Computer Science 4271
Spring 2022
Midterm exam 1 (solutions)
February 22nd, 2022
Time Limit: 75 minutes, 9:45am-11:00am

- Before starting the exam, you can fill out your name and other information of this page, but don't open the exam until you are directed to start. Don't put any of your answers on this page.
- This exam contains 6 pages (including this cover page) and 4 questions. Once we tell you to start, please check that no pages are missing.
- You may use any textbooks, notes, or printouts you wish during the exam, but you may not use any electronic devices: no calculators, smart phones, laptops, etc.
- You may ask clarifying questions of the instructor or TAs, but no communication with other students is allowed during the exam.
- Please read all questions carefully before answering them. Remember that we can only grade what you write on the exam, so it's in your interest to show your work and explain your thinking.
- By signing below you certify that you agree to follow the rules of the exam, and that the answers on this exam are your own work only.

The exam will end promptly at 11:00am. Good luck!

Your name (print): _____

Your UMN email/X.500: _____`@umn.edu`

Number of rows ahead of you: _____ Number of seats to your left: _____

Sign and date: _____

Question	Points	Score
1	28	
2	20	
3	28	
4	24	
Total:	100	

1. (28 points) Multiple choice. Each question has only one correct answer: circle its letter.

   (a) Which of these `printf` format specifiers reads and prints a 64-bit value on Linux/x86-64?

   A. `%lx` B. `%d` C. `%c` D. `%%` E. `%x`

   `%d` and `%x` both print in an `int`-sized value, which is 32 bits on Linux/x86-64, in decimal and hexadecimal respectively. `%lx`, by comparison, uses hexadecimal to print a value the size of a `long`, which is 64 bits. `%c` prints a single byte (8 bits) as a character, while `%%` prints a literal percent sign without reading any value.

   (b) Which of these mechanisms primarily exists to stop repudiation threats?

   A. ASLR B. W⊕X C. encryption **D. logging** E. passwords

   ASLR and W⊕X are mechanisms to block low-level attacks, which are most directly associated with privilege escalation thought they could be part of other kinds of threats as well. Encryption and passwords are primarily used to protect against information disclosure and spoofing respectively.

   (c) Which of these values is **not** found in the stack on Linux/x86-64?

   A. A global variable
   B. An environment variable
   C. A local array variable in a function
   D. A function return address
   E. The program's command-line arguments

   Environment variables and the program's command-line arguments are placed in the oldest (highest-addressed) part of the stack by the OS before the program starts executing. Local variables are stored within a function's stack frame, and return addresses lie at the boundaries of stack frames. Global variables, by comparison, have their locations chosen by the compiler in a memory area called the data segment, which is far from the stack.

   (d) In the architecture of our system, software component A requests some information from component B and gets a response. What arrowheads should we draw on the edge connecting boxes for A and B?

   A. Neither, because arrowheads make the diagram cluttered.
   B. Only from A to B, because the response could not be part of a threat.
   C. Only from B to A, because the request could not be part of a threat.
   D. Both, because both the request and the response could be part of a threat.

   Both the request and the response are data flows that might be part of an attack.

   (e) Which of these is **not** a kind of code-reuse attack?

   A. Call-oriented programming
   B. Jump-oriented programming
   C. Return-to-libc
   D. Integer overflow
   E. Return-oriented programming

   Return-oriented programming (ROP) is the most famous kind of code-reuse attack. Return-to-libc attacks are a simpler special case of ROP where the code that is reused from library is an entire function. Call-oriented programming and jump-oriented programming are names

given to variants of ROP where the gadgets are connected by indirect call instructions or indirect jump instructions respectively. Integer overflow, by contrast, is a bug rather than any kind of attack technique.

(f) Which of these is **not** either a synonym for a kind of defense, or a hardware mechanism that implements it?

A. W⊕X B. NX bit C. DEP **D. ASLR** E. XD bit

It seems like the wording of this question confused many students, sorry about that. All of the answers are either names for defenses, or for hardware mechanisms used to implement defenses. But four out of the five answers all relate to what is basically the same defense, in that they are either synonyms for the name of the defense, or hardware mechanisms that implement the same defense. Namely, DEP (Data Execution Prevention) is another name for a W⊕X (write xor execute) defense, and the NX bit and the XD bit are names for hardware mechanisms that implement W⊕X in AMD and Intel processors respectively (in fact for compatibility they are the same bit position). ASLR is the odd one out because while it is also the name of a defense, it is a different defense not synonymous with W⊕X/DEP.

(g) A buffer overflow within a `struct` located on the heap might overwrite all of the following **except**:

A. A field located later in the same structure
B. A pointer used to locate another heap object
C. A field in another heap object
D. A local variable
E. An area that will be reused in a later heap allocation

Local variables are stored on the stack, which is far away from the heap so it would not be overwritten by an overflow within the heap. But the other answers all refer to data in the heap that might be overwritten.

2. (20 points) C programming bugs.

   Each of the following snippets of C code illustrates an erroneous code pattern that might cause a program to crash. For each code example, give a value of the variable `x` that would cause the dangerous behavior, and the name of the bug. (Assume that `malloc` never fails in these examples.)

   (a)
   ```
   char *p = malloc(10);
   if (x >= 10)
       free(p);
   if (x < 11)
       free(p);
   ```

   Value of `x`: **10** Bug name: **double free**

   (b)
   ```
   int a[20], i;
   for (i = 0; i < x; i++)
       a[i] = 12;
   ```

   Value of `x`: **30** Bug name: **buffer overflow**

   (c)
   ```
   char *p = malloc(20);
   if (x < 20)
       free(p);
   p[0] = p[1];
   ```

   Value of `x`: **10** Bug name: **use after free**

   (d)
   ```
   *x = 30;
   ```

   Value of `x`: **0** Bug name: **null pointer dereference**

   (e)
   ```
   printf(x, 36, "carrot");
   ```

   Value of `x`: `"%s%s%s%s"` Bug name: **format string injection**

3. (28 points) Buffer overflow with machine code.

   The following function from a Linux/x86-64 program has a buffer overflow vulnerability. (It uses the function `strlcpy` which is like `strcpy` but taking a maximum size parameter. Unfortunately the maximum size parameter is set too large.)

   Answer the following questions about details of how the code was compiled that relate to how the vulnerability could be attacked. Write numbers in your answers in decimal. Some of your answers should be expressed in relation to the variable R, which represents the value held in both the `%rbp` and `%rsp` registers after the instruction numbered `2:`. These answers should be something like $R + 17$ or $R - 4000$.

   Below are excerpts of the relevant code in C and assembly language.

```
void f(char *p) {
    char buf[32];
    strlcpy(buf, p, MAX_SIZE);
}
```

```
 1:   push   %rbp
 2:   mov    %rsp,%rbp
 3:   sub    $0x30,%rsp
 4:   mov    %rdi,-0x28(%rbp)
 5:   mov    -0x28(%rbp),%rcx
 6:   lea    -0x20(%rbp),%rax
 7:   mov    $0x50,%edx
 8:   mov    %rcx,%rsi
 9:   mov    %rax,%rdi
10:   call   strlcpy
11:   leave
12:   ret
```

   (a) What is the location of the function's return address, relative to R?
   R + 8

   (b) What is the starting location of the buffer `buf`, relative to R?
   R - 32

   (c) What is the value of the third argument to `strlcpy` (i.e., of the macro `MAX_SIZE`)?
   80

   (d) If the bytes within the buffer are numbered starting at 0, what is the range of byte values that would overwrite the return address?
   40–47

4. (24 points) Short answers. Each of the following questions has a brief answer.

   (a) An `unsigned short` variable in C is 16 bits long, and holds a value between 0 and 65535. Suppose that `us16` is an `unsigned short` containing 16. Give, in hexadecimal, a value for an unsigned short `usx` such that `us16 * usx < usx`, according to the arithmetic rules of C.

   Many possible answers including `0x1000`, `0x4321`, or `0xffff`

   (b) What is the name for the kind of picture with boxes and arrows we use to describe the structure of a system in threat modeling?

   A **data-flow diagram**

   (c) The functions `strcpy` and `memcpy` both copy memory regions, but an important difference is that `strcpy` treats one particular byte value differently; what is it?

   0, *which* `strcpy` *treats as a terminator*

   (d) In 2015, it was revealed that computer systems of the US government's Office of Personnel Management (OPM) had been attacked over the previous years, apparently allowing the attackers to obtain information about millions of government employees such as fingerprints and security clearance forms. If the OPM had performed threat modeling under the STRIDE taxonomy, what kind of threat was the possibility of this kind of attack?

   information disclosure

   (e) Trying to access memory via a pointer that is outside any valid memory region will cause a program to immediately crash with a segfault. If this is the only dangerous behavior that a bug allows, the STRIDE threat it is most relevant to is:

   denial of service

   (f) The instructions in a NOP sled don't do anything. So why is a NOP sled useful to an attacker?

   It allows a **larger target** *for a jump to shellcode. If the NOP sled comes before the shellcode, then jumping to any location in the NOP sled will also lead to the shellcode getting executed.*