SI485H Stack Based Binary Exploits and Defenses
FALL 2015

NAME:______________________________

HW4

COLLABORATOR(S):______________________________

1. Consider the following x86 code for function **foo**:

```
(gdb) ds foo
Dump of assembler code for function foo:
   0x0804846d <+0>:     push   ebp
   0x0804846e <+1>:     mov    ebp,esp
   0x08048470 <+3>:     sub    esp,0x28
   0x08048473 <+6>:     lea    eax,[ebp-0x18]
   0x08048476 <+9>:     mov    DWORD PTR [esp+0x4],eax
   0x0804847a <+13>:    mov    DWORD PTR [esp],0x8048540
   0x08048481 <+20>:    call   0x8048360 <scanf@plt>
   0x08048486 <+25>:    lea    eax,[ebp-0x18]
   0x08048489 <+28>:    mov    DWORD PTR [esp+0x4],eax
   0x0804848d <+32>:    mov    DWORD PTR [esp],0x8048540
   0x08048494 <+39>:    call   0x8048330 <printf@plt>
   0x08048499 <+44>:    leave
   0x0804849a <+45>:    ret
End of assembler dump.
(gdb) x/s 0x8048540
0x8048540:  "%s"
```

5/3/1/0 a) **CIRCLE** the function call in the x86 code where there is a potential buffer overflow and security violation.

10/8/5/0 b) Write the equivalent C code for **foo** below.

10/8/5/0 c) Consider executing the program like so:

python -c "print 'A'***x**"| ./main

What is the *smalleset* value of **x** that will crash the program. **Explain** how this causes a crash. *(hint: you do not have to overflow to the return address to cause the first crash)*

____/25

NAME: ______________________________

2. Consider the following x86 code analysis for function **foo**:

```
(gdb) ds foo
Dump of assembler code for function foo:
   0x0804844d <+0>:     push   ebp
   0x0804844e <+1>:     mov    ebp,esp
   0x08048450 <+3>:     sub    esp,0x48
   0x08048453 <+6>:     mov    DWORD PTR [ebp-0xc],0x0
   0x0804845a <+13>:    mov    eax,DWORD PTR [ebp+0x8]
   0x0804845d <+16>:    mov    DWORD PTR [esp+0x4],eax
   0x08048461 <+20>:    lea    eax,[ebp-0x2c]
   0x08048464 <+23>:    mov    DWORD PTR [esp],eax
   0x08048467 <+26>:    call   0x8048320 <strcpy@plt>
   0x0804846c <+31>:    jmp    0x804848c <foo+63>
   0x0804846e <+33>:    lea    eax,[ebp-0x2c]
   0x08048471 <+36>:    mov    DWORD PTR [esp+0x8],eax
   0x08048475 <+40>:    mov    eax,DWORD PTR [ebp-0xc]
   0x08048478 <+43>:    mov    DWORD PTR [esp+0x4],eax
   0x0804847c <+47>:    mov    DWORD PTR [esp],0x8048540
   0x08048483 <+54>:    call   0x8048310 <printf@plt>
   0x08048488 <+59>:    add    DWORD PTR [ebp-0xc],0x1
   0x0804848c <+63>:    cmp    DWORD PTR [ebp-0xc],0x2
   0x08048490 <+67>:    jle    0x804846e <foo+33>
   0x08048492 <+69>:    leave
   0x08048493 <+70>:    ret
End of assembler dump.
(gdb) r "Go Navy"
Starting program: /home/user/git/si485-binary-exploits/hw/04/demo/main "Go Navy"
0: Go Navy
1: Go Navy
2: Go Navy
[Inferior 1 (process 3044) exited with code 013]
```

10/8/5/0

a) Write the source code for the function **foo**:

b) Consider executing the program **main** which calls foo using the command line arguments like in the gdb code.

./main \`python -c "print 'A'\***x**"\`

At what value of **x** does the functionality of the loop change? **Explain.**

5/3/1/0

c) Complete a command line argument so that the loop will execute **4 times** and **Explain why.**
(hint: 2's-compliment numbers)

10/8/5/0

./main \`python -c " ______________________ "\`

NAME: ____________________________

3. Consider the dissaembled x86 code below:

```
(gdb) ds foo
Dump of assembler code for function foo:
   0x08048461 <+0>:     push   ebp
   0x08048462 <+1>:     mov    ebp,esp
   0x08048464 <+3>:     sub    esp,0x28
   0x08048467 <+6>:     mov    eax,DWORD PTR [ebp+0x8]
   0x0804846a <+9>:     mov    DWORD PTR [esp+0x4],eax
   0x0804846e <+13>:    lea    eax,[ebp-0xc]
   0x08048471 <+16>:    mov    DWORD PTR [esp],eax
   0x08048474 <+19>:    call   0x8048310 <strcpy@plt>
   0x08048479 <+24>:    lea    eax,[ebp-0xc]
   0x0804847c <+27>:    mov    DWORD PTR [esp],eax
   0x0804847f <+30>:    call   0x8048320 <puts@plt>
   0x08048484 <+35>:    leave
   0x08048485 <+36>:    ret
End of assembler dump.
(gdb) ds bar
Dump of assembler code for function bar:
   0x0804844d <+0>:     push   ebp
   0x0804844e <+1>:     mov    ebp,esp
   0x08048450 <+3>:     sub    esp,0x18
   0x08048453 <+6>:     mov    DWORD PTR [esp],0x8048540
   0x0804845a <+13>:    call   0x8048320 <puts@plt>
   0x0804845f <+18>:    leave
   0x08048460 <+19>:    ret
End of assembler dump.
(gdb) x/s 0x8048540
0x8048540:  "Beat Army!"
(gdb) r "Go Navy!"
Starting program: /home/user/git/si485-binary-exploits/hw/04/demo/main
"Go Navy!"
Go Navy!
[Inferior 1 (process 3129) exited with code 011]
```

a) Write the source code for **foo** and **bar**: 5/3/1/0

b) At what value of **x** will the **return address** be overwritten? **Explain.**

./main `python -c "print 'A'***x**"`

10/8/5/0

c) Complete the command line so that **bar** executed due to smashing the stack.

10/8/5/0

./main `python -c " ______________________ "`

NAME: ______________________________

5/3/1/0

4. What is *shell code*? And what three properties must it have?

5. Match the **execve** arguments to their register settings for the system call:
(a) eax (b) ebx (c) ecx (d) edx

```
char * argv[]= {"/bin/sh", NULL};
execve(argv[0], argv, NULL);
```

5/3/1/0
6. Why does sytem calls use registers for passing arguments?

5/3/1/0
7. What registers is used for the return value? Why does this well match the system call return value semantics?

6. Consider the following x86 code that is initializing the arguments for the **exec** system call before the interupt.

```
0x0804806e <+14>:     mov    eax,0xb
0x08048073 <+19>:     lea    ebx,[esp+0xc]
0x08048077 <+23>:     mov    ecx,DWORD PTR [esp]
0x0804807a <+26>:     mov    edx,0x0
0x0804807f <+31>:     int    0x80
```

Complete the stack diagram such that the shell code completes properly. The value of esp is 0xbffff72c.

10/8/5/0

esp ->