

**Computer Science 4271**  
**Fall 2023**  
**Midterm exam 2**  
**November 16th, 2023**  
**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 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 each question carefully before answering it. 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 within the available time and space.
- 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, to an aisle: \_\_\_\_\_

Sign and date: \_\_\_\_\_

Question	Points	Score
1	38	
2	20	
3	12	
4	30	
Total:	100	

## 1. (38 points) SQL injection

Each of the following scenarios presents Java code that constructs a SQL database statement by using a string `input` in an unsafe way. Your task is to choose a malicious value for the `input` variable that will achieve an attacker's goal via SQL injection. In each case the SQL query is constructed by concatenating constant strings (Java strings use similar syntax as C) and variables using the `+` operator. Some reminders about SQL syntax are at the end.

- (a) One query is constructed as follows:

```
String query = "SELECT name, roomnum FROM Guests " +  
              "WHERE name = '" + input + "'";
```

This query retrieves information about a guest staying in a hotel. But there is another table in the same database, named `Evidence`, which contains information about your bad behavior. Choose a value for `input` that will modify the SQL to remove that table using SQL's `DROP TABLE` command.

- (b) Here the vulnerable query is the same as in the previous part, but your goal is different. Instead you'd like to get information about all the guests (i.e., all rows of the table). Choose a value for `input` that will achieve that.

- (c) This query is more complex, with three concatenated variables, but assume that the other two variables are properly sanitized:

```
acct = sanitizeAcctNum(acct);  
myname = sanitizeName(myname);  
String query = "UPDATE PayableInfo SET acctnum = " + acct +  
              " WHERE name = '" + myname +  
              "' AND ssn = " + input + "";
```

This query is used in a corporate HR system to update the information used for direct deposit of employees' salaries. In normal operation, you would be allowed to update the bank account number used for your pay, as long as you showed that you knew your Social Security number (`ssn`, stored as an integer). For your attack, your goal is to change the information for your co-worker Bob (his `name` in the database is just `Bob`) so that his paychecks are sent to your account instead. In your attack, the `acct` and `myname` variables will still hold your account number and your name, respectively; you can't change them. Instead, you must achieve your attack by changing only the `input` variable in this query.

- (d) This query uses features to limit the size of the results that are returned:

```
String query = "SELECT username from Users " +  
              "WHERE (uid = " + input + ") LIMIT 1;"  
ResultSet results = stmt.executeQuery(query);  
if (results.length() > 1) { throw new InvalidQuery(); }
```

This query converts a numeric user ID `uid` into a string `username`. Because user IDs are supposed to be unique, this query should yield at most one result row. The `LIMIT 1` clause enforces this so that even if the query matches multiple rows, only the first will be returned by the database. Furthermore, it is also checked by the Java code, which will raise an exception (and not print any data) if it gets more than one row from the database. It turns out that this database also contains Social Security numbers, 9-digit numbers stored as integers in a column named `ssn`. You are a budding identity thief, and you have the partial information that the SSN of one user of the system is of the form `47213xxxx`, where you know the first five digits but not the last four. Your goal is to use SQL injection to change this query into one that returns the username of the user who matches the information you know about their SSN. You can assume only one user has an SSN of that form, but you don't know their user ID either.

We are following the convention of using **ALLCAPS** for SQL keywords, **CamelCase** for the names of tables, and **lowercase** for the names of columns. Constant strings in SQL are delimited by single quote marks. SQL can contain comments that are enclosed between `/*` and `*/`, as in C, or that start with `--` and go until the end of the line. SQL statements should properly be terminated with semicolons, as in C, though the server used here won't complain if a semicolon is missing at the very end of a query. The comparison operators in SQL are `=`, `<>`, `<`, `>`, `<=`, and `>=`. They can all be used on numbers, and the first two can also be used on strings. The logical operators in SQL, in order of decreasing precedence, are named **NOT**, **AND**, and **OR**.

## 2. (20 points) Matching Unix files and permissions

On the left are 8 sets of permissions for Unix filesystem objects, with the permission bits shown in octal, followed by the owner (always `root` in these examples), and the group owner. On the right are 10 pathnames and descriptions of Unix filesystem objects (e.g., files and directories). Match the objects on the right with their appropriate permissions from the left by writing the corresponding uppercase letter in each blank. In our model solution, which is based on pretty normal Ubuntu 22.04 Linux machine, all of the choices on the left are used at least once, and two are used twice.

- |                                 |   |
|---------------------------------|---|
|                                 | (a) _____ <code>/dev/sda</code> , representing direct access to a hard disk   |
|                                 | (b) _____ <code>/tmp</code> , a directory for anyone's temporary files        |
| A. 0440 <code>root root</code>  | (c) _____ <code>/bin/cp</code> , a utility program                            |
| B. 0644 <code>root root</code>  | (d) _____ <code>/usr/games/robots</code> , a game with a high-score file      |
| C. 0660 <code>root disk</code>  | (e) _____ <code>/var/log/dpkg.log</code> , a log of package installations     |
| D. 0666 <code>root root</code>  | (f) _____ <code>/bin/su</code> , a setuid utility program                     |
| E. 0755 <code>root root</code>  | (g) _____ <code>/etc/sudoers</code> , a security-sensitive configuration file |
| F. 1777 <code>root root</code>  | (h) _____ <code>/etc/timezone</code> , a non-private configuration file       |
| G. 2755 <code>root games</code> | (i) _____ <code>/usr/bin</code> , a directory of system programs              |
| H. 4755 <code>root root</code>  | (j) _____ <code>/dev/null</code> , writes to which are discarded              |

3. (12 points) Multiple choice. Each question has only one correct answer: circle its letter.
- (a) Which of these vulnerability types is a kind of race condition?
    - A. Double free
    - B. Stack buffer overflow
    - C. Response splitting
    - D. TOCTTOU
    - E. Insecure file permissions
  - (b) The “MIFARE Classic” is a model of chip that was widely used in public transit smart cards since its introduction in the 1990s, but whose core cryptographic primitive was shown to be insecure in research published in 2008 and 2009. What was this primitive?
    - A. Triple DES
    - B. AES-128
    - C. A proprietary shift-register stream cipher
    - D. DES
    - E. XORing each byte with 0x3c
  - (c) Which of these vulnerability types is a kind of injection?
    - A. Insecure use of temporary files
    - B. XSS
    - C. History stealing
    - D. Uninitialized use
    - E. Use after free
  - (d) Which of these is **not** a way to virtualize a program’s interface with the operating system?
    - A. VMware
    - B. chroot(2)
    - C. Solaris Zones
    - D. FreeBSD jails
    - E. JavaScript

4. (30 points) Matching definitions and concepts. Fill in each blank with the letter of the corresponding answer. Each answer is used exactly once.

- (a) \_\_\_\_ Standard user name for UID 0
- (b) \_\_\_\_ When a results depends on which of two programs executes first
- (c) \_\_\_\_ A character encoding with synonyms for ASCII symbols
- (d) \_\_\_\_ Transforming data to prevent its being useful for an attack
- (e) \_\_\_\_ Tricking a user so their UI actions have an unintended effect
- (f) \_\_\_\_ A file referring to another file system object by path
- (g) \_\_\_\_ Multiple directory entries referring to the same inode
- (h) \_\_\_\_ XORing a plaintext with fresh random bits
- (i) \_\_\_\_ Used for escaping in C strings
- (j) \_\_\_\_ E.g., trying all possible encryption keys one by one
- (k) \_\_\_\_ A virtual machine monitor running under a normal OS
- (l) \_\_\_\_ Makes permissions on `/tmp` work differently
- (m) \_\_\_\_ A modern version of the Caesar cipher
- (n) \_\_\_\_ A command-line utility that shows extended permissions information
- (o) \_\_\_\_ A declarative formatting language used with HTML
- (p) \_\_\_\_ CPU flag set when an OS kernel executes
- (q) \_\_\_\_ An idealized model of a cryptographic primitive
- (r) \_\_\_\_ A system call to get metadata about a file
- (s) \_\_\_\_ A command-line utility that shows the results of `stat(2)`
- (t) \_\_\_\_ A command-line utility to change file permissions

A. \ (backslash)   B. brute-force   C. `chmod`   D. clickjacking   E. CSS   F. `getfacl`  
G. hard link   H. hypervisor   I. one-time pad   J. race condition   K. random oracle  
L. `root`   M. ROT-13   N. sanitization   O. `stat(1)`   P. `stat(2)`   Q. sticky bit  
R. supervisor bit   S. symbolic link   T. UTF-7