Computer Science 5271
Spring 2019
Midterm exam
March 5th, 2019
Time Limit: 75 minutes, 4:00pm-5:15pm

- This exam contains 8 pages (including this cover page) and 4 questions. Once we tell you to start, please check that no pages are missing.
- 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.
- 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 TA, 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 5:15pm. 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	24	
3	26	
4	20	
Total:	100	

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

   (a) Which of these "terminators" is commonly part of a terminator canary?

   A. The C comment terminator `*/`
   B. The C statement terminator `;`
   C. The HTML line terminator `<br>`
   D. The file input terminator Ctrl-D
   E. The C string terminator `\0`

   (b) In a 32-bit Linux/x86 program, which of these objects would have the lowest address (numerically least when considered as unsigned)?

   A. An environment variable
   B. The program name in `argv[0]`
   C. A command-line argument in `argv[1]`
   D. A local `float` variable in a function called by `main`
   E. A local `char` array in `main`

   (c) If UMN students begin to choose easier-to-guess passwords after they are required to use two-factor authentication with a smartphone in addition to their password, this would be an example of:

   A. defense in depth
   B. false positives
   C. compromise recording
   D. risk compensation
   E. economy of mechanism

   (d) When configured as recommended by the manufacturer, a fingerprint recognition system had a false rejection (false negative) rate of 6%, and a false acceptance (false positive) rate of 0.1%. What does this imply about the system's equal error rate?

   A. The EER is less than 0.1%
   B. The EER is between 0.1% and 6%
   C. The EER is more than 6%
   D. The EER could be anything from 0% to 100%
   E. EER is not meaningful for fingerprint recognition

   (e) This function by default will securely abort the program if there is not enough space in the destination buffer:

   A. `str5cpy` B. `strncpy` C. `strcpy_s` D. `strcpy` E. `strlcpy`

   (f) In the Unix access control model, subjects are primarily identified by their:

   A. email address
   B. username
   C. executable inode
   D. program name
   E. UID

(g) One way to ensure that there is no information flow between two computer systems is to ensure that there is no connection between them. This lack of a connection is called a(n):

A. digital no-mans land
B. cyber moat
C. whitespace
D. DMZ
E. air gap

(h) Any of these technologies might be used to implement a sandbox for privilege separation, **except**:

A. SFI
B. `chroot`
C. QEMU virtual machine
D. FreeBSD jail
E. ASLR

(i) Which of the following is **not** always true, when the variables are interpreted as 32-bit unsigned `int`s in C?

A. `x*y` is odd, if both `x` and `y` are odd
B. `x*y == y*x`
C. `x + x + x + x == 4*x`
D. `16*x >= x`
E. `x + (-x) == 0`

(j) A confused deputy is able to take an unsafe action because of its:

A. certified capability
B. attested authentication
C. persistent privilege
D. ambient authority
E. trusted trust

2. (24 points) Unix file permissions. The course CSci 5721 is being taught by Prof. Forgetful (user name `forgetfu`) and the TA is Pat (`patx1792`); the permissions group `S19C5721` for the course staff consists of just the two of them. Files for the course are stored in a directory named `website` on a Linux system, which has the owner `forgetfu`, group owner `S19C5721`, and permissions `02775`:

```
% ls -ld website
drwxrwsr-x 5 forgetfu S19C5721 4096 Feb  4 12:31 website
```

Prof. Forgetful created a homework assignment in a file named `ex3.pdf` that is due on Friday, but the professor forgot to update the permissions on the assignment so that students can read it. Right now the permissions on this file inside the `website` directory are `0640`, again with owner `forgetfu` and group owner `S19C5721`:

```
% ls -l website/ex3.pdf
-rw-r----- 1 forgetfu S19C5271 103079 Feb  7 08:37 website/ex3.pdf
```

Prof. Forgetful is travelling and has not been responding to email, so it falls to Pat to make the assignment available to students; students in the class are members of a different group `students` but not of `S19C5721`. Your task is to give Pat a sequence of Unix commands that can be executed just by Pat, without help from Prof. Forgetful or the sysadmins, to make it so that the `website` directory contains a file named `ex3.pdf` with the same contents as the current one, but that students can access to do the assignment. However students should not be able to modify the assignment; still only Prof. Forgetful and Pat should be allowed to do that.

Specifically, choose your commands from among a list that appears on the next page. All the commands will run in the `website` directory. Pat's `umask` is set to `077`, so plain files newly created by Pat will have permissions `0600`.

You don't need to use all of the commands, or to fill in all of the blanks for the sequence, and you can use a command more than once if you would like.

Some reminders about the commands involved. `chown` changes the user owner of a file. `chgrp` changes the group owner of a file. `cp` copies the contents from a source file to a destination file, newly creating the destination file if it does not already exist. `mv` renames (moves) a file to a different name within a directory. `rm` removes a file from a directory. `chmod` changes the permissions on a file. All permissions are expressed in octal. Before the process starts, there is no file `ex3-copy.pdf` in the directory.

Here are the allowed commands, each indicated by a capital letter:

```
A. chown patx1792 ex3.pdf
B. chgrp students ex3.pdf
C. chgrp S19C5271 ex3.pdf
D. cp ex3.pdf ex3-copy.pdf
E. mv ex3.pdf ex3-copy.pdf
F. cp ex3-copy.pdf ex3.pdf
G. mv ex3-copy.pdf ex3.pdf
H. rm ex3.pdf
I. rm ex3-copy.pdf
J. chmod 0666 ex3.pdf
K. chmod 0664 ex3.pdf
L. chmod 0644 ex3.pdf
M. chmod 0640 ex3.pdf
N. chmod 0600 ex3.pdf
```

Write your sequence of commands below by writing one letter per blank, as many as needed. Optionally (e.g., to help get partial credit), you can use the space on the right to write other notes about what the commands are doing.

(a) ____

(b) ____

(c) ____

(d) ____

(e) ____

(f) ____

(g) ____

(h) ____

(i) ____

(j) ____

(k) ____

(l) ____

3. (26 points) Control-flow Integrity.

   This question concerns the following C program. It has a blatant buffer-overflow vulnerability, using which an attacker who controls the `controlled` variables through the environment can cause the program to execute various dangerous code. We will try to block these attacks using CFI. The comments in the code label *return sites*: the points in the code after a function call that a return address normally points to.

```
char *controlled1, *controlled2;
int privileged = 0;
void f(void) {
    int is_allowed = (privileged > 0);
    char small_buf[2];
    strcpy(small_buf, controlled1);
    if (is_allowed) {
        printf("Dangeous operation is allowed");
        /* printf_ret1: */
        system(controlled2);
        /* system_ret1: */
    }
}
void dangerous2(void) {
    system("bash");
    /* system_ret2: */
}
void dangerous3(void) {
    system("/bin/rootshell");
    /* system_ret3: */
}
int main(int argc, char **argv) {
    int safe = (1 == 1);
    controlled1 = getenv("ATTACKER1");
    controlled2 = getenv("ATTACKER2");
    if (safe) {
        f(); /* f_ret1: */
    } else {
        f(); /* f_ret2: */
        dangerous3(); /* dangerous3_ret1: */
    }
    return 0;
}
```

Specifically we will consider 4 attacks made possible by the buffer overflow of the buffer `small_buf` in function `f`:

Attack 1 Set `f`'s variable `is_allowed` to a non-zero value

Attack 2 Replace `f`'s return address with `dangerous2`

Attack 3 Replace `f`'s return address with `f_ret2`

Attack 4 Replace `f`'s return address with `printf_ret1`

We will also consider 5 levels of CFI protection for return instructions, in order of increasing strength:

CFI F No CFI protection

CFI D A return can go to any function entry or return site in the program

CFI C A return can go to any return site in the program

CFI B A return can go to any return site of that function

CFI A A protected shadow stack for return addresses

In the following table, the rows represent the different levels of CFI protection. The first set of blank columns correspond to three different function entry points in the program. The second set of blank columns correspond to six different return sites in the program. The third set of blank columns correspond to the four different potential attacks. Fill in the entries for function entry and return sites by putting an A for "allowed" in those entries for places the CFI protection would allow the return to jump to; leave an entry blank if a return to that location would not be allowed. Fill in the entries for attacks with P for "possible" if the attack would work under this CFI protection, or I for "impossible" if the attack would be blocked. We have filled the first row in for you.

	`f`	`dangerous2`	`dangerous3`	`printf_ret1`	`system_ret1`	`system_ret2`	`f_ret1`	`f_ret2`	`dangerous3_ret3`	Attack 1	Attack 2	Attack 3	Attack 4
CFI F	A	A	A	A	A	A	A	A	A	P	P	P	P
CFI D													
CFI C													
CFI B													
CFI A													

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) ____ Where users choose the permissions on their files

   (b) ____ Random testing trying to find crashes

   (c) ____ Ensures no memory region is usable for shellcode

   (d) ____ Protected place to store return addresses

   (e) ____ MTA originally written by Eric Allman

   (f) ____ Unit of code for ROP

   (g) ____ Controls how shared libraries are loaded

   (h) ____ Means that anything could happen

   (i) ____ Technique for guessing a password

   (j) ____ Memory region holding zero-initialized globals

   (k) ____ Benefit of a security mechanism minus its cost

   (l) ____ Dividing software into pieces with clear interfaces

   (m) ____ Directory where the list of users is stored

   (n) ____ Controls how shell commands are parsed

   (o) ____ MTA primarily written by Daniel Bernstein

   (p) ____ Removed from C standard as too insecure

   (q) ____ Requires that a file not already exist

   (r) ____ Directory commonly marked with the sticky bit

   (s) ____ Special case of ROP using complete functions

   (t) ____ Shows trade-off between true and false positives

A. `.bss` B. DAC C. dictionary attack D. `/etc` E. fuzzing F. gadget G. `gets` H. `IFS` I. `LD_LIBRARY_PATH` J. modularity K. net risk reduction L. `O_EXCL` M. qmail N. return to `libc` O. ROC curve P. Sendmail Q. shadow stack R. `/tmp` S. undefined behavior T. W⊕X