SI485H Stack Based Binary Exploits and Defenses
FALL 2015

NAME:_____

HW8

10/8/5/1/0 COLLABORATOR(S):_____

1. How many bits in a 32-bit memory address are typically randomized in ASLR?

3/1/0

2. For the following address, what is the non randombase address? For example, for the stack, the base portion is 0xbf80000.

0x09bed40 3/1/0

0xbd90023 3/1/0

0xe11eefa 3/1/0

3. Consider a guessing game, where you are trying to guess the number I'm thinking of right now. It's a number between 1 and **m. Every time I guess, I change my number!** How many guesses, **n,** would it take until you have a 50% chance of gettting it right within that number of guesses? Write your answer in reduced form with respect to **m** and **n. Hint: use base 2 log.**

8/5/3/1/0

4. Explain how this problem from before relates to breaking Address Space Layout Randomization.

5/3/1/0

__/25

NAME: _____

5. Explain how a NOP sled improve your ability to brute force an ASLR program? Consider the situation of 32 bit address spaces from the class notes.

5/3/1/0

6. Consider a 48 bit memory address space (6 bytes) that emlpoys 22 bits of randomness when basing the stack address. The first 20 bits are fixed for the stack, e.g., 0xbff, the next 22 bits are randomn, and the last 6 bits are fixed again. (**Write a program to solve these problems!**)

a) If you had **no** NOP sled, how many brute force attempts would it take to hack this program with 50% liklihood?

5/3/1/0

b) If you had a NOP sled of length 255 bytes, how many brute force attempts would it take to the hack this program with 50% liklihood?

5/3/1/0

c) If you had a NOP sled of length 8192 bytes, how many breut force attemps would it take to the hack this program with 50% liklihood?

5/3/1/0

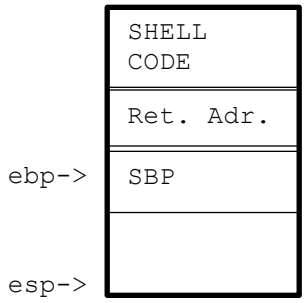
d) Suppose you wanted to ensure you hacked this program with 50% liklihood with 25 guesses, how long must your NOP sled be?

5/3/1/0

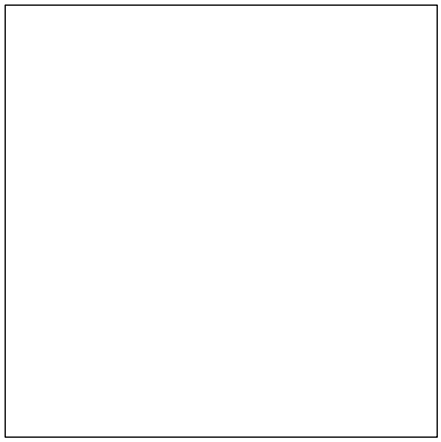
__/25

NAME: _____

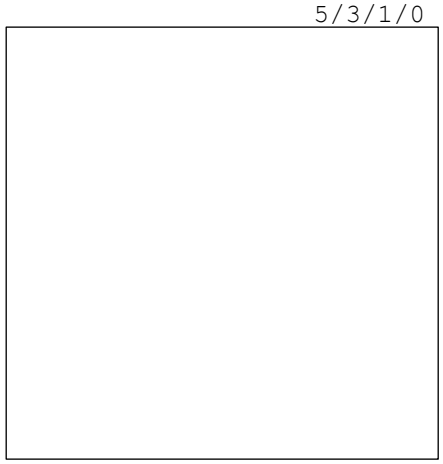
7. Complete the stack diagrams for leave and ret for the following function:



5/3/1/0 After **leave**



After **ret** 5/3/1/0



8. Using the diagram above, explain why a **jmp esp** or a **call esp** instruction will execute the shell code? How can this technique be used to subvert ASLR?

5/3/1/0

9. The following is a hexdump of the text segement of a program, what addresses (plural!) could be used as bounce points? **CIRCLE the bytes below, and write the addresses in the box.**

```
0x08048470 b830a004 082d30a0 0408c1f8 0289c2c1 .0...-0.........
0x08048480 ea1f01d0 d1f87501 c3ba0000 000085d2 ......u.........
0x08048490 74f65589 e583ec18 89442404 c7042430 t.U......D$...$0
0x080484a0 a00408ff d2c9c389 f68dbc27 00000000 ...........'....
0x080484b0 803d30a0 04080075 135589e5 83ec08e8 .=0....u.U......
0x080484c0 7cffffff c60530a0 040801c9 f3c36690 |.....0.......f.
0x080484d0 a1109f04 0885c074 1fb80000 000085c0 .......t........
0x080484e0 74165589 e583ec18 c7042410 9f0408ff t.U.......$.....
0x080484f0 d0c9e979 ffffff90 e973ffff ff5589e5 ...y.....s...U..
0x08048500 5dc35589 e583ec18 c7042470 860408e8 ].U.......$p....
0x08048510 acfeffff ffe4c9c3 5589e583 ec18c704 ........U.......
0x08048520 24858604 08e896fe ffffffd4 c9c35589 $.............U.
0x08048530 e583ec48 8b450c89 45c465a1 14000000 ...H.E..E.e.....
0x08048540 8945f431 c0c745d0 00000000 8b45c489 .E.1..E......E..
0x08048550 4424048d 45d48904 24e852fe ffffeb20 D$..E...$.R....
```

10/8/5/1/0

__/25

NAME: _____

11. What is linux gate and why in 2.6.X kernels was bouncing off linux gate a problem for defeating ASLR? What address can you jump to in linux gate to do a bounce?

5/3/1/0

12. What is basing for defeating ASLR?

5/3/1/0

13. What is dmesg and what output does it provide that is useful when trying to hack a program via basing?

5/3/1/0

15. For the following dmesg output, match the label to the output:

```
[2877400.157327] logger[23967]: segfault at a0a35 ip 08048af3 sp bfe11eb0 error 4 in
                                                              logger[8048000+1000]
```

5/3/1/0

(a) Address of instruction pointer
(b) The stack pointer
(c) The memory map of the programs text segment
(d) The address that was being derferenced causing the segfault

14. Consider using the following overflow on a remote program:

```
python -c "print 'A'*X + '\xef\xbe\xfe\xca'" | netcat localhost 2525
```

When **X** is correct such that the right number of padding bytes is present, what should the **dmesg** output be.

5/3/1/0

__/25