Homework 1 for CS 165 (Fall 2021)

Due: on ilearn by the end of day on Oct 22 2021

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.

NAME: STEVEN TRUONG

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? **_____a____**

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

b) It is hard to notice.

c) It can happen either locally or over the network.

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

a) Because vulnerabilities are triggered by inputs that are commonly observed in typical workloads.

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

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

4. Which of the following statements are true? **___b, c___**

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: **Something the individual knows**

Smartphone: **Something the individual is (static biometrics)**

Typing rhythm: **Something the individual does (dynamic biometrics)**

PIN: **Something the 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)

Confidentiality: When a user inputs their PIN to the ATM, nobody should be able to discover what the PIN is while it is being sent to the ATM.

Integrity: ATM should always function honestly. Withdrawing X dollars should take away exactly X dollars from the account and depositing Y dollars should always add exactly Y dollars to the account.

Availability Requirements: Having more ATMs available would be convenient for customers.

Most Important To Least Important: Confidentiality > Integrity > Availability

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 (with your own assumption on how many CPU cores can be used). (1 point)

- **There are 26^8 possible password combinations for 8 characters**
- **Assumption: 26 cores are used**
- **My program takes about 0.002 seconds to compare 2 hashes**
- **Therefore it takes $(26^7 * 0.002)/60/60/24 = 185.9$ days to check every possible combination.**
- **Consequently, 8-character passwords are not suitable if they only contain english letters. The time it takes to crack should ideally be years to be secure.**

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

a) call: **Invokes a function. It pushes the return address onto the stack. Then it transfers control to a specified address**

b) leave: **Cleans up the stack space when function ends. It moves the contents of ebp into esp. Then pop caller's ebp off stack and stores it in ebp.**

c) ret: **Transfer control back to the caller. It pops the return address off stack and jumps to the return address.**

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 %esp contents into %ebp**

4 subl $24,%esp **# Reserve 24 bytes on stack**

5 addl $-4,%esp **# Reserve 4 more bytes on stack**

6 leal -4(%ebp),%eax **# Load &x into %eax**

7 pushl %eax **# Push %eax onto stack**

8 leal -8(%ebp),%eax **# Load &y into %eax**

9 pushl %eax **# push %eax onto stack**

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

11 call scanf **# Transfer control to scanf**

Diagram stack frame at this point

...
return address
caller's ebp

&x
&y
“x”
“%x %”

12 movl -8(%ebp),%eax **# Move y into %eax**

13 movl -4(%ebp),%edx **# Move x into %edx**

14 subl %eax,%edx **# Compute x-y and store it in %edx**

15 movl %edx,%eax **# Move contents of %edx into %eax**

16 movl %ebp,%esp **# Move contents of %ebp into %esp**

17 popl %ebp **# Pop caller’s %ebp off stack and store it in %ebp**

18 ret **# Pop return address off stack and jump to it**

VII. Answer questions below regarding the buffer overflow.

```
1 /* This is very low quality code.
2 It is intended to illustrate bad programming practices.
3 */
4 char *getline()
5 {
6     char buf[8];
7     char *result;
8     gets(buf);
9     result = malloc(strlen(buf));
10    strcpy(result, buf);
11    return(result);
12 }
```

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 0x8049643, register %ebp equal to 0xbffffc96, 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 96 43 | Return Address
+-------------+
| bf ff fc 96 | caller's ebp  <------ %ebp
+-------------+
|             |  buf[4..7]
+-------------+
|             |  buf[0..3]
+-------------+
|             |  result
+-------------+
|             |
+-------------+
|00 00 00 01  |  esi
+-------------+
| 00 00 00 02 |  ebx
+-------------+
```

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

```
+-------------+
| 08 04 96 00| Return Address
+-------------+
|31 30 39 38  | caller's ebp  <------ %ebp
+-------------+
|37 36 35 34  |  buf[4..7]
+-------------+
|33 32 31 30   |  buf[0..3]
+-------------+
|             |  result
+-------------+
|             |
+-------------+
|00 00 00 01 |  esi
+-------------+
|00 00 00 02 |  ebx
+-------------+
```

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

08 04 96 00

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

Register %ebp.

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

1. **On line 9, it should be strlen(buf)+1 to include the null terminated character**
2. **strncpy should be used instead of strcpy.**