# REVERSING CLIENT AND SERVER

Emanuelle (Manny) Crespi UID: ecrespi, 111556427 ENEE459B - 0102

#### Running the Client and Server:

The client is allowed to run after the server has been set up to wait for a connection. (Figure 1)

```
111556427@ENEE459B-2:~/proj2$ ./server server: waiting for connections...
```

Figure 1: Server Application at Runtime

Following the connection, a prompt for the binary executable is displayed on the client application. It is important to give a valid IP as a command line argument to the client binary for full functionality (Figure 2).

```
[111556427@ENEE459B-2:~/proj2$ ./client 127.0.0.1 Please Enter the name of the executable:
```

Figure 2: Client Application at Runtime

For the software to send a data packet, a valid binary must exist within the "../elfs" directory from where the binary is running. This allows the application to parse, verify and send the file. Without this, the client program will complain that there is no file to open before terminating. Below is an example for both a successful run (Figure 3) that sends a packet to the server, and an unsuccessful run (Figure 4) that terminates without a correct file name.

```
Please Enter the name of the executable:
[client
size: 1202
client: connecting to 127.0.0.1

[111556427@ENEE459B-2:~/proj2$ ./server
server: waiting for connections...
server: got connection from 127.0.0.1
Got name...
Got data! Waiting for the next connection.
```

Figure 3: Successful Run Client-Server Interaction

```
111556427@ENEE459B-2:~/proj2$ ./client 127.0.0.1

Please Enter the name of the executable:
fasdf

File "../elfs/fasdf" NOT FOUND!

111556427@ENEE459B-2:~/proj2$ clear
```

Figure 4: Unsuccessful Run

There are also other cases to consider where the binary is not the correct format. Attempting to use another file type that is in the correct directory will cause the program to output "File is not ELF." before terminating (Figure 5).

```
I111556427@ENEE459B-2:~/proj2$ ./client 127.0.0.1

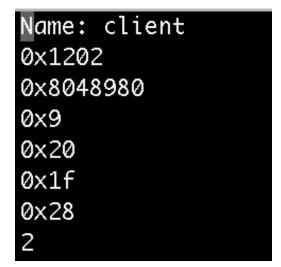
Please Enter the name of the executable:

[client1
File is not ELF.

111556427@ENEE459B-2:~/proj2$
```

Figure 5: Attempting to process an incorrect binary format

We can also verify a successful packet sending interaction by checking the directory where the server binary resides. The file name should be present on the server with the same name chosen on the client side prompt. The file contents are shown and described below for the example of a successful run above (Figure 6).



# **FORMAT - Head = Binary Name**

- size of text section (bytes)
- entry of text section
- # of program headers
- size of program headers
- # of section headers
- size of section headers
- data form

*Figure 6:* The contents/format of the file 'client1' after a successful data transfer

### Parsing/Obfuscating/Sending Data packet:

The parsing of the files is correct in retrieving the attributes of the binary data. This has been verified by careful observation of the source code in IDA Pro, followed by comparison to the output of 'readelf -h -S'.

```
ELF Header:
            7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Magic:
                                       ELF32
  Class:
  Data:
                                        2's complement, little endian 🥌
  Version:
                                        1 (current)
  OS/ABI:
                                       UNIX - System V
  ABI Version:
                                        EXEC (Executable file)
  Type:
  Machine:
                                        Intel 80386
                                        0 \times 1
  Version:
                                        0x8048980
  Entry point address:
                                        52 (bytes into file)
  Start of program headers:
                                        11840 (bytes into file)
  Start of section headers:
  Flags:
                                        0×0
                                        52 (bytes)
  Size of this header:
  Size of program headers:
                                        32 (bytes)
  Number of program headers:
                                       40 (bytes)
  Size of section headers:
                                        31
  Number of section headers:
  Section header string table index: 28
Section Headers:
  [Nr] Name
                                                     Off
                                                                    ES Flg Lk Inf Al
                           Type
                                            Addr
                                                             Size
  [ 0]
                                            00000000 000000 000000 00
                                                                             0
                          NULL
                                                                                 Ø
                                                                                    0
  [ 1] .interp
                          PROGBITS
                                            08048154 000154 000013 00
  [ 2] .note.ABI-tag
                                            08048168 000168 000020 00
                                                                                 0
                          HOTE
                                                                          A
                                                                             0
                                                                                 0
                                                                             0
    3] .note.gnu.build-i NOTE
                                            08048188 000188 000024 00
                                                                          А
                          GNU_HASH
                                                                             5
                                                                                 0
   4] .gnu.hash
                                            080481ac 0001ac 000024 04
                                                                          A
                                                                             6
                                                                          А
                                                                                 1
  [ 5] .dynsym
                          DYNSYM
                                            080<del>4</del>81d0 0001d0 000250 10
                                                                          A
                                                                             0
                                                                                 0
  [ 6] .dynstr
                          STRTAB
                                            08048420 000420 00014d 00
                                                                             5
  [ 7] .gnu.version
                          VERSYM
                                            0804856e 00056e
                                                             00004a 02
                                                                          A
                                                                                 0
                                                                                    2
    8] .gnu.version_r
                          VERNEED
                                            080<del>4</del>85b8 0005b8 000050 00
                                                                          A
                                                                             6
                                                                                 1
                                                                          A
                                                                             5
                                                                                 0
    9] .rel.dyn
                          REL
                                            08048608 000608 000010 08
                                                                             5
                                                                                24
  [10] .rel.plt
                          REL
                                            08048618 000618 000108 08
                                                                         ΑI
  [11] .init
                          PROGBITS
                                            08048720 000720 000023 00
                                                                        ΑX
                                                                                 0
                                                                         ΑX
                                                                                 0
  [12] .plt
                                            08048750 000750 000220 04
                          PROGBITS
                                                                                   16
  [13] .plt.got
                          PROGBITS
                                            08048970 000970 000008289
                                                                         HΧ
                                                                                 Ø
                                                                                    8
                          PROGBITS
                                            08048980 000980 001202 00
                                                                        RΧ
                                                                             0
                                                                                 0 16
  [14] .text
  [15] .fini
                           PROGBITS
                                            08049684 001684 000014 00
                                                                                 0
```

Figure 7: Parsing of the ELF files are reliable

Following the verification of the ELF file header, the client application establishes an <u>application level handshake</u> with the server by sending the value 17 in a buffer. The server expects this value on the other end, and responds by sending 18 before moving on to the next phase (Figure 8). On the client side, the next two sends will complete the transaction. The first send will contain a <u>request value</u> appended to the name of the ELF file, and the last send will forward the entire packet.

The server is waiting to receive each of these transactions, so it expects to retrieve the value 21 from the buffer after receiving the first data packet.

NOTE: The ELF filename is also placed within the packet before sending. The entire packet is encrypted by some key generated with a combination of functions beforehand esp, 18h

```
[ebp+power key enc], eax
mov
mov
        byte ptr [ebp+packet.var1], 21
MOVZX
        eax, [ebp+tcp_buf+1]
                         ; taking the char recieved from server in bu
        eax, 1
add
mov
        byte ptr [ebp+packet.var1+1], al
sub
        esp, 4
push
        92
                                                                       CLIENT
        eax, [ebp+filename]; copying filename to structure packet
1ea
add
        eax, 8
push
        eax
                          src
1ea
        eax, [ebp+packet]
                        ; offset 3 will be data section in packet
add
        eax, 3
                         ; dest
push
        eax
                         ; copy over the filename into head of data
call
        memcpy
add
        esp, 16
         eax, [ebp+packet]
ecx, 256
1ea
         ecx,
mov
mov
         edi, edx
mov
         esi, eax
rep movsd
         eax, esi
mov
mov
         edx, edi
movzx
         ecx.
              word ptr [eax]
         [edx], cx
edx, [edx+2]
mov
lea
         eax, [eax+2]
push
         ebx
.
call
         encrypt
                            ; encrypt before send
         esp, 40Ch
add
1ea
         edx, [ebp+packet]
         eax,
               [ebp+var_1158]
lea
                                                                      SERVER
               100h
mov
         ecx,
mov
         edi, edx
mov
         esi, eax
rep movsd
mov
         eax, esi
mov
         edx, edi
movzx
         ecx.
              word ptr [eax]
mov
         [edx], cx
lea
         edx, [edx+2]
Lea
         eax, [eax+2]
                              flags
push
         S
         1026
push
                             n
lea
         eax, [ebp+packet]
push
                             buf
         eax
         [ebp+fd]
.
push
                            ; fd
; send 1026 bytes of data (struct pa
call
         _send
```

Figure 8: Prep work for first (encrypted) send of packet structure to server

A brief explanation of the structure is necessary at this point before discussing the formatting/decryption/encryption of the message that the client prepares and sends to the server. The structure is identified as a 1026 byte word before it is sent to the server, so the following identity is what I have used to identify packet sends/retrieval (Figure 9).

```
struct packet{
    short a;
    short b;
    char data[1024]
}
```

Figure 9: Packet structure on the client side for sending ELF attributes to server

Retrieval of the first encrypted packet is identified to pass on the server side, only if 21 is present within its decrypted format [ packet.a == 21 ]. We can see below that the socket connection is terminated, if this condition is not met. When the condition is met, the server will proceed to wait for the next and final packet (Figure 10).

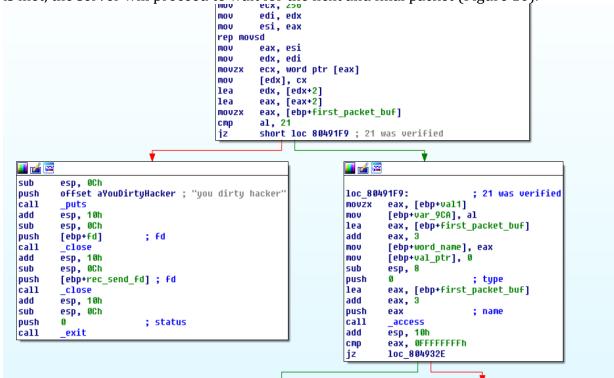


Figure 10: First packet is retrieved and verified by the server. There is one last request made by the client as it sends the final packet to the server. It checks for the same old request as before! Figure 11 below shows a comparison on the re-named variable 'old\_request' that has not been touched since the last request. This is implying a comparison on 21 yet again.

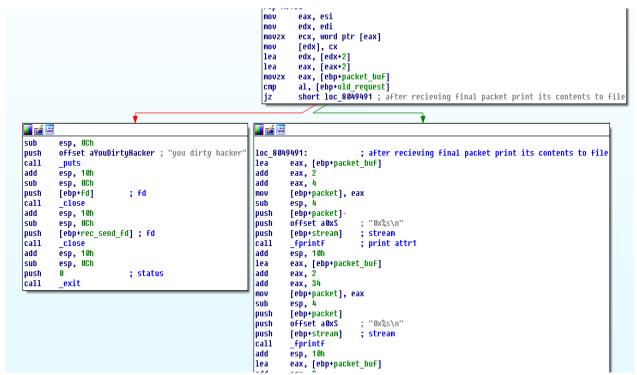


Figure 11: Passing the second request to process data from client and write to a file

Immediately after these two requests are fulfilled, the server proceeds to write the parsed attributes to the file. The offsets by which the server extracts from the structure indicate a mapping for the data elements that are written to the file since they are the same offsets as the ones used by the client to populate the buffer before sending the packet. Figure 12 below shows this overlap, and Figure 13 describes a mapping for where to find these 7 attributes in the buffer packet.

```
push
                                                 [ebp+packet]
        offset asc_8049C8F ; "%x"
push
                                        push
                                                 offset a0xS
                                                                   ; "0x%s\n"
1ea
        eax, [ebp+packet]
                                        push
                                                 [ebp+stream]
                                                                   ; stream
add
        eax, 6
                                        call
                                                 fprintf
                                                                   ; print attr1
push
        eax
                                        add
                                                 esp, 10h
call
        sprintf
                                        lea
                                                 eax, [ebp+packet_buf]
        esp, 10h
add
                                        add
                                                 eax, 2
MOVZX
        eax, [ebp+var_D2E]
                                        add
                                                 eax, 34
sub
        esp, 4
                                        mov
                                                 [ebp+packet], eax
bush
        eax
                                                 esp, 4
                                        sub
push
        offset asc_8049C8F ; "%x"
                                        push
                                                 [ebp+packet]
1ea
        eax, [ebp+packet]
                                        push
                                                 offset a0xS
                                                                   ; "0x%s\n"
add
        eax, OBAh
                                        push
                                                 [ebp+stream]
                                                                   ; stream
push
        eax
                       : 5
                                        call
                                                  fprintf
        _sprintf
call
                                        add
                                                 esp, 10h
add
        esp, 10h
                                        lea
                                                 eax, [ebp+packet_buf]
        eax, [ebp+var_CF8]
mov
                                        add
                                                 eax, 2
mov
        eax, [eax+18h]
                                        add
                                                 eax, 64
        esp, 4
                                        mov
                                                 [ebp+packet], eax
bush
        eax
                                        sub
                                                 esp, 4
        offset asc_8049C8F ; "%x"
push
                                        push
                                                 [ebp+packet]
        eax, [ebp+packet]
1ea
                                                                   ; "0x%s\n"
                                        push
                                                 offset a0xS
add
        eax, 24h
                                        push
                                                                   ; stream
                                                 [ebp+stream]
push
        eax
                                        call
                                                 _fprintf
        _sprintf
call
                                        add
                                                 esp, 10h
add
        esp, 10h
                                        1ea
                                                 eax, [ebp+packet_buf]
mov
        eax, [ebp+var_CF8]
                                        add
                                                 eax, 2
        eax, word ptr [eax+2Ch]
MOVZX
                                                 eax, 154
                                        add
MOVZX
        eax, ax
                                        mov
                                                 [ebp+packet], eax
sub
        esp, 4
                                        sub
                                                 esp, 4
push
        eax
                                        push
                                                 [ebp+packet]
        offset asc 8049C8F; "%x"
push
                                                                   ; "0x%s\n"
                                        push
                                                 offset a0xS
1ea
        eax, [ebp+packet]
                                                 [ebp+stream]
                                        push
                                                                   ; stream
add
        eax, 42h
                                        call
                                                  fprintf
push
        eax
                                        add
                                                 esp, 10h
call
        _sprintf
                                                 eax, [ebp+packet_buf]
                                        1ea
add
        esp, 10h
                                        add
                                                 eax, 2
mov
        eax, [ebp+var_CF8]
                                        add
                                                 eax, 124
MOVZX
        eax, word ptr [eax+2Eh]
                                        mov
                                                 [ebp+packet], eax
MOVZX
        eax, ax
                                        sub
                                                 esp, 4
sub
        esp, 4
                                        push
                                                 [ebp+packet]
push
                                                                   ; "0x%s\n"
                                        push
                                                 offset a0xS
nush
        offset asc 8049C8F; "%x"
                                        push
                                                 [ebp+stream]
                                                                   ; stream
1ea
        eax, [ebp+packet]
                                        call
                                                 _fprintf
add
        eax, 60h
                                        add
                                                 esp, 10h
bush
        eax
                       ; 5
                                        1ea
call
        _sprintf
                                                 eax, [ebp+packet_buf]
                                                 eax, 2
        esp, 10h
                                        add
add
                                        add
                                                 eax, 94
mov
        eax, [ebp+var_CF8]
                                        mov
                                                 [ebp+packet], eax
MOVZX
        eax, word ptr [eax+30h]
                                                 esp, 4
                                        sub
MOVZX
        eax, ax
sub
        esp, 4
                                        push
                                                 [ebp+packet]
                                                                   ; "0x%s\n"
push
                                        push
                                                 offset a0xS
        eax
        offset asc_8049C8F ; "%x"
push
                                        push
                                                 [ebp+stream]
                                                                   ; stream
                                        call
1ea
        eax, [ebp+packet]
                                                 _fprintf
add
        eax, 7Eh
                                        add
                                                 esp, 10h
push
        eax
                                        1ea
                                                 eax, [ebp+packet_buf]
call
        _sprintf
                                        add
                                                 eax, 2
add
        esp, 10h
                                        add
                                                 eax, 184
                                                 [ebp+packet], eax
        eax, [ebp+var CF8]
mov
                                        mov
        eax, word ptr [eax+2Ah]
MOUZX
                                        sub
                                                 esp, 4
MOVZX
        eax, ax
                                        push
                                                 [ebp+packet]
                                                                   ; "%s\n"
sub
        esp,
                                        push
                                                 offset aS
push
        eax
                                        push
                                                 [ebp+stream]
                                                                   ; stream
push
        offset asc_8049C8F ; "%x"
                                       call
                                                 _fprintf
              CLIENT
                                                       SERVER
```

Figure 12: The offsets are useful for mapping the data within the packet

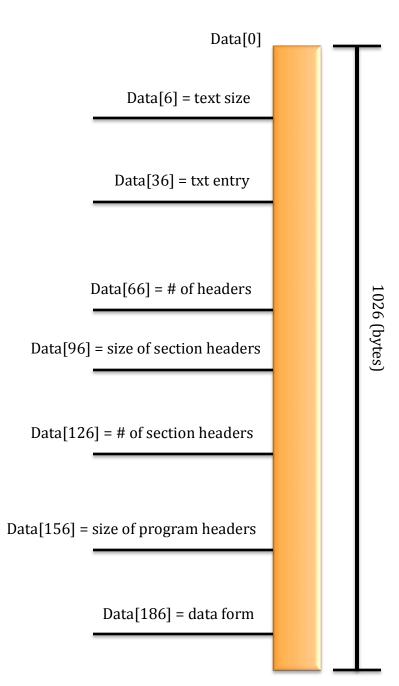


Figure 13: Mappings of the attributes residing in data

The mappings of the seven attributes are implied by the order in which the

It is also important to see that there is enough space at the head of the data packet where at least 6 bytes of data

server writes to the file.

are free.

lo.	Time	≜ Source	Destination	Protocol	Length			
	2 0.000012	::1	::1	TCP				ACK] Seq=1 Ack=1 Win=0 Len=0
	3 0.000107	127.0.0.1	127.0.0.1	TCP				Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2
	4 0.000117	127.0.0.1	127.0.0.1	TCP	74	2532→59384	[SYN,	ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PER
	5 0.000126	127.0.0.1	127.0.0.1	TCP				Seq=1 Ack=1 Win=43776 Len=0 TSval=2440709166 TSecr=
	6 0.000381	127.0.0.1	127.0.0.1	TCP	1092	59384→2532	[PSH,	ACK] Seq=1 Ack=1 Win=43776 Len=1026 TSval=244070916
	7 0.000386	127.0.0.1	127.0.0.1	TCP	66	2532→59384	[ACK]	Seq=1 Ack=1027 Win=45824 Len=0 TSval=2440709166 TSe
	8 0.000458	127.0.0.1	127.0.0.1	TCP	1092	2532+59384	[PSH,	ACK] Seq=1 Ack=1027 Win=45824 Len=1026 TSval=244070
	9 0.000465	127.0.0.1	127.0.0.1	TCP	66	59384→2532	[ACK]	Seq=1027 Ack=1027 Win=45824 Len=0 TSval=2440709166
	10 0.000474	127.0.0.1	127.0.0.1	TCP	1092	59384→2532	[PSH,	ACK] Seq=1027 Ack=1027 Win=45824 Len=1026 TSval=244
	11 0.000487	127.0.0.1	127.0.0.1	TCP	1092	59384→2532	[FIN,	PSH, ACK] Seq=2053 Ack=1027 Win=45824 Len=1026 TSva
	12 0.000490	127.0.0.1	127.0.0.1	TCP	66	2532→59384	[ACK]	Seq=1027 Ack=3080 Win=49920 Len=0 TSval=2440709166
	13 0.001002	127.0.0.1	127.0.0.1	TCP	66	2532→59384	[FIN,	ACK] Seq=1027 Ack=3080 Win=49920 Len=0 TSval=244070
	Urgent pointer							
⊳ ( ⊳ Dat	Options: (12 by [SEQ/ACK analys a (1026 bytes)	ytes), No-Operation sis]	(NOP), No-Operation		tamps			
⊳ ( ⊳ Dat	Options: (12 by [SEQ/ACK analys a (1026 bytes) Data: 140229626 [Length: 1026]	ytes), No-Operation sis] 5d68646f750101010101	0101010101010101010101		tamps			
Dat	Options: (12 by [SEQ/ACK analysta (1026 bytes) Data: 140229626 [Length: 1026]	ytes), No-Operation sis] 5d68646f750101010101 8 00 00 00 00 00 00	0101010101010101010101	1	Ε.			
Dat	Options: (12 by [SEQ/ACK analysta (1026 bytes) Data: 140229626 [Length: 1026] 00 00 00 00 00 04 36 97 32 40	ytes), No-Operation sis]  5d68646f7501010101010101010101010101010101010101	0101010101010101010101 000 08 00 45 00	5.2@.@	E.			
Dat  0000 0010 0020	Options: (12 by [SEQ/ACK analys: a (1026 bytes) Data: 140229626 [Length: 1026] 00 00 00 00 04 36 97 32 44 00 01 e7 f8 09	ytes), No-Operation sis]  5d68646f7501010101010101010101010101010101010101	0101010101010101010101 00 08 00 45 00 00 00 01 7f 00 87 d9 39 80 18	1 5.2@.@	E.			
Dat (	Options: (12 by [SEQ/ACK analy: a (1026 bytes) Data: 140229626 [Length: 1026] 00 00 00 00 04 36 97 32 44 00 01 07 f8 09 01 66 02 2b 00	ytes), No-Operation sis]  5d68646f7501010101010101010101010101010101010101	0101010101010101010101 00 08 00 45 00 00 00 01 7f 00 .6 87 d9 39 80 18 7a 44 2e 91 7a .f	5.2@.@	E.			
Dat 1	Options: (12 by [SEQ/ACK analy; a (1026 bytes) Data: 140229626 [Lenoth: 1026] 00 00 00 00 00 00 00 00 00 00 00 00 00	ytes), No-Operation sis] 5d68646f7501010101010101010101010101010101010101	01010101010101010101010101010101010101	3.2@.@. C3g	E.			
Dat (	Options: (12 by ISEQ/ACK analy: a (1026 bytes) abata: 140229624 [Lenoth: 1026] 00 00 00 00 00 00 00 00 00 00 00 00 00	ytes), No-Operation sis]  5d68646f7501010101010101010101010101010101010101	00000000000000000000000000000000000000	5.2@.@. C3	E			
Dat  0000 0010 0020 0030 0040 0050 0060 0070	Options: (12 by [SEQ/ACK analy; a (1026 bytes) Data: 14022962[[Length: 1026]] 00 00 00 00 00 00 00 01 67 78 05 01 66 02 2b 04 42 e 14 02 25 01 01 01 01 01 01 01 01 01 01 01 01 01	ytes), No-Operation sis] 5d68646f7501010101010101010101010101010101010101	01010101010101010101010101010101010101	5.2@.@. 	E			
Dat    0000   0010   0020   0030   0040   0050   0060   0070   0080	Options: (12 b) [SEQ/ACK analy: a (1026 bytes) Data: 14022962! [Length: 1026] 00 00 00 00 00 44 36 97 32 44 00 01 e7 f8 00 10 60 02 2b 04 42 e 14 02 25 01	ytes), No-Operation sis]  5d68646f7501010101010101010101010101010101010101	01010101010101010101010101010101010101	5.2@.@. 	E.			
Dat  Dat  Dat  Dat  Dat  Dat  Dat  Dat	Options: (12 b) [SEQ/ACK analys] a (1026 bytes) Data: 140229626 [Lenath: 1026] 00 00 00 00 00 04 36 97 32 40 00 01 07 78 09 01 66 02 2b 00 04 42 2c 14 02 22 01	ytes), No-Operation sis]  5668646f7501010101010101010101010101010101010101	00 08 00 45 00 00 08 00 45 00 00 08 00 01 7f 00 87 d9 39 80 18 7a 44 2e 91 7a .f 101 01	5.2@.@. C3	E			
Dat (10000 0000 0000 0000 0000 0000 0000 0	Options: (12 b) [SEQ/ACK analys] a (1026 bytes) Data: 140229626 [Lenath: 1026] 00 00 00 00 00 04 36 97 32 40 00 01 07 78 09 01 66 02 2b 00 04 42 2c 14 02 22 01	ytes), No-Operation sis]  5d68646f75010101010101  8 00 00 00 00 00 00 00 00 00 00 00 00 00	01010101010101010101010101010101010101	5.2@.@. 	E			
Dat  Dat  Dat  Dat  Dat  Dat  Dat  Dat	Options: (12 b) [SEQ/ACK analys] a (1026 bytes) Data: 140220626 [Length: 1020] 00 00 00 00 00 04 36 97 32 40 00 01 07 f8 09 01 66 02 2b 09 01 66 02 2b 01	ytes), No-Operation sis]  5d68646f75010101010101  8 00 00 00 00 00 00 00 00 00 00 00 00 00	00000000000000000000000000000000000000	1 5.2@.@. 	E			
Dat  Dat  Dat  Dat  Dat  Dat  Dat  Dat	Options: (12 by [SEQ/ACK analys] (a (1026 bytes) Data: 14022962 [Length: 1026] 00 00 00 00 00 00 00 00 00 00 00 00 00	ytes), No-Operation sis]  5668646f75010101010101  9 00 00 00 00 00 00  9 00 40 06 a1 8d 7f  9 04 43 33 47 03 b4  9 00 10 11 08 0a 91  9 06 26 66 86 64 6f 75  10 10 10 10 10 10 10  10 10 11 01 01 01  10 10 11 01 01 01  10 10 11 01 01 01  10 10 11 01 01 01  10 10 11 01 01 01  10 10 11 01 01 01  10 10 11 01 01 01  10 10 11 01 01 01  10 10 11 01 01 01  10 10 11 01 01 01  10 10 10 10 10 10 10  10 10 10 10 10 10 10  10 10 10 10	01010101010101010101010101010101010101	1 5.2@.@. C3	E			

Figure 14: Intercepting client- server communication with Wireshark

At this point we know the order of the packets being sent and by whom. Using wire-shark to read intercepted messages, it has become clear that the message 09e44333 shows up in multiple packets during the sending process. Even though the encryption/decryption scheme is overly complicated to reverse, we can still analyze a consistent pattern between packets being sent to the server.

#### Vulnerabilities/Limitations:

It is also important to note that there is no checksum used on the other end to detect malformed data. If a packet, sent from client to server, has been intercepted and modified, the server would never be aware of this. It would continue to load the data into its database as long as there is a correct header = 21. Cannot find any problems with the obfuscation, encryption, decryption of the messages since I can hardly decipher it myself!

## Logical Flow Chart (Server-Client Relationship):

Finally, with the data collected throughout this analysis of client and server, it would help to have a logical model (Figure 15) describing every interaction between these two applications. This way, any useful information I may have missed may be reconsidered as an added bonus to the reversing for this design.

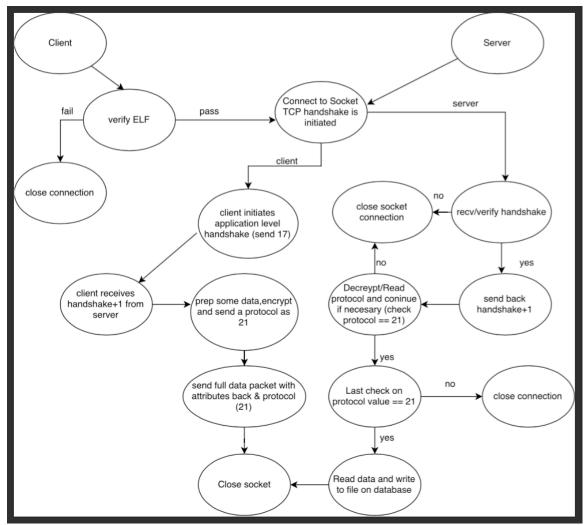


Figure 15: Logical Model Client-Server Flow Chart