CPSC 526 Fall 2017 – Assignment 1

Due date: **Sunday October 1, 2017 at 23:59.**
Weight: 10% of your final grade.
Group work is allowed but not required. Maximum group size is 2 people.

In this assignment you are going to find a vulnerability in a badly implemented server program and then exploit it to break its security.

Running the server program:

You can download the source code for the server program from the assignment page:

https://sites.google.com/site/cpsc526626fall2017/assignments/assignment-1

To compile this code, you will need to use a C compiler and a Linux machine, e.g.:

```
$ gcc secretServer.c
```

The compiled executable will accept 3 command line arguments: a port number, a password and a secret phrase. Once started, the server will listen for new connections on the specified port. When a client connects to the port, the server will read a single line from the client. If that line consists of the correct password, the server will reveal the secret phrase to the client. If the password is incorrect, the server will not reveal the secret phrase.

Below is an example of starting the server and interacting with it. Let's start the server:

```
$ ./a.out 1234 P4$$ 'My favorite number is 42.'
Waiting for a new connection...
```

In the above example the server will listen on port '1234' for connections. To connect to the server, you can use the 'nc' utility program (or 'telnet'). You should do this from a different terminal window, e.g.:

```
$ nc localhost 1234
Secret Server 1.0
```

When the client connects, the server sends the client the text 'Secret Server 1.0'. At this point the server will be waiting for the client to send a password. Below is an example of the client sending the wrong password, and the server reacting by not revealing the secret:

```
$ nc localhost 1234
Secret Server 1.0
xxx
I am not talking to you, bye!
```

If you provide the correct password, the server will reveal the secret phrase:

```
$ nc localhost 1234
Secret Server 1.0
P4$$
The secret is: My favorite number is 42.
```

Whether you supply the right or wrong password, the server will eventually close the connection. The server will then continue listening on the original port for a new connection. The only way to terminate the server is to kill it, for example using <CTRL-C>.

Task 1 – Exploit the vulnerability

The secret server program has a buffer overflow bug hidden in it, which can be exploited by a client over the network. For the first part of the assignment you must figure out how to accomplish this. If you are successful, you will be able to force the server to reveal its secret without knowing the password with which it was started.

In your report you should briefly describe this exploit. Also, you will demonstrate to your TA during a live demo that you understand what the bug is, and that you know how to exploit it.

Task 2 – Fix the vulnerability

For the second part of the assignment you will need to remove the vulnerability from the source code. During the live demo you will need to demonstrate to your TA that you fixed the vulnerability and be able to explain how you fixed it. You will submit the fixed source code to D2L.

Additional notes

- You may implement your program in C or C++.
- Your program must run on the Linux machines in the labs (MS).
- You may not call any external programs.
- You will be required to demo your assignments individually to your TA. Demo times will be arranged by your TA.
- You are allowed to work on this assignment with another student (max. group size is 2 students). But beware that during the demo you will be asked to demonstrate your familiarity with all of the code. So if you do decide to group up, both of you should understand the code 100%.

Submission

You must submit your source code and your report (**`readme.pdf` or `readme.txt`**) to D2L. Please use ZIP or TAR archives. If you decide to work in a group, each group member needs to submit the assignment.

Your report must include:

- Your name, ID and tutorial section, and if applicable, the name of your group partner;
- A section that describes how to compile and how to run your code (if different from original version).
- Sample exploit of the original server.

You must submit the above to D2L to receive any marks for this assignment.

Marking

Category	Marks
Describe the vulnerability and the exploit (demo and report).	10
Show you can break into an unmodified server (demo and report).	10
Show you can break into a server after modification of some of the buffer sizes (demo).	10
Show you can force a particular password (demo).	10
Demonstrate the lack of exploit in your fixed server (demo and source).	20
Describe your fix (demo).	20
Program documentation and style (source).	20

General information about all assignments:

1. Late assignments or components of assignments will not be accepted for marking without approval for an extension beforehand. What you have submitted in D2L as of the due date is what will be marked.

2. **Extensions** may be granted for reasonable cases, but only by the course instructor, and only with the receipt of the appropriate documentation (e.g., a doctor's note). Typical examples of reasonable cases for an extension include: illness or a death in the family. Cases where extensions will not be granted include situations that are typical of student life, such as having multiple due dates, work commitments, etc. Forgetting to hand in your assignment on time is not a valid reason for getting an extension.

3. After you submit your work to D2L, make sure that you check the content of your submission. It's your responsibility to do this, so make sure that you submit your assignment with enough time before it is due.

4. All assignments should include contact information, including full name, student ID and tutorial section, at the very top of each file submitted.

5. Although group work is allowed, you are not allowed to copy the work of others. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k-5.html.

6. You can and should submit many times before the due date. D2L will simply overwrite previous submissions with newer ones. It’s better to submit incomplete work for a chance of getting partial marks, than not to submit anything.

7. Only one file can be submitted per assignment. If you need to submit multiple files, you can put them into a single container. Supported container types are: ZIP, TAR or gzipped TAR. No other formats will be accepted.

8. Assignments will be marked by your TA. If you have questions about the assignment or assignment marking, contact your TA first. If you still have question after you have talked to your TA then you can contact your instructor.

Appendix A – source code for the server

You can download this file (secretServer.c) from the assignment page:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <stdarg.h>
#include <ctype.h>

// global variables nicely grouped
struct {
    int port; // listening port
    char buffer[32]; // temporary buffer for input
    char password[32]; // required password
    char secret[1024]; // the secret to reveal
} globals;

// report error message & exit
void die( const char * errorMessage, ...) {
    fprintf( stderr, "Error: ");
    va_list args;
    va_start( args, errorMessage);
    vfprintf( stderr, errorMessage, args);
    fprintf( stderr, "\n");
    va_end( args);
    exit(-1);
}

// read a line of text from file descriptor into provided buffer
void readLineFromFd( int fd, char * buff) {
    char * ptr = buff;
    while(1) {
        // try to read in the next character from fd, exit loop on failure
        if( read( fd, ptr, 1) < 1) break;
        // character stored, now advance ptr
        ptr ++;
        // if last character read was a newline, exit loop
        if( * (ptr - 1) == '\n') break;
    }
    // rewind ptr to the last read character
    ptr --;
    // trim trailing spaces (including new lines, telnet's \r's)
    while(ptr > buff && isspace(* ptr)) ptr --;
    // terminate the string
    * (ptr + 1) = '\0';
}

// write a string to file descriptor
int writeStrToFd( int fd, char * str) {
   return write( fd, str, strlen( str));
}
```

```
int main( int argc, char ** argv)
{
    // parse command line arguments
    if( argc != 4) die( "Usage: server port password secret");
    char * end = NULL;
    globals.port = strtol( argv[1], & end, 10);
    if( * end != 0) die( "bad port %s", argv[1]);
    strncpy( globals.password, argv[2], sizeof( globals.password));
    globals.password[ sizeof(globals.password)-1] = 0;
    strncpy( globals.secret, argv[3], sizeof( globals.secret));
    globals.secret[sizeof(globals.secret)-1] = 0;

    // create a listenning socket on a given port
    struct sockaddr_in servaddr;
    int listenSockFd = socket(AF_INET, SOCK_STREAM, 0);
    if( listenSockFd < 0) die("socket() failed");
    bzero( (char *) & servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htons(INADDR_ANY);
    servaddr.sin_port = htons(globals.port);
    if( bind(listenSockFd, (struct sockaddr *) & servaddr, sizeof(servaddr)) < 0)
        die( "Could not bind listening socket: %s", strerror( errno));

    // listen for a new connection
    if( listen(listenSockFd, 3) != 0)
        die( "Could not listen for incoming connections.");

    while(1) {
        printf( "Waiting for a new connection...\n");
        // accept a new connection
        int connSockFd = accept(listenSockFd, NULL, NULL);
        if( connSockFd < 0) die( "accept() failed: %s", strerror(errno));
        printf( "Talking to someone.\n");
        // sey hello to the other side
        writeStrToFd( connSockFd, "Secret Server 1.0\n");
        // read response from socket
        readLineFromFd( connSockFd, globals.buffer);
        // check if it was a correct password
        if( strcmp( globals.buffer, globals.password) == 0) {
            // password was correct, reveal the secret
            printf( "Someone used the correct password.\n");
            writeStrToFd( connSockFd, "The secret is: ");
            writeStrToFd( connSockFd, globals.secret);
            writeStrToFd( connSockFd, "\n");
        }
        else {
            // password was incorrect, don't reveal the secret
            printf( "Someone used an incorrect password.\n");
            writeStrToFd( connSockFd, "I am not talking to you, bye!\n");
        }
        // close the connection
        close( connSockFd);
    }

    // this will never be called
    close( listenSockFd);
    return 0;
}
```