Homework 1 for CS 165 (Winter 2019)

Due: on ilearn by the end of day on Feb 6 2019

Instructions:

* Be brief in your answers. You will be graded for correctness, not on the length of your answers.

* Remember to submit online through ilearn if you didn't turn it in in-class. Paper copy will not be accepted.

I. Answer the following multiple choice questions (one or more correct answers) about password. (1 point x 4)

1. Why don't we store user passwords directly in a machine so it can authenticate a user by comparing the input password with the correct one? ______A,C____

a) Because if the machine is compromised, passwords can be stolen directly.

b) Because password is supposed to be remembered only by human.

c) Because there are better ways to store them so that the passwords won't be easily stolen, and yet still be able to authenticate users.

2. Which of the following describes a denial of service attack? ______B__C__

a) It cannot be detected.

b) It can stop legitimate users from using a service.

c) It always happens over the network.

3. Why is computer security about looking at corner cases of a program? __A________

a) Because vulnerabilities or exploitable bugs occur when programs deviate from their expected behaviors.

b) Because many security vulnerabilities are hidden and hard to discover.

c) Because security problems cannot occur in common cases of a program.

4. Which of the following statements are true? ___B_______

a) Security vulnerabilities are the same as program bugs.

b) Finding software vulnerabilities is analogous to finding loopholes in a complex game.

c) Analyzing the security of a system typically requires establishing the threat model.

II. There are many ways a user can be authenticated to a system, e.g., **Something the individual knows, Something the individual possesses, Something the individual is (static biometrics), Something the individual does (dynamic biometrics).** Describe which category the following instances belong to: (2 points)

Face: static biometrics

Smartphone: Individual possesses

Typing rhythm: dynamic biometrics

PIN: individual knows

III. Consider an automated teller machine (ATM) in which users provide a personal identification number (PIN) and a card for account access. Give examples of confidentiality, integrity, and availability requirements associated with the system and, in each case, indicate the degree of importance of the requirement.
(2 points)

The card is the physical possession while the pin is the thing that the individual must know.

IV. Based on the speed of bruteforce password cracking in project 1, explain the suitability of 8-character length passwords. In particular, estimate how long it will take for a single machine to crack.
(1 point)

26 hours for 6 char = 26*26*26 = 17576 hours Since there is 26^8 combos to try

V. Briefly describe the purpose the following instructions and what they do: (3 points)

a) call

Calls the next instruction

b) leave

Release the frame stack of the instruction called by Enter

c) ret

Return to the address that was executed usually memory at the top of the stack

VI. Function calls are implemented using stack. We have shown a function below in C and its assembly code. You are required to

(1) Explain each assembly instruction briefly by inline annotation (example given for the first instruction). (3 points)

(2) Draw the stack frame after instruction 11. (2 points)

```
1 int proc(void)  {
2     int x,y;
3     scanf("%x %x", &y, &x);
4     return x-y;
5 }
```

GCC compiles it into the following assembly code:

```
1 proc:
2     pushl %ebp                  # push (store) the ebp register onto stack
3     movl %esp,%ebp.    #Move the stack pointer to the base pointer
4     subl $24,%esp      #%esp -24
5     addl $-4,%esp      #add -4 to esp
6     leal -4(%ebp),%eax #load address of ebp - 4 into eax
7     pushl %eax #push eax onto the stack
8     leal -8(%ebp),%eax  #load address ebp -8 onto eax
9     pushl %eax #push eax onto the stack
```

10 pushl $.LC0 (Pointer to string "%x %x”) #push .LCO onto the stack

11 call scanf #call scanf instruction

Diagram stack frame at this point

```
| PREV_EBP    | Return Address
+-------------+
|     x       |
+-------------+
|             |
+-------------+
|y            |
+-------------+
|             |
+-------------+
|             |
+-------------+
|             |
+-------------+
|             |
+-------------+

+-------------+
|             |
+-------------+

+-------------+
|     ptr_to_x|
+-------------+

+-------------+
| ptr_to_y    |
+———————+

+-------------+
|LOC          |
+-------------+
```

12 movl -8(%ebp),%eax eax = ebp - 8

13 movl -4(%ebp),%edx eax = ebp - 4

14 subl %eax,%edx edx = edx - eax

15 movl %edx,%eax move edx to eax

16 movl %ebp,%esp move ebp to esp

17 popl %ebp pop the stack and move to ebp

18 ret return to top of the stack

VII. Answer questions below regarding the buffer overflow.

```
/* This is very low quality code.
It is intended to illustrate bad programming practices.
*/
char *getline()
{
    char buf[8];
    char *result;
    gets(buf);
    result = malloc(strlen(buf));
    strcpy(result, buf);
    return(result);
}
```

The above C code gets compiled into the following assembly code below:

```
1 08048524 <getline>:
2 8048524: 55 push %ebp
3 8048525: 89 e5 mov %esp,%ebp
4 8048527: 83 ec 10 sub $0x10,%esp
5 804852a: 56 push %esi
6 804852b: 53 push %ebx
Diagram stack at this point
7 804852c: 83 c4 f4 add $0xfffffff4,%esp
8 804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
9 8048532: 53 push %ebx
10 8048533: e8 74 fe ff ff call 80483ac <_init+0x50>  # gets
Modify diagram to show values at this point
```

The code shows an implementation of a function that reads a line from standard input copies the string to newly allocated storage, and returns a pointer to the result. Consider the following scenario. Procedure getline is called with the return address equal to 0x8048643, register %ebp equal to 0xbffffc94, register %esi equal to 0x1, and register %ebx equal to 0x2. You type in the string “012345678901.” The program terminates with a segmentation fault.

(1) Fill in the diagram below indicating as much as you can about the stack just after executing the instruction at line 6 in the disassembly. Label the quantities stored on the stack (e.g., “Return Address”) on the right, and their hexadecimal values (if known) within the box. Each box represents four bytes. Indicate the position of %ebp. (2 points)

```
+-------------+
| 08 04 86 43 | Return Address
+-------------+
| bf ff fc 94 | contents of buf[8]
+-------------+
|             |
+-------------+
|             |
+-------------+
|             |
+-------------+
|             |
+-------------+
|00 00 00 01  |
+-------------+
|00 00 00 02  |
+-------------+
```

(2) Modify your diagram to show the effect of the call to gets (line 10). (2 points)

```
+-------------+
| 08 03 86 43 | Return Address
+-------------+
| 31 30 39 38 |
+-------------+
| 37 36 35 34 |
+-------------+
|33 32 31 30  |
+-------------+
|             |
+-------------+
|             |
+-------------+
|             |
+-------------+
|             |
+-------------+
|             |
+-------------+
```

(3) To what address does the program attempt to return? (1 point)

```
08 03 86 43
```

(4) What register(s) have corrupted value(s) when getline returns? (1 point)

0x2

(5) Besides the potential for buffer overflow, what two other things are wrong with the code for getline? (2 points)

strcopy

```
malloc(strlen(buf)) plus 1 after the string
```