Homework Number: 10

Name: Alex Goebel

ECN Login: goebel2

Due Date: 4/15/2021

Buffer overflow string: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x18\x0e\x40\x00

The reason this string was selected is because I took the base address and the stack pointer address at the first breakpoint, which was at the entry of clientComm, and found the difference between these addresses to be 40. This meant that I needed 40 As and then the address value (in reverse) to successfully accomplish the buffer overflow attack. This amount of As was able to get the last 4 bytes of the entry address to secretFunction to overwrite the return address of the stack and therefore execute the secretFunction.

The modified server code is pasted below:

```
/*

/ 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.

/*
Homework Number: 10
Name: Alex Goebel
ECN Login: goebel2
Due Date: 4/15/2021
*/

#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);
    }


    /*The issue was using strcpy(str, recvBuff). This would try and put too many bytes into str which was
only allocated with MAX_DATA_SIZE bytes, which in this case was 5. The way I fixed it was by checking
the size of the senderBuff and if that was greater than MAX_DATA_SIZE I just exited the program*/
    if (*senderBuffSize_addr > MAX_DATA_SIZE) {
            fprintf(stderr, "ERROR, client sent too many bytes. Exiting");
        exit(1);
  }
  strcpy(str, recvBuff);

  /* 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);
}
```

The highlighted portion signifies the point of interest. The underlined part of the highlighted code is what I actually added. Essentially, the issue was that str was only allocated MAX_DATA_SIZE amount of bytes, which in this case was 5. Therefore, when trying to send in more than 5 bytes from the client, it would result in a segfault. This was what was exploited to successfully run the secretFunction. To fix this, I simply put an if statement in the check if the size of the senderBuff was greater than the MAX_DATA_SIZE, then the program immediately exits and doesn’t run the strcpy, thus avoiding the overflow issue.