SI485H Stack Based Binary Exploits and Defenses
FALL 2015

NAME:_____

HW2

COLLABORATOR(S):_____

1. Fill in the descriptions for different parts of the sample code below:

3/2/0

3/2/0

3/2/0

3/2/0

3/2/0

3/2/0

3/2/0

4/2/0

```
0x0804841d <+0>:    push   ebp
0x0804841e <+1>:    mov    ebp,esp
0x08048420 <+3>:    and    esp,0xfffffff0

0x08048423 <+6>:    sub    esp,0x30

0x08048426 <+9>:    mov    DWORD PTR [esp+0x1d],0x6c6c6548
0x0804842e <+17>:   mov    DWORD PTR [esp+0x21],0x57202c6f
0x08048436 <+25>:   mov    DWORD PTR [esp+0x25],0x646c726f
0x0804843e <+33>:   mov    WORD PTR [esp+0x29],0xa21
0x08048445 <+40>:   mov    BYTE PTR [esp+0x2b],0x0

0x0804844a <+45>:   lea    eax,[esp+0x1d]
0x0804844e <+49>:   mov    DWORD PTR [esp+0x2c],eax

0x08048452 <+53>:   jmp    0x804846b <main+78>

0x08048454 <+55>:   mov    eax,DWORD PTR [esp+0x2c]
0x08048458 <+59>:   movzx  eax,BYTE PTR [eax]
0x0804845b <+62>:   movsx  eax,al
0x0804845e <+65>:   mov    DWORD PTR [esp],eax
0x08048461 <+68>:   call   0x8048310 <putchar@plt>
0x08048466 <+73>:   add    DWORD PTR [esp+0x2c],0x1

0x0804846b <+78>:   mov    eax,DWORD PTR [esp+0x2c]
0x0804846f <+82>:   movzx  eax,BYTE PTR [eax]
0x08048472 <+85>:   test   al,al
0x08048474 <+87>:   jne    0x8048454 <main+55>
0x08048476 <+89>:   mov    eax,0x0

0x0804847b <+94>:   leave
0x0804847c <+95>:   ret
```


/25

NAME: _____

3. What is the difference between a BYTE PTR, DWORD PTR, WORD PTR, and QWORD PTR?

6/4/2/0

4. For the x86 dissaembly below draw lines matching each of the jumps to where they jump to

```
0x0804841d <+0>:     push   ebp
0x0804841e <+1>:     mov    ebp,esp
0x08048420 <+3>:     and    esp,0xfffffff0
0x08048423 <+6>:     sub    esp,0x30
0x08048426 <+9>:     mov    DWORD PTR [esp+0x1f],0x74616542
0x0804842e <+17>:    mov    DWORD PTR [esp+0x23],0x796d7241
0x08048436 <+25>:    mov    BYTE PTR [esp+0x27],0x0
0x0804843b <+30>:    mov    DWORD PTR [esp+0x2c],0x0
0x08048443 <+38>:    jmp    0x804846c <main+79>
0x08048445 <+40>:    mov    DWORD PTR [esp+0x28],0x0
0x0804844d <+48>:    jmp    0x8048460 <main+67>
0x0804844f <+50>:    lea    eax,[esp+0x1f]
0x08048453 <+54>:    mov    DWORD PTR [esp],eax
0x08048456 <+57>:    call   0x80482f0 <puts@plt>
0x0804845b <+62>:    add    DWORD PTR [esp+0x28],0x1
0x08048460 <+67>:    cmp    DWORD PTR [esp+0x28],0x3
0x08048465 <+72>:    jle    0x804844f <main+50>
0x08048467 <+74>:    add    DWORD PTR [esp+0x2c],0x1
0x0804846c <+79>:    cmp    DWORD PTR [esp+0x2c],0x4
0x08048471 <+84>:    jle    0x8048445 <main+40>
0x08048473 <+86>:    leave
0x08048474 <+87>:    ret
```

6/4/2/0

5. For the x86 assambly above, how many times is the call to **puts** executed? **Explain.**

6/4/2/0

6. For the x86 assembly above: What is being outputted to the screen for the **puts** call? **Explain.**

6/4/2/0

7. Write the C equivalent of this program for the x86 asembly above:

6/4/2/0

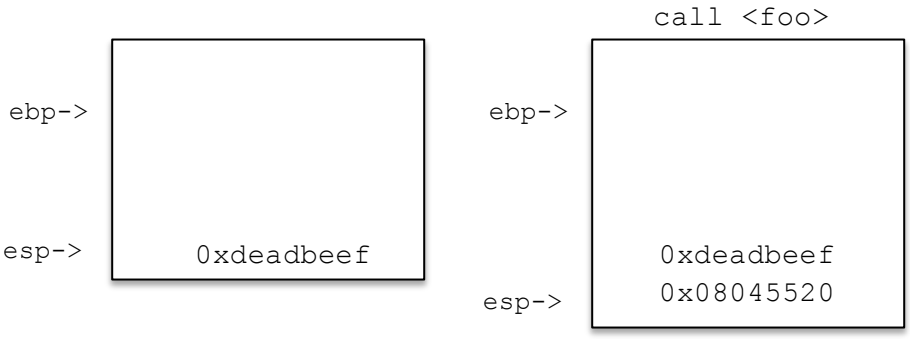
__/30

NAME: _____

8. Consider the following scenario for setting rigth before the a function call to foo() from the function bar():

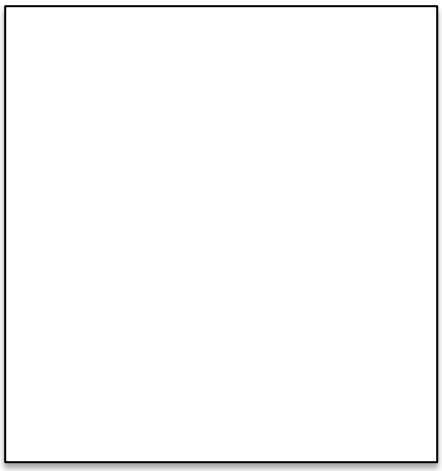
- The calling functions instruction pointer will be at address 0x08045520 after foo() returns
- The register ebp has value 0xbffff540
- The register esp has value 0xbffff504
- Foo takes one argument, an integer, the value 0xdeadbeef

The stack famres after each of the routines highlighted. The initial stack frame for bar() right before the call to foo() and right after would look like:

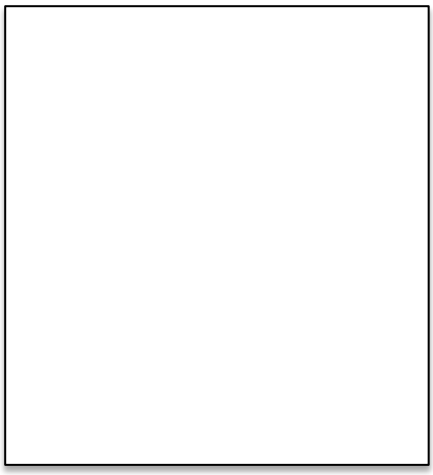


Complete the rest of the stack frame construction for foo():

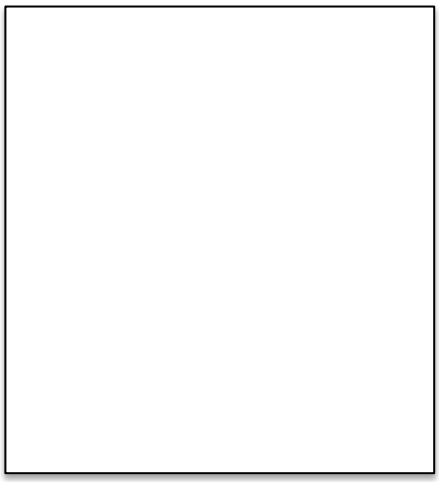
6/4/2/0
a) push ebp



6/4/2/0
b) mov ebp,esp



6/4/2/0
c) sub esp, 0x30



7/5/2/0
9. Using the same information as above, consider what happens after the function foo() returns. In x86, write the sequence of instructions used to reset the stack to bar() and reset the instruction pointer:

__/25

NAME: _____

10. If the regsiter eax stores 0xdeadbeef what is:

3/2/0 a) ax

3/2/0 b) al

3/2/0 c) ah

11. Consider the source code and the x86 instructions:

```
int foo(int a){
  int i,r;
  r=0;
  for(i=1;i<=a;i++){
    r+=i;
  }
  return r;
}
```

```
0x0804846d <+0>:     push   ebp
0x0804846e <+1>:     mov    ebp,esp
0x08048470 <+3>:     sub    esp,0x10
0x08048473 <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x0804847a <+13>:    mov    DWORD PTR [ebp-0x8],0x1
0x08048481 <+20>:    jmp    0x804848d <foo+32>
0x08048483 <+22>:    mov    eax,DWORD PTR [ebp-0x8]
0x08048486 <+25>:    add    DWORD PTR [ebp-0x4],eax
0x08048489 <+28>:    add    DWORD PTR [ebp-0x8],0x1
0x0804848d <+32>:    mov    eax,DWORD PTR [ebp-0x8]
0x08048490 <+35>:    cmp    eax,DWORD PTR [ebp+0x8]
0x08048493 <+38>:    jle    0x8048483 <foo+22>
0x08048495 <+40>:    mov    eax,DWORD PTR [ebp-0x4]
0x08048498 <+43>:    leave
0x08048499 <+44>:    ret
```

a) Match the memory address to the variable name:

3/2/0 ebp-0x4

3/2/0 ebp-0x8

3/2/0 ebp+0x8

c) On line <+38> how is the jle command interact with the cmp command on the previous line <+35>?

3/2/0

4/2/0 c) In what register is the return value placed? And what line of code indicates that?

__/25