BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY

ISA project Manual

File Transfer Through a Hidden Channel

November 15, 2021

Fridrich David

Contents

1 Abstract

Program is divided into two parts: Client & Server. Program's purpose is to transfer (send AND recieve) a file in encrypted form through ICMP traffic. Client encrypts and subsequently sends a file to server's IP address. Server catches packets of data send by client (with specific packet ID & Sequence Number), decrypts and saves the file to current working directory (where server is run from).

2 Implementation

2.1 Overview

Program is sectioned into multiple files:

- **secret.h** – all function declarations (for both client and server)
- **secret.c** – contains main(), argument parsing, decides between client and server
- **client.c** – implementation of client logic (more Client subsection)
- **server.c** – implementation of server logic (more Server subsection)

Program begins by parsing command line arguments which determines the course of the program. More about argument parsing in 2.2.

2.2 Independent

2.2.1 Argument parsing

`func:  int parser(int argc, char **argv, char **file,char **host)`

Function takes in cmd arguments by `argv` and additional arguments `file` and `host` which are used to return given arguments to calling function to be passed further. Main arguments being resolved here are `-l`, `-s`, `-r` which determine if function is a server, ip/hostname (for client) and file (for client) respectfully. Once this is resolved, control of the program is passed respectfully to `client()` (see 2.3) or `server()` (see 2.4) or error is returned if wrong or missing arguments are given.

2.3 Client

`func:  void client(char *file, char *host)`

Client function checks whether `file` is valid argument. If it is, open a file for reading in binary mode, otherwise print error to stderr and exit unsuccessfully. Extracts file name and save it. Initialize structures and variables for socket creation and conection-less trasfer using `getaddrinfo()` [4] function which is a better `gethostname()` because `gethostname()` is depricated at this date. Sets up all structures and attempts to create a socket. If successful, everything is ready for transfer, otherwise error is returned. Now initializes icmp header which will be at the very front of the packet on the client side. By use of `createFirstPacket(char (*p)[PACKET_MAX_SIZE], int used, unsigned int l)` resolve file name length, file name itself and total length of the file. These 3 values are appended after icmp header in `packet[PACKET_MAX_SIZE]`, variable which holds data to send in one packet. Dynamically determine how much free space is left in `packet` by `int getMaxDataAvailable(int used,int done,int fl)` and read this size from file and append it to `packet`. Whole packet to send is setup, now call `unsigned char *encryptData(char *in,unsigned int *l)` to encrypt the data part of the packet (excluding icmp header) and return it back. Calculate checksum [9] of packet and send first packet containing specific details about the file being send. If the file did not fit into the first packet, us while loop to repeatedly send file data until the whole file is read and send.

2.4 Server

`func:  void server()`

Server function is inspired by my previous IPK project – sniffer and example file by Petr Matousek [6] and [7] Logic is implemented by pcap library. Program sniffs for ICMP packets (IPv4 and IPv6) on *any* interface. Initialize struct and variables used, compile and set filter as "icmp or icmp6" and listen for traffic indefinitely (or until CTRL+C is pressed or error occures). Every icmp packet is catched but right packets are determined by packet ID and sequence number which must follow the last one starting from 1. Since the program listens on interface *any*, recieved packets contain Linux Cooked Capture instead of Ethernet header, followed by IP header which every ICMP packet must be wrapped in, followed by the actual packet data. Packet data begins with icmp header followed by specific data (either file information or file data). Whole packet is decrypted at first (except icmp header), subsequently if its the first packet, file information is read. This also determines the length of the file itself therefore the program knows how much data to read. More about encryption in 2.5. Since the whole packet is decrypted regardless of how many bytes it actually holds, length of the file is known in order to read exact amount of bytes, because decryption might leave "garbage" behind the decrypted data if its not aligned to 16B.

2.5 Encryption

Encryption is done by `<openssl/aes.h>` library [1] [3] using `AES_set_encrypt_key`, `AES_encrypt` with specified key as my xlogin. Decryption is done exactly the same way, exchanging "encrypt" for "decrypt" in function names. Encryption is done in 16 byte blocks therefore data max length is calculated with this in mind (Enough space must be available if current data block to encrypt is NOT 16B but is less. AES_encrypt will encrypt 16B regardless and the encrypted data with "garbage" values are passed to packet).

3 Testing & Problems

Testing was done via `Wireshark` [2], provided VM (with Linux Ubuntu) and other locally connected PCs with Linux OS. These PCs must be connected locally, otherwise we encounter a problem with NAT and private networks, where the actual IP is not reachable. It is possible to do but NAT/router configuration would be required.
Since packets are send in swift succession it is possible that program tries to write to buffer in `sendto()` [8] before its available because of the previous data. This invokes an error during `sendto()` "No buffer space available". Because of this, `poll()` [5] function is used in order to check availability of socket file descriptor used in `sendto()`.
Another problem occurs when trying to send larger files. Network is not a really stable environment and recieving packets is not guaranteed in sent order or the fact that all packets are catched. Therefore during larger files its possible to lose same data along the way.

References

1. *AES_encrypt(3) - OpenBSD manual pages* [online] [visited on 2021-11-14]. Available from: `https://man.openbsd.org/AES_encrypt.3`.
2. COMBS, Gerald. *Wireshark* [online]. 2021 [visited on 2021-11-14]. Available from: `https://www.wireshark.org/`.
3. *EVP Symmetric Encryption and Decryption - OpenSSLWiki* [online] [visited on 2021-11-14]. Available from: `https://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption`.
4. *getaddrinfo(3) - Linux manual page* [online] [visited on 2021-11-14]. Available from: `https://man7.org/linux/man-pages/man3/getaddrinfo.3.html`.

5. JACOBSON, Van, et al. *pcap(3) - Linux User's Manual* [online]. 2021 [visited on 2021-11-14]. Available from: `https://www.tcpdump.org/pcap3_man.html`.
6. MATOUŠEK, Petr. *sniff-filter.c* [online]. 2020 [visited on 2021-11-14]. Available from: `https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FISA-IT%2Fexamples&cid=14699`.
7. MOON, Silver. *How to code a Packet Sniffer in C with Libpcap on Linux - BinaryTides* [online] [visited on 2021-11-14]. Available from: `https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/`.
8. *sendto(2): send message on socket - Linux man page* [online] [visited on 2021-11-14]. Available from: `https://linux.die.net/man/2/sendto`.
9. UNKNOWN. *RFC 1071 - Computing the Internet checksum (RFC1071)* [online] [visited on 2021-11-14]. Available from: `http://www.faqs.org/rfcs/rfc1071.html`.