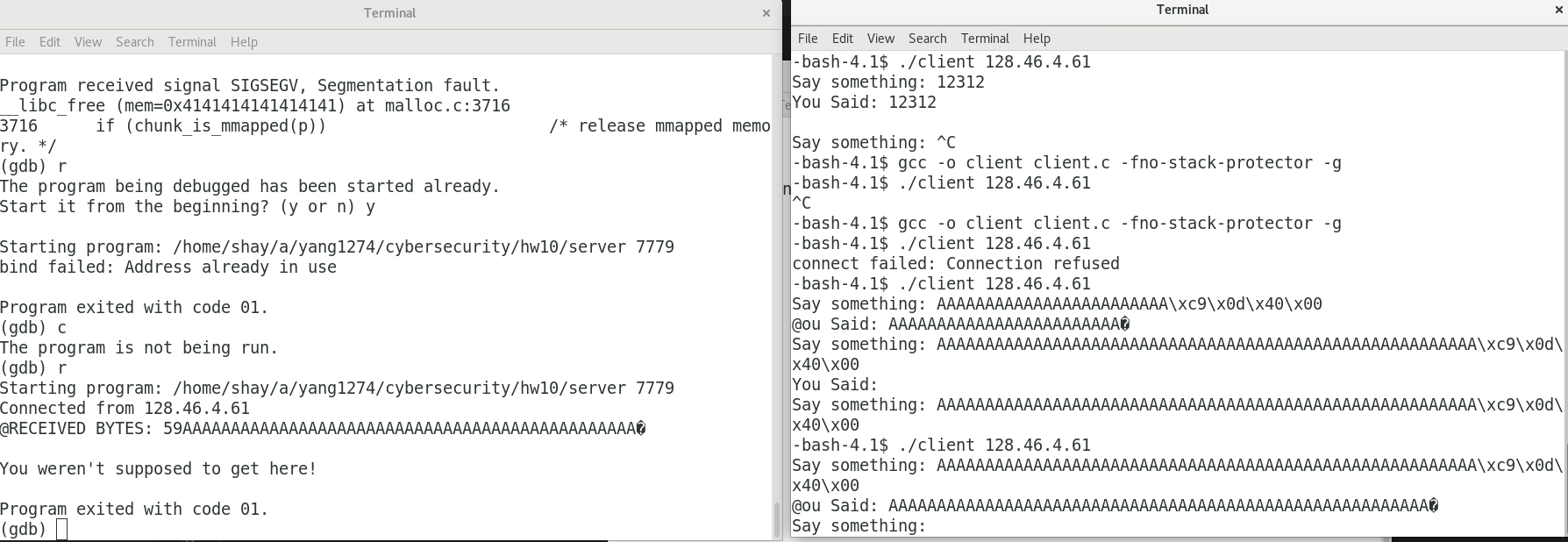
Homework Number: hw8

Name: Jiaxing Yang

ECN Login: yang1274

Due Date: 4/02/2020

Buffer Overflow attack string:



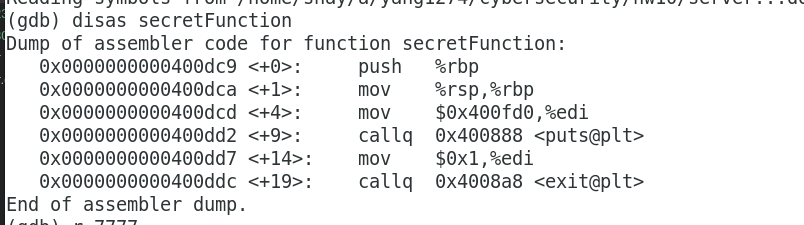
Explanation:

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



The address of the rbp is 7fffffffdf90, and the address of the string from the client is 7ffffffffdf60, 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):



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);   
  
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;  
 }  
}