Computer Science 5271
Fall 2021
Midterm exam
October 25th, 2021
Time Limit: 75 minutes, 2:30pm-3:45pm

- 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 7 pages (including this cover page) and 4 questions. Once we tell you to start, please check that no pages are missing, and write your username in the upper-right corner of each subsequent page.
- 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. But write only on the printed sides of exam pages.
- 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 3:45pm. 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	30	
2	20	
3	30	
4	20	
Total:	100	

1. (30 points) Multiple choice. Each question has only one correct answer: circle its letter.

   (a) Suppose that the UMN IT managers are considering increasing the minimum password length from 16 to 24 characters, to improve resistance to offline guessing attacks. Suppose a user has selected a 24-character password in which each character is any uppercase letter, lowercase letter, or digit independently and with equal probability (e.g., `SQMq1bvXfddhhGH24W2DV2fv`). Which of these formulas gives the total number of possible passwords?

      A. $26^{24} + 26^{24} + 10^{24} \approx 2^{114}$
      B. $24^{26} + 24^{26} + 24^{10} \approx 2^{120}$
      C. $(26 + 26 + 10)^{24} \approx 2^{143}$
      D. $24^{(26+26+10)} \approx 2^{284}$
      E. $26^{24} \cdot 26^{24} \cdot 10^{24} \approx 2^{305}$

   (b) An attacker with access to a million devices, each of which can check a billion passwords per second, can try about 2^{75} passwords in a year. Whichever of the computations in the previous question is correct, this suggests that an attack against a password chosen according to the process in that question would be infeasible. However, many objections might still be raised to a policy of requiring 24-character passwords containing letters and digits. Which of the following is **not** a reasonable criticism?

      A. Users will make more typos with 24-character passwords.
      B. Users will have a hard time remembering 24-character passwords.
      C. A password cannot be secure unless it contains punctuation characters like `(` or `@`.
      D. Users will not generate their passwords uniformly at random.
      E. A 16-character length would be enough to resist guessing attacks.

   (c) The following properties are true for all values of 32-bit two's complement integer values `x` and `y` (i.e., C `int`s), **except**:

      A. `x + 1 != x`
      B. `5*x == (x << 2) + x`
      C. `x * y == y * x`
      D. `x + -x == 0`
      E. `-x <= x`

   (d) Consider the set of positive integers where the relationship $a \sqsubseteq b$ is defined to hold when a divides b evenly (i.e., there is another positive integer c such that $ac = b$). This operation forms a partial order. If we want to make it a lattice, what operation should be the meet $a \sqcap b$?

      A. $\min(a, b)$
      B. $\max(a, b)$
      C. $\lfloor (a + b)/2 \rfloor$ ($\lfloor x \rfloor$ represents rounding down to the nearest integer)
      D. $\gcd(a, b)$
      E. $a + b$

   (e) This number x is the multiplicative inverse of 3 mod 2^{32}, i.e. $3 \cdot x \equiv 1 \pmod{2^{32}}$. (Possibly relevant facts: `3*x = x + (x << 1)`; `0xffffffff = 3*0x55555555`; `3*6667 = 20001`)

      A. `0x33333333` B. `0x55555555` C. `0x80000003` D. `0xaaaaaaab` E. `0xfffffffd`

(f) Compared to x86-32, x86-64 has several features that make things easier for defenders and harder for attackers. Which of these is **not** such a difference?

A. The stack grows upwards, so a buffer overflow can't overwrite the return address
B. Expanded RIP-relative addressing and more registers allow more efficient PIC
C. Arguments in registers cannot be corrupted via a stack buffer overflow
D. A larger address space allows ASLR to have more entropy
E. All valid x86-64 user-space addresses contain null bytes

(g) When logging on to the course Canvas page, you must use a registered smartphone or security token in addition to providing a password. This is an example of:

A. Multi-factor authentication
B. Separation of duty
C. A CAPTCHA
D. Single sign-on
E. Biometric authentication

(h) Suppose you want to use a format-string vulnerability to cause a program to output a lot of output, say n characters, with a short format string. About how long of a format string is required?

A. 2 characters
B. $\log_2(\log_2 n)$ characters
C. $\log_{10} n$ characters
D. $\sqrt{n}$ characters
E. $(4/16) \cdot n$ characters

(i) On the CSE Labs machines, the standard umask is `0077`, which causes files to be created without permissions for other users. This is an example of which of the following secure design principles?

A. Complete mediation
B. Open design
C. Fail-safe defaults
D. Least privilege
E. Economy of mechanism

(j) For an attack to be possible, a use-after-free bug usually needs to involve a memory region that was first allocated with one type ("type 1") being reused with a different type ("type 2"). Suppose there is an attack possible where a value is written with type 1 before the value is reallocated, then read with type 2 later. For this attack to work, what other problem must the program have?

A. Null pointer dereference
B. Format string vulnerability
C. Race condition
D. Infinite recursion
E. Reading uninitialized data

2. (20 points) Matching vulnerability types.

   On the left side are ten examples of C code patterns that represent various security-relevant bugs. Fill in the blank next to each with a letter corresponding to the name of a vulnerability type from the right side. Every vulnerability type is used exactly once, except that "Buffer Overflow" is used exactly two times. You can assume any variables named `evil` or `bad` might be under the control of an attacker, and have not had relevant security checks performed. Ellipses . . . represent omitted code.

   (a) ____

   ```
   num_objs = evil;
   p = malloc(num_objs * sizeof(obj));
   ```

   (b) ____

   ```
   p = q = malloc(...);
   free(p);
   q->field = evil;
   ```

   (c) ____

   ```
   char buf[30];
   for (i = 0; i < bad; i++)
       buf[i] = evil[i];
   ```

   (d) ____ `system("cp a b");`

   (e) ____

   ```
   free(p);
   ...
   free(p);
   ```

   (f) ____ `printf(evil, 42);`

   (g) ____ `gets(buf);`

   (h) ____

   ```
   if (access(fname, R_OK) == 0)
       fd = open(fname, O_RDONLY);
   ```

   (i) ____

   ```
   char path[] = "/tmp/x.XXXXXXX";
   mktemp(path);
   fd = open(path, O_WRONLY|O_CREAT);
   ```

   (j) ____

   ```
   snprintf(buf, sizeof(buf),
            "/dir/%s", evil);
   fd = open(buf, O_RDWR);
   ```

A. Use after free

B. TOCTTOU race

C. Integer overflow

D. Format string vulnerability

E. Directory traversal

F. File creation race

G. Double free

H. Insecure PATH dependency

I. Buffer overflow

I. Buffer overflow

3. (30 points) Buffer overflow attack.

   The following function from a Linux/x86-64 program has a buffer overflow vulnerability. In this question, you'll figure out the stack layout of the function and what data should be used in the overflow to build a successful attack.

   Specifically, assume that normally the function would return to the address `0x4011fb`, and that the argument `s` points to a string under the attacker's control. Your goal as the attacker it to make the execution instead jump to the address `0x4d5271`, where you have arranged for some shellcode to exist. Below are excerpts of the relevant code in C and assembly language.

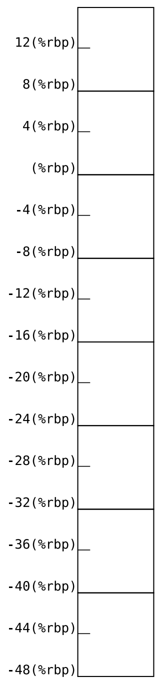
```
int global_fd = -1;

void func(char *s) {
    int fd = 0;
    char buf[16];
    fd = open("/tmp/foo", O_RDONLY);
    strcpy(buf, s);
    if (fd != global_fd) {
        exit(1);
    }
}

.LC0: .string "/tmp/foo"
func: pushq   %rbp
      movq    %rsp, %rbp
      subq    $48, %rsp
      movq    %rdi, -40(%rbp)
      movl    $0, -4(%rbp)
```

```
          movl    $0, %esi
          movl    $.LC0, %edi
          movl    $0, %eax
          call    open
          movl    %eax, -4(%rbp)
          movq    -40(%rbp), %rdx
          leaq    -32(%rbp), %rax
          movq    %rdx, %rsi
      ==> movq    %rax, %rdi
          call    strcpy
          movl    global_fd(%rip), %eax
          cmpl    %eax, -4(%rbp)
          je      .L3
          movl    $1, %edi
          call    exit
    .L3:  nop
          leave
          ret
```

   Use this code to answer the questions on the following page.

   Recall that `strcpy` copies a sequence of characters pointer to by its second argument to the location pointed to by its first argument, up to a null terminator. The `open` system call opens the file specified in its first arguments for the operation(s) specified in its second argument, returning a non-negative file descriptor if successful or -1 on an error. Assume that the variable `global_fd` still has the value -1 when the attack occurs.

(a) First, let's draw a diagram of the function's stack frame layout. In particular, draw the layout right before the call to `strcpy`, at the location marked with an arrow in the assembly code. On the left is a blank picture of a stack frame, broken into 8-byte segments and labelled by offsets relative to the frame pointer `%rbp`. On the right is a list of descriptions of contents that might appear in each segment. Fill in each box on the left with a letter from the right. Some descriptions might be used more than once, others not at all.



A. Unused/padding

B. `global_fd` and unused/padding

C. Stack canary

D. Return address

E. Saved `%rbx`

F. Saved `%rbp`

G. `buf[0 .. 7]`

H. `buf[8 .. 15]`

I. `s`

J. `fd` and unused/padding

(b) Now, show what contents for the string `s` should be used to create a successful attack that hijacks control flow. Each blank below represents one character in the attack string, in order of increasing address. Fill in each blank with one character. You can write printable ASCII characters like letters as themselves, and for non-printable characters use C escapes like `\0` for null, `\n` for newline, or `\xfa` for a byte with hex value `0xfa`. You may not need to use every blank. The first 16 characters of the string that will go inside the bounds of `buf` won't be part of the attack, so we've filled them in with regular letters.

```
A    A    A    A    A    A    A    A     B    B    B    B    B    B    B    B

___  ___  ___  ___  ___  ___  ___  ___   ___  ___  ___  ___  ___  ___  ___  ___

___  ___  ___  ___  ___  ___  ___  ___   ___  ___  ___  ___  ___  ___  ___  ___

___  ___  ___  ___  ___  ___  ___  ___   ___  ___  ___  ___  ___  ___  ___  ___
```

4. (20 points) Matching definitions and concepts. Fill in each blank with the letter of the corresponding answer. Each answer is used exactly once.

   (a) ____ A value that would cause copying to stop

   (b) ____ A bug allowing memory reuse with a different type

   (c) ____ Not connecting networks at different levels

   (d) ____ An attack that can be in binary or interpreted software

   (e) ____ An instruction that has no side-effects

   (f) ____ A memory permissions bit used to implement W $\oplus$ X

   (g) ____ A program that runs with the identity of its file owner

   (h) ____ The group of users who can become the superuser

   (i) ____ Slightly modifying an OS kernel to run better in a VM

   (j) ____ An abstract model for MLS confidentiality

   (k) ____ Privilege based on identity, not a capability

   (l) ____ A measure of the uncertainty in a probability distribution

   (m) ____ An attack accessing unintended parts of the filesystem

   (n) ____ The standard username for UID 0

   (o) ____ Binary-only software without symbol information

   (p) ____ For example, $\subseteq$ on sets

   (q) ____ A mail server designed for increased security

   (r) ____ A defense that changes the base address of a memory region

   (s) ____ An invariant true about the results of a function

   (t) ____ Unintended communication with cooperating sender and receiver

A. air gap B. ambient authority C. ASLR D. Bell-LaPadula E. code injection
F. COTS G. covert channel H. directory traversal I. entropy J. NOP K. NX
L. paravirtualization M. partial order N. postcondition O. Postfix P. `root`
Q. setuid R. terminator canary S. use after free T. `wheel`