**Midterm Exam for CS 165 (Fall 2021)**

***Oct 27, 2021 (from 9:30 – 10:50am)***

**Instructions (write down your name):**

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

I. (1 point \*5) Answer the following multiple choice questions (one or more correct answers).

1. Browsers nowadays store our passwords with a click of a button “remember my password”. Which of the following are true about the stored passwords? \_\_\_\_\_\_

a) The passwords are hashed, and therefore even if they are stolen, an attacker needs to bruteforce a large space of all possible passwords.

b) The passwords can be stolen, for example, if the hard drive is stolen.

c) The passwords are encrypted, for example, using a key derived from login credentials.

2. Which of the following are true about x86 assemble code? \_\_\_\_\_\_

a) AT&T and Intel conventions are simply different ways of representing the same instructions

b) There are often multiple different instructions that can achieve the same goal

c) The x86 instructions are fixed length.

3. Which of the following are true about the stack? \_\_\_\_\_\_\_

a. Stack is part of a process’s memory address space.

b. Stack stores the return address, local variables, and all the constants we use in a program.

c. A return from a function call always causes more stack space to be occupied.

4. Which of the following are true about control flow hijack defenses? \_\_\_\_\_\_\_

a. Control flow integrity (CFI) is a strictly better defense than canary because it defends what canary can defend and more.

b. ROP can never bypass the simple defenses such as canary, ASLR, and DEP.

c. ASLR is capable of defending against ROP.

5. Which of the following are true about the blind ROP attack? \_\_\_\_\_\_\_

a. The attacker needs to first find enough gadgets for write() so it can see the binary.

b. Crash gadget and stop gadget are relatively easy to find.

c. The attack leverages the fact x86 instructions can overlap (i.e., one can jump to the middle of an instruction and the CPU can recognize it as a different yet valid instruction).

II. (1 point) Please write two other ways to achieve the same goal of “mov edx, 0” (Intel convention)

III. (1.5 points) In a successful stack buffer overflow attack, what security properties can be broken out of confidentiality, integrity, and availability? Provide an explanation for each property.

IV. (1.5 points) We have seen unsafe in standard C library functions such as fgets(), strcpy(), and sprintf(). Show the safer alternatives of these three functions and explain why they are safer.

V. (2 points) In control flow hijacking, it is always the case that some critical control data (e.g., return address, function pointer) got maliciously modified. Consider the following plausible defense: instead of storing such control data in a writable memory page, can we design a memory page protection mechanism that allows write-once and read-only afterwards. This will presumably prevent any malicious overwrite of the control data. Discuss any limitations or weaknesses of the approach.

VI. (4 points) The following code takes two strings as arguments and returns a pointer to a new string that represents their concatenation. n1 and n2 are supplied by the caller of the function and can be arbitrary values. s1 and s2 on the other hand are not NULL pointers.

1 char \*concat(char s1[], int n1, char s2[], int n2)

2 {

3 int i, j;

4 int n = n1 + n2;

5 char \*s;

6

7 if (n1 < 0 || n2 < 0 || n <= 0) return 0;

8

9 s = malloc(n);

10

11 for (i=0; s1[i] != ’\0’; ++i)

12 s[i] = s1[i];

13

14 for (j=0; s2[j] != ’\0’; ++j)

15 s[i+j] = s2[j];

16

17 s[i+j] = ’\0’;

18 return s;

19 }

a. Please identify at least three places where potential memory safety problems can occur (crashing the program, overwriting the buffer, etc.). (1.5 points)

b. Please rewrite the program to eliminate the problem. (1.5 points)

c. What does this example tell you regarding the best practices in software engineering, e.g., what principles/assumptions should be followed to minimize or eliminate problems like this. (1 point)

VII. (4 points) The following example shows a typical shell code that an attacker would like to execute in a stack buffer overflow (due to a strcpy) to turn a vulnerable program into a shell.

int main (int argc, char \*argv[])

{

char \*sh;

char \*args[2];

sh = "/bin/sh;

args[0] = sh;

args[1] = NULL;

execve (sh, args, NULL);

}

1. **Desired shellcode code in C**

nop

nop //end of nop sled

jmp find //jump to end of code

cont: pop %esi //pop address of sh off stack into %esi

xor %eax, %eax //zero contents of EAX

mov %al, 0x7(%esi) //copy zero byte to end of string sh (%esi)

lea (%esi), %ebx //load address of sh (%esi) into %ebx

mov %ebx,0x8(%esi) //save address of sh in args [0] (%esi+8)

mov %eax,0xc(%esi) //copy zero to args[1] (%esi+c)

mov $0xb,%al //copy execve syscall number (11) to AL

mov %esi,%ebx //copy address of sh (%esi) into %ebx

lea 0x8(%esi),%ecx //copy address of args (%esi+8) to %ecx

lea 0xc(%esi),%edx //copy address of args[1] (%esi+c) to %edx

int $0x80 //software interrupt to execute syscall

find: call cont //call cont which saves next address on stack

sh: .string "/bin/sh " //string constant

args: .long 0 //space used for args array

.long 0 //args[1] and also NULL for env array

**(b) Desired shellcode in x86 assembly (injected onto the stack), in AT&T convention**

The shellcode shown in (b) assumes that the execve() system call will not return. If it is the case, then we do not need to worry. Nevertheless, it is possible that it may fail. In that case, we would like to consider adding another system of exit(0) to allow the program to exit gracefully if execve() does return an error of -1. This is better than having the program crash. Show a modified shellcode that would accomplish this. (hint: pay attention to how we might encode the arguments and invoke the system call).