ECE404

HW11

Yudi Wu

0026402011

wu553@purdue.edu

This server program is vulnerable to the buffer overflow attack.

The “str” which is a copy of “recvbuff” that is used to send back to the client. This “str” variable initially is only defined with size of 5 bytes, but input could be 4096 bytes. When the client send something which is larger than 5 bytes, the information would go to other place on the stack and that would cause serious problems.

To fix this problem, I decided to make “str” allocated in runtime. “str” is allocated in the receiving while loop with a size of “senderBuffSize”, in the result “recvbuff” and “str” would have the same size, and there would be no overflow when the server does the strcpy.

Server2.c

/\*

/ file : server2.c

/-----------------------------------------------------------------------------

/

/ This is the server side of a server-client pair for the buffer-overflow

/ homework in ECE 404 at Purdue. The client program is called client2.c and

/ should be available at the same location where you found this server program.

/

/ This server program echos back the messages received from a client. Initially,

/ you may wish to run the server in one window and the client in another window

/ on the same machine by issuing calls like

/

/ server2 9000

/ client2 localhost 9000

/

/ Subsequently, run the server on one of the ECN machines and the client

/ on your laptop.

\*/

// For compiling this file:

// Linux: gcc server2.c -o server2

// Solaris: gcc server2.c -o server2 -lsocket -lnsl

//

// Solaris needs to be explicitly told about the libraries libsocket and

// libnsl. The latter library is for "Network Service Library"; it has

// the implementation code for functions such as gethostbyname(), etc.

//

// For running the server program:

//

// server2 9000

//

// where 9000 is the port you want your server to monitor. Of course, this can

// be any high-numbered port that is not currently being used by others.

//

// Note that this server program DOES NOT terminate when the client shuts down the

// client-side socket, say, by entering the ctrl-C interrupt.

//

// Code originally pulled off the internet a very long time back and then modified by

// Avi Kak on April 12, 2014. The modifications to the original code are: (1) The

// original server code would die if the client killed the process running on its

// side by entering Ctrl-C. The version shown below does not do that. (2) The

// original server program also terminated even when a client exited under normal

// conditions. The version shown below should not do that. And, finally,

// (3) The original program could not be compiled on Solaris. The new version is.

// For that, the following call

// inet\_ntoa(clientAddr.sin\_addr))

// was replaced by

// inet\_ntop(AF\_INET, &(clientAddr.sin\_addr), strrr, INET\_ADDRSTRLEN));

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <strings.h> // for bzero(), bcopy()

#include <sys/socket.h>

#include <arpa/inet.h>

//#include <unistd.h>

//#include <sys/types.h>

//#include <netinet/in.h>

//#include <sys/wait.h>

#define MAX\_PENDING 10 /\* maximun # of pending for connection \*/

#define MAX\_DATA\_SIZE 5

int display\_connection\_data(char \*recvBuff,char \*str, int numBytes);

int main(int argc, char \*argv[]) {

if (argc < 2) {

fprintf(stderr,"ERROR, no port provided\n");

exit(1);

}

int PORT = atoi(argv[1]);

char \*recvBuff; /\* recv data buffer \*/

int numBytes = 0;

int senderBuffSize;

//changed part

//char str[MAX\_DATA\_SIZE];

char \*str;

//

int servSockfd, clientSockfd;

struct sockaddr\_in sevrAddr;

struct sockaddr\_in clientAddr;

int clientLen;

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); // set the first 8 bytes to zero, that is '\0'

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) {

printf("Waiting for a client to connect\n\n");

clientLen = sizeof(struct sockaddr\_in);

if ((clientSockfd = accept(servSockfd, (struct sockaddr \*) &clientAddr, &clientLen))

== -1) {

perror("accept failed");

exit(1);

}

char str2[INET\_ADDRSTRLEN];

// The call to inet\_ntop from the defined in `arpa/inet.h' reuturns a newtwork

// address in the `struct in\_addr' format to its decimal-dot notation. Older

// versions of such software used to call inet\_ntoa() for the same thing:

printf("Connected from %s\n", inet\_ntop(AF\_INET, &(clientAddr.sin\_addr),

str2, INET\_ADDRSTRLEN));

if (send(clientSockfd, "Connected!!!\n",

strlen("Connected!!!\n"), 0) == -1) {

perror("send failed");

close(clientSockfd);

exit(1);

}

// Wait in an infinite loop for the client to say something. This inner

// is executed once for each message from the client.

while(1) {

/\* recv data from the client \*/

getsockopt(clientSockfd, SOL\_SOCKET,SO\_SNDBUF,

&senderBuffSize, &optlen); /\* check sender buffer size \*/

recvBuff = malloc(senderBuffSize \* sizeof (char));

//changed part

str = malloc(senderBuffSize \* sizeof (char));

//

if ( (numBytes = recv(clientSockfd, recvBuff, senderBuffSize, 0)) <= 0) {

printf("The client closed the socket\n");

close(clientSockfd);

break;

}

recvBuff[numBytes] = '\0';

strcpy(str, recvBuff);

/\* send data to the client \*/

if (send(clientSockfd, str, strlen(str), 0) == -1) {

perror("send failed");

close(clientSockfd);

exit(1);

}

display\_connection\_data(recvBuff, str, numBytes);

}

}

}

int display\_connection\_data(char \*recvBuff,char \*str, int numBytes) {

printf("RECEIVED: %s", recvBuff);

printf("SENT: %s", str);

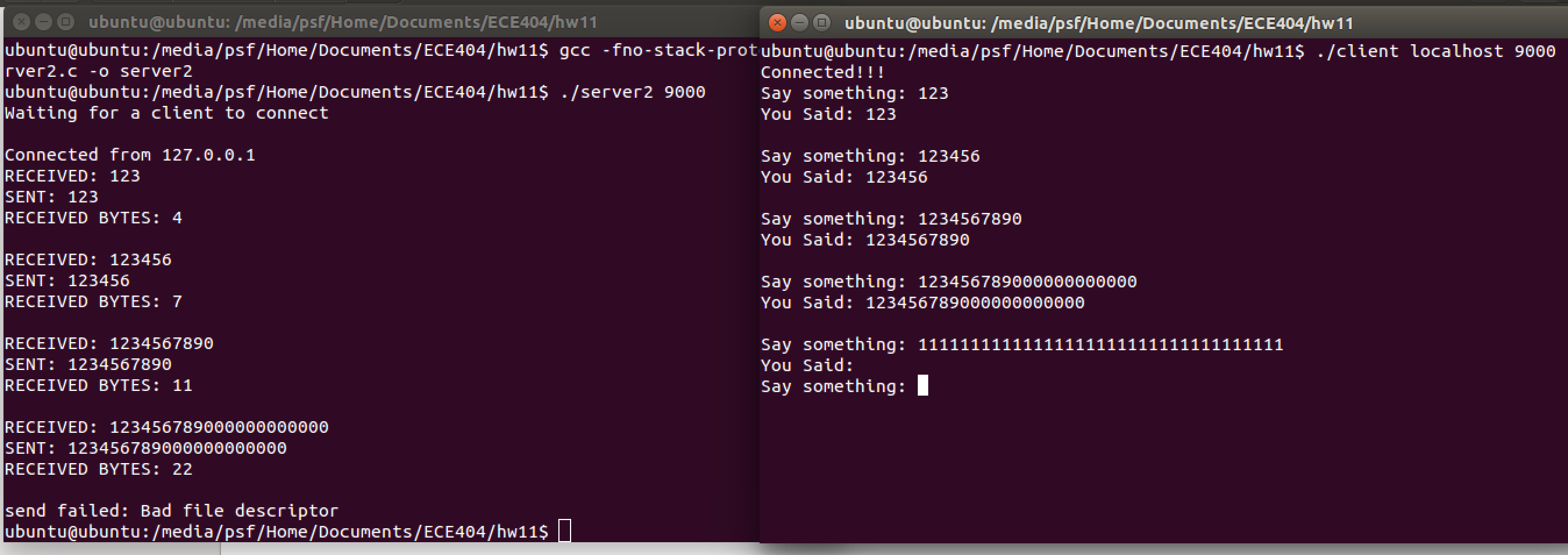
printf("RECEIVED BYTES: %d\n\n", numBytes);

return(0);

}

Output:

Before updating the server.c, if the input is too large, it returns the error bad file descriptor.



After the update, there would be no issues anymore.