Computer Science 4271
Fall 2022
Midterm exam 2
November 15th, 2022
Time Limit: 75 minutes, 4:00pm-5:15pm

- 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.
- 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 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	25	
2	18	
3	30	
4	27	
Total:	100	

1. (25 points) Manipulating Unix permissions.

   You are the vice president (username `veep4271`) of the Student Security Club at a large Midwestern university, whose website is hosted on a shared Unix computer system reminiscent of CSE Labs. You've been working recently with the club president (username `patx1600`) on a large public event for your club, and the president wrote up a detailed web page for the club home page to advertise the event. Unfortunately, the president did not set the web page permissions correctly before leaving on a hiking trip for the weekend, and you have to fix them.

   The officers of the club, including you and the president, are members of a Unix group named `ssecclub`. All thousands of students using the system, including you and the president, are members of a Unix group named `student`, and this is your default group. The files for the club website are stored in a directory named `website` on a Linux system, which has the owner `patx1600`, group owner `ssecclub`, and permissions `0775`:

   ```
   % ls -ld website
   drwxrwxr-x 5 patx1600 ssecclub 4096 Nov  4 12:31 website
   ```

   The web page for the event is a file named `event.html` stored inside the `website` directory. Right now the permissions on this file are `0640`, with owner `patx1600` and group `student`.

   ```
   % ls -l website/event.html
   -rw-r----- 1 patx1600 ssecclub 10307 Nov  7 08:37 website/event.html
   ```

   The web server runs as a process that is not in either of the `ssecclub` or `student` user groups, so in order to be shown properly, the `event.html` file needs to be readable by any user on the system. Your task is to give a sequence of Unix commands that can be executed just by you, without help from the president or the sysadmins, to make it so that the `website` directory contains a file named `event.html` with the same contents as the current one, but that the web server can display. However the ability to modify the web page in the future should be limited to the officers of the club.

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

   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 commands start, there is no file `copy.html` in the directory.

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

```
A. chown patx1600 event.html
B. chgrp students event.html
C. chgrp ssecclub event.html
D. cp event.html copy.html
E. mv event.html copy.html
F. cp copy.html event.html
G. mv copy.html event.html
H. rm event.html
I. rm copy.html
J. chmod 0666 event.html
K. chmod 0664 event.html
L. chmod 0644 event.html
M. chmod 0640 event.html
N. chmod 0600 event.html
```

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) ____

2. (18 points) Cross-site scripting in different contexts.

   Below is the source code for a web page implemented using HTML, CSS, and JavaScript. At each of the locations marked by a circled letters ⓐ through ⓕ, untrusted text can be inserted. Because no filtering or sanitization of the inserted text is performed, all the locations are vulnerable to cross-site scripting. However different payloads are required at different locations. At each location, the injected text should keep the syntax correct, but call the JavaScript `alert` method with a string starting with `XSS`. Match the six payloads on the right with the correct locations on the left by writing the number of the payload in the blank. In our solution, each payload is used exactly once.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Capture the flag event Thursday</title>
    <style type="text/css">
      h1 { background-color: ⓐ; }</style>
    </style>
  </head>
  <body bgcolor="gray">
    ⓑ
    <h1>Capture the flag event Thursday: with refreshments!</h1>
    <img src="capture.png" ⓒ>
    <img src="ⓓ">
    <p>Have fun and win prizes! There will be <b>all-you-can-eat</b>
    pizza and limited beverages.</p>
    <script>alert("The message is " + "ⓔ")</script>
    <p onmouseover="alert('The secret message is ' + 'ⓕ')">
      If you've read this far, there is a secret message!
    </p>
  </body>
</html>
```

(a) ____

(b) ____

(c) ____

(d) ____

(e) ____

(f) ____

1. `onload="alert('XSS 1')"`
2. `<script>alert('XSS 2')</script>`
3. `"); alert("XSS 3");("`
4. `flag.png" onclick="alert('XSS 4')`
5. `'); alert('XSS 5');('`
6. `red;}</style><script>alert('XSS 6')</script> <style type="unused">`

3. (30 points) Matching definitions and concepts. Fill in each blank with the letter of the corresponding answer. Each answer is used exactly once.

(a) ____ When set, the CPU allows all instructions to execute

(b) ____ A common kind of race condition vulnerability

(c) ____ Cryptographic primitive for integrity protection

(d) ____ Trusted code that checks all sensitive operations

(e) ____ Effect depends on which of two actions happens first

(f) ____ A parallelizable mode of operation

(g) ____ Injection into HTTP headers

(h) ____ $(\mathrm{E}(M), \mathsf{MAC}(M))$

(i) ____ Implements a pseudo-random permutation

(j) ____ Can fake interest in advertisements

(k) ____ A small random modification to a program input

(l) ____ $(\mathrm{E}(M), \mathsf{MAC}(\mathrm{E}(M)))$

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

(n) ____ E.g., `OR 1 = 1;--`

(o) ____ A fuzzing tool named after a breed of rabbits

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

(q) ____ Malicious requests made to another web site

(r) ____ Used on directories with multiple independent users

(s) ____ A block cipher design based on invertible components

(t) ____ Explains why free collisions are easier to find

A. AFL B. birthday paradox C. block cipher D. clickjacking E. CTR F. encrypt and MAC G. encrypt then MAC H. MAC I. mutation J. paravirtualization K. race condition L. reference monitor M. response splitting N. setuid O. SQL injection P. sticky bit Q. substitution-permutation network R. supervisor bit S. TOCTTOU T. XSRF

4. (27 points) Bad hash functions.

   Below are three C functions which each implement a hash function taking a sequence of unsigned short integers (16-bit values) as input, and producing a 16-bit hash result as output. No function with a 16-bit output would be suitable for use as a cryptographic hash function, since a computer could easily find pre-images and collisions by brute force. But these functions are even worse than that, because such attacks can be carried out even by hand.

   For each implementation and requested attack below, provide input(s) to the hash function that show the existence of the attack. We recommend that you write your answers as sequences of 16-bit values written as 4 hexadecimal digits. For partial credit, also briefly explain why your attacks work. All the parts have answers that do not require complex calculations.

   (a) Pre-image attack

```
unsigned short hash1(unsigned short *in, int len) {
    unsigned short total = 0;
    for (int i = 0; i < len; i++)
        total = total + in[i];
    return total;
}
```

   Given an input for which the output of `hash1` is `0x4271`.

(b) Second pre-image (targeted collision) attack

```
unsigned short hash2(unsigned short *in, int len) {
    unsigned short total = 0;
    for (int i = 0; i < len; i++) {
        total = total ^ in[i];
        total = total + i;
    }
    return total;
}
```

`hash2` applied to `0x28c3, 0x23ab, 0xe47a, 0xbd7b` gives `0x5271`. Find a different input that produces the same output. Recall that the `^` operator in C represents XOR, which is also written $\oplus$. You may want to use the following fact about the XOR operator: $A \oplus B = C$ if and only if $(A \oplus D) \oplus (B \oplus D) = C$.

(c) Free collision attack

```
unsigned short hash3(unsigned short *in, int len) {
    unsigned short total = 0;
    for (int i = 0; i < len; i++)
        total = total + first_16_bits_of_sha256(in[i]);
    return total;
}
```

Find two distinct inputs to `hash3` that produce the same output. The function named `first_16_bits_of_sha256` computes the first 16 bits of the SHA256 hash of its 16-bit input; in other words, it is a pretty good hash function on 16-bit values. But you don't have to give the colliding hash output in your answer, and you shouldn't need to compute `first_16_bits_of_sha256`.