ RC2
- ✓ Twofish
- ✓ Cast-256
- ✓ Saferplus
- ✓ loki97
- ✓ Serpent
- ✓ AES
- ✓ rijndael-128
- ✓ rijndael-192
- ✓ rijndael-256
- ✓ Enigma
- ✓ RC4
- ✓ Wake

Symmetric key encryption scheme

Symmetric Ciphers Online allows you to encrypt or decrypt arbitrary message using several well known symmetric encryption algorithms such as AES, 3DES, or BLOWFISH.

Symmetric ciphers use the same (or very similar from the algorithmic point of view) keys for both encryption and decryption of a message. They are designed to be easily computable and able to process even large messages in real time. Symmetric ciphers can operate either in the block mode or in the stream mode. The Rijndael uses the block mode for encryption whereas some of the other algorithms can work in both modes.

Coding and testing platform

The coding has been done in GNU C and compiled through GCC 4.7 compiler for x86_64 core and runs on UNIX based machines. The code is essentially small, distributed across two C files:

- 1) **server.c**: This contains the telnet server implementation.
- 2) **client.c**: This contains the telnet client implementation.
- 3) **encrypt.c**: This has the encryption and decryption related functions.

There is a basic GNU Makefile which scripts the compilation and cleaning process. The final executable binaries have been given the name “server” and “client”.

Apart from this the common definitions and file includes are in the header files network.h and encrypt.h

The library MCRYPT for Linux has been used for encryption APIs and related data structures. Install the mcrypt library on both the client and server system before compiling and running the tests.

MCrypt:

MCrypt is a replacement for the old crypt() package and crypt(1) command, with extensions. It allows developers to use a wide range of encryption functions, without making drastic changes to their code. It allows users to encrypt files or data streams without having to be cryptographers.

The companion to MCrypt is Libmcrypt, which contains the actual encryption functions themselves, and provides a standardized mechanism for accessing them.

For running on Mac OSX (Darwin based systems)

- 1) Install the mcrypt library from the command prompt using any of the package installers for Mac OS like HomeBrew or MacPorts on both the client and the server machine.
- 2) Make sure that the mcrypt.h file is included in the same directory as the source code before compiling and running the code.

DESIGN DESCRIPTION

- The server creates a socket and waits for client to connect. The server in itself is a process running continuously.
- Client connects to the server at the specified socket.
- On successful connection, a child process is created by the server which provides the shell utility to the client.
- client has a layer of encryption, which encrypts everything that is send to the server.
- The server end has a decrypter which decodes the command passed by the client and performs the necessary action.

USAGE AND SAMPLE OUTPUTS

In order to compile the source, give a “make” which uses the Makefile to compile the code.

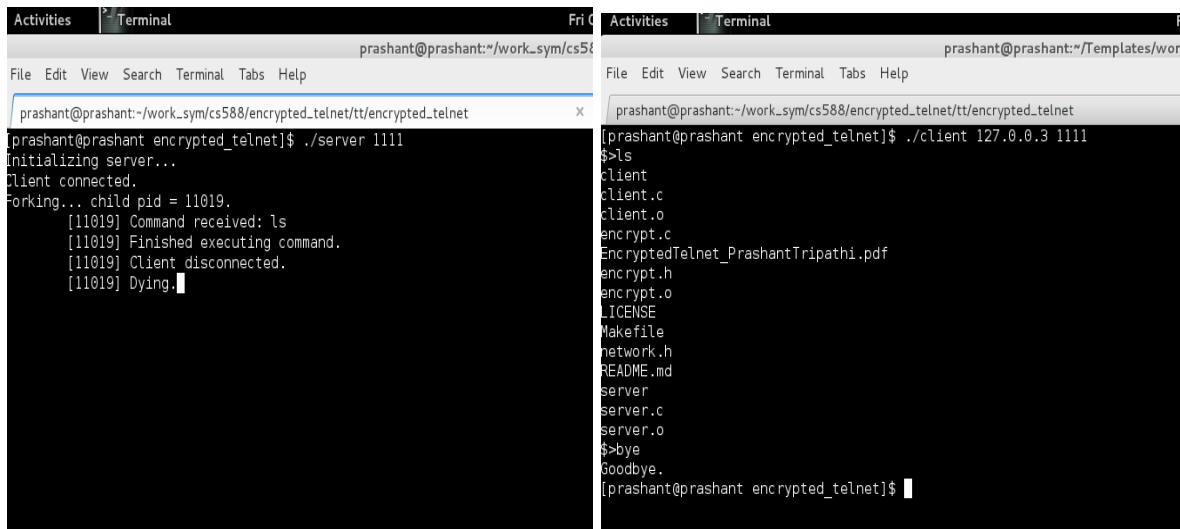
To run the application, the command format is as follows:

- To run the server: `./server <port number>`
- To run the client: `./client <server IP Address> <port number>`

The server and client sides were run on multiple machines with the symmetric key encryption scheme working perfectly between them. Failure to eavesdrop was evident by running Wireshark on the network and receiving all encrypted traffic. In case you need standard Telnet port 23 server has to run as root.

The private key which helps to decrypt the message has been defined in `encrypt.h`.

Some sample runs have been embedded below.



The image contains two side-by-side terminal window screenshots. The left terminal shows the server being run on port 1111. It displays the initialization process, a client connection, the execution of the 'ls' command, and the client's disconnection. The right terminal shows the client being run, connecting to the server at 127.0.0.3 on port 1111, and listing the contents of the current directory, which includes files like 'client', 'client.c', 'client.o', 'encrypt.c', 'EncryptedTelnet_PrashantTripathi.pdf', 'encrypt.h', 'encrypt.o', 'LICENSE', 'Makefile', 'network.h', 'README.md', 'server', 'server.c', 'server.o', and 'bye'.

```
prashant@prashant:~/work_sym/cs588/encrypted_telnet$ ./server 1111
Initializing server...
Client connected.
Forking... child pid = 11019.
[11019] Command received: ls
[11019] Finished executing command.
[11019] Client disconnected.
[11019] Dying.

prashant@prashant:~/work_sym/cs588/encrypted_telnet$ ./client 127.0.0.3 1111
$>ls
client
client.c
client.o
encrypt.c
EncryptedTelnet_PrashantTripathi.pdf
encrypt.h
encrypt.o
LICENSE
Makefile
network.h
README.md
server
server.c
server.o
bye
Goodbye.
```

Contributions

Mentioned below is what each member of the group contributed to the entire project:

- Prashant Tripathi: Coded the server side i.e `server.c`.
- Abhishek Singh Rathore: Developed the client side i.e `client.c` and the final report.

- Abhishek Kanchan: Implemented function calling utilizing the MCRYPT.lib file in order for symmetric key encryption to run between the client and the server i.e encrypt.c.

Each person's code was verified by the other two group members.

Future Work:

- Our code uses the MCRYPT_RIJNDAEL-128 but all the above mentioned symmetric algorithms can be provided as a choice to the user at runtime by modifying the network.h file.
- The code is extensible to incorporate in the telnet client a Kerberos authentication mechanism. If symmetric ciphers are to be used for secure communication between two or more parties or a large number of hosts, the management of such a large number of symmetric keys becomes an issue. Such problems can be solved using a hybrid approach that includes using asymmetric ciphers for which Kerberos can be useful.

References:

- Telnet RFC <https://tools.ietf.org/html/rfc854>
- <http://mcrypt.sourceforge.net/>
- <http://mcrypt.sourceforge.net/>
- Research of the Telnet Remote Login: Geng Sanjing, Huang Lihui
School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo, China