USER MANUAL:

1. unzip ELFbinary_recon.zip into preferred directory
2. make sure elf binary is in that same directory for verification
   optional -> run 'make clean' followed by 'make all' to compile executables
3. start server executable to wait for a connection
4. start client executable with appropriate IP to connect to the server
5. execute available commands

'verify' → verify an elf binary
Prompt user for an elf binary to analyze. Sends to server if specified.

'request' → get information on a binary on the server
Request information on an analyzed binary on the server.
User will receive less information than admin privileges allow.
Admin is able to request all information on stored binaries.

'admin' → enter special administration mode (password = "#Harambe")

Credentials are processed successfully ONLY when the file 'elf' is present in that directory. The admin has special privilege of receiving all information for the requested binary that has been stored on the server

'exit' → close connection

Backdoor sequence to enter admin mode...
a) hit enter 5 times
b) type 'admin' command
c) type "harambe" when prompted for password
This will only work when steps a,b,and c are taken in that order

Technical/LOGICAL FLOW and DESIGN

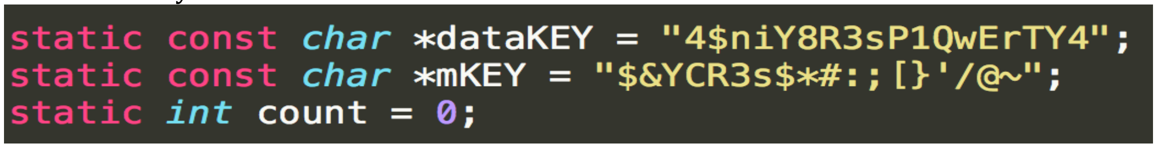
Connecting to the server:

Client.c and server.c compile to binaries ("client" and "server") and communicate via TCP/IP protocol. The client application connects to port on the server and proceeds to enter a handshake routine where it receives some computed value to the server.
[handshake = ((int) &sockfd) *100 %2379]
Verification of this handshake is done on the client sending back handshake+1. A successful handshake starts the client application, closing the connection otherwise.

Client Application: (refer to pg 4 for client application control flow)
The client application will receive prompts from the server as the message "->" to indicate it is waiting for some command from the client.
[refer pg1 for explanation of valid commands]

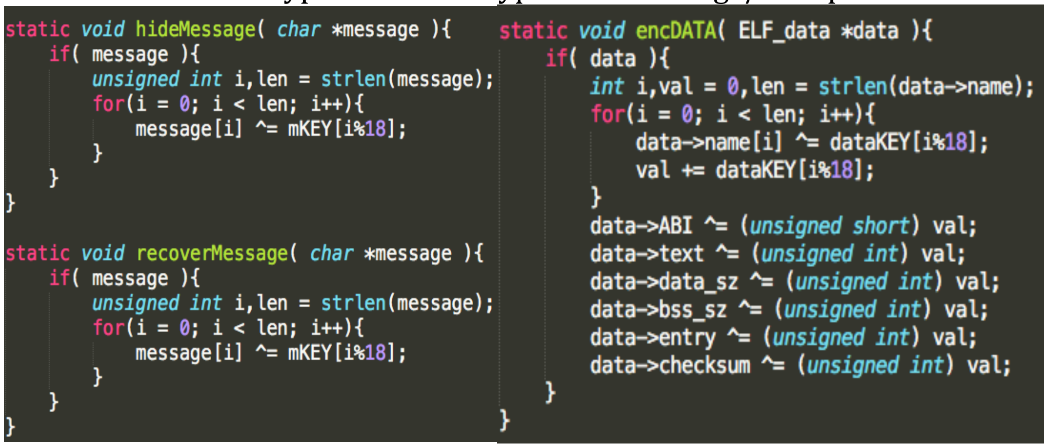
NOTE: The majority of communication between server and client is done by explicit message passing, in the form of encrypted characters.
→Data packets are encrypted/decrypted structs on both ends using private key (***dataKEY***)
→Messages are encrypted/decrypted on both ends using private key (***mKEY***)
→***count*** is used in the client application for backdoor purposes to set the admin flag
→a checksum is used on the struct to verify data integrity on both ends

Static hash keys and count



```c
static const char *dataKEY = "4$niY8R3sP1QwErTY4";
static const char *mKEY = "$&YCR3s$*#:;[}'/@~";
static int count = 0;
```

Source code for encryption and decryption of message/data packet



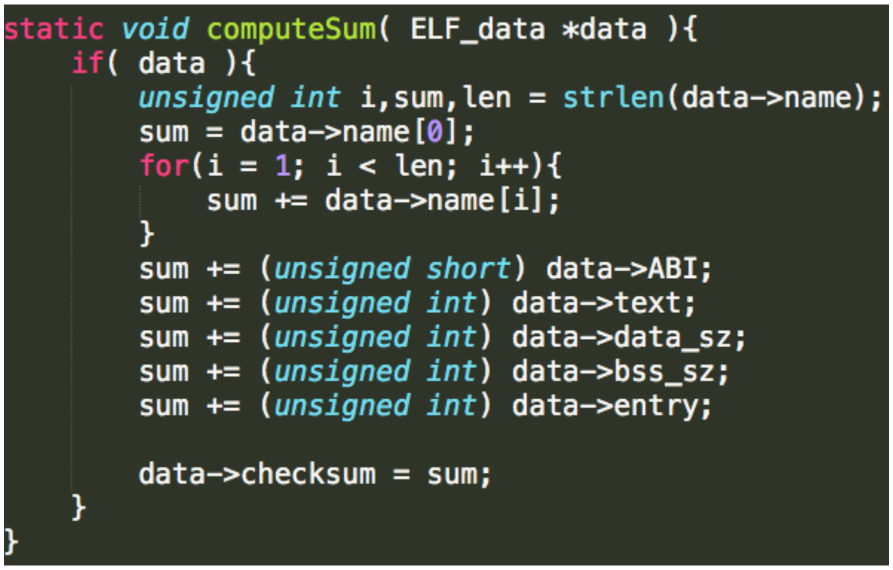
```c
static void hideMessage( char *message ){
    if( message ){
        unsigned int i,len = strlen(message);
        for(i = 0; i < len; i++){
            message[i] ^= mKEY[i%18];
        }
    }
}

static void recoverMessage( char *message ){
    if( message ){
        unsigned int i,len = strlen(message);
        for(i = 0; i < len; i++){
            message[i] ^= mKEY[i%18];
        }
    }
}
```

```c
static void encDATA( ELF_data *data ){
    if( data ){
        int i,val = 0,len = strlen(data->name);
        for(i = 0; i < len; i++){
            data->name[i] ^= dataKEY[i%18];
            val += dataKEY[i%18];
        }
        data->ABI ^= (unsigned short) val;
        data->text ^= (unsigned int) val;
        data->data_sz ^= (unsigned int) val;
        data->bss_sz ^= (unsigned int) val;
        data->entry ^= (unsigned int) val;
        data->checksum ^= (unsigned int) val;
    }
}
```

IMPORTANT: Admin mode can only be initiated through verification of the encrypted password ("131#Harambe131") found in the file named 'elf'. If this file is ever lost, it is only possible to enter admin mode through use of the backdoor process described on pg1 user

instructions. The password file ‘elf’ may be recreated by writing the result of hidemessage(saltedpass) function shown above, saltedpass = “131#Harambe131”.
Source code for checksum

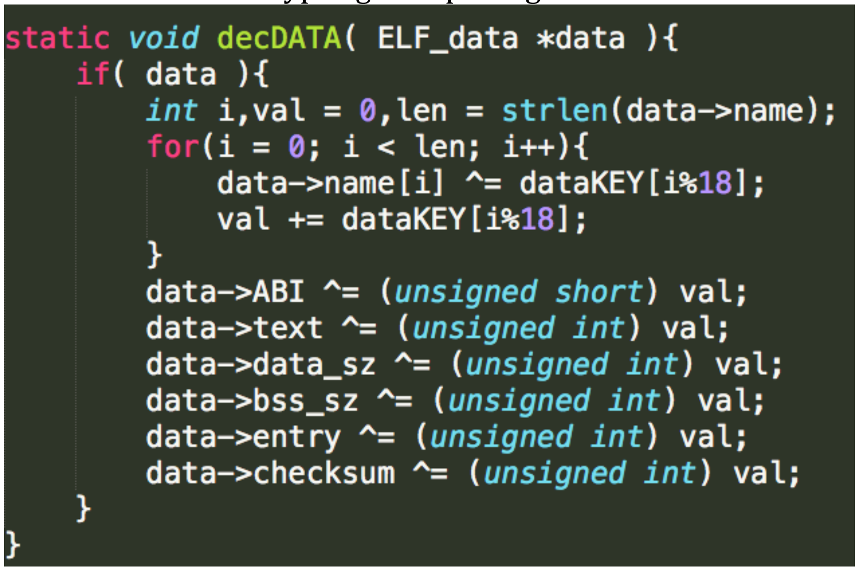


```c
static void computeSum( ELF_data *data ){
    if( data ){
        unsigned int i,sum,len = strlen(data->name);
        sum = data->name[0];
        for(i = 1; i < len; i++){
            sum += data->name[i];
        }
        sum += (unsigned short) data->ABI;
        sum += (unsigned int) data->text;
        sum += (unsigned int) data->data_sz;
        sum += (unsigned int) data->bss_sz;
        sum += (unsigned int) data->entry;

        data->checksum = sum;
    }
}
```

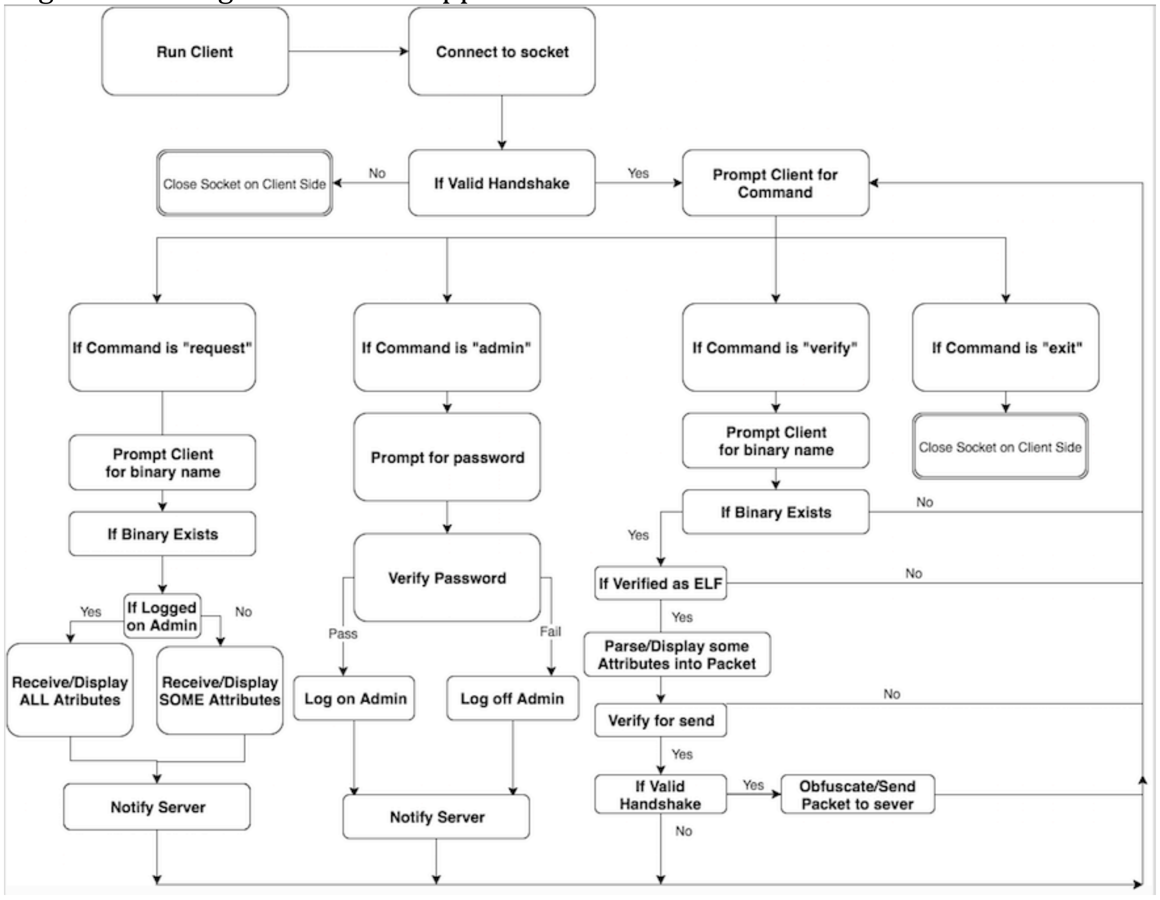
Server Application:
The server application will send prompts from the server as the message “->” to indicate it is waiting for some command from the client. It reacts accordingly to the commands (pg1) by sending and receiving the correct handshake value (pg 2) before receiving/sending packets.

Source code for decrypting data package



```c
static void decDATA( ELF_data *data ){
    if( data ){
        int i,val = 0,len = strlen(data->name);
        for(i = 0; i < len; i++){
            data->name[i] ^= dataKEY[i%18];
            val += dataKEY[i%18];
        }
        data->ABI ^= (unsigned short) val;
        data->text ^= (unsigned int) val;
        data->data_sz ^= (unsigned int) val;
        data->bss_sz ^= (unsigned int) val;
        data->entry ^= (unsigned int) val;
        data->checksum ^= (unsigned int) val;
    }
}
```

Logical flow diagram for client application


Run Client
Connect to socket
If Valid Handshake
No
Close Socket on Client Side
Yes
Prompt Client for Command
If Command is "request"
If Command is "admin"
If Command is "verify"
If Command is "exit"
Prompt Client for binary name
If Binary Exists
If Logged on Admin
Yes
No
Receive/Display ALL Atributes
Receive/Display SOME Attributes
Notify Server
Prompt for password
Verify Password
Pass
Fail
Log on Admin
Log off Admin
Notify Server
Prompt Client for binary name
If Binary Exists
No
Yes
If Verified as ELF
No
Yes
Parse/Display some Attributes into Packet
Verify for send
No
Yes
If Valid Handshake
Yes
Obfuscate/Send Packet to sever
No
Close Socket on Client Side