CS 88: Week 4: Class 7: Defensive Programming and SQL Injection Attacks
Swat IDs of your Group:

Discussion Question 1: The Art of Fuzzing

Testing allows us to see if running our code results in expected behavior. However, reasoning about "correct" behavior is hard and often has complex subtleties.

Part A: What is the fault in this program? Identify one test case that identifies the failure and one test case that does not.

```
void toUpperCase (char * str){
      for (int i = 0, e = strlen(str) - 1; i < e; i++){
            if (isalpha(str[i] && islower(str[i]))){
                  str[i] = str[i] - 32;
            }
      }
      printf("%s\n", str);
}
```

Part B: Testing program behavior is more than half of the development costs of most software development. Below is pseudo code for testing a critical program named run_program over the entire universe of possible inputs. Discuss why or why not this code is a good example of testing, and suggest improvements to this pseudo code (consider whether the code below is scalable, likelihood of use, etc)

```
for test in allPossibleInputs:
      run_program(test)
```

Part C: Generating carefully crafted random input, is often referred to as Fuzz testing. The goal is to find crashes that can be triggered by input that a typical user would never think of entering. What kinds of software do you think can be tested with fuzz testing?

Circle all that apply:
PDF readers, Graphics, office suites, Libraries (images, audio),, gcc, SQL databases, Browsers, Unix tools: fopen,Crypto APIs, filesystem drivers, OS kernels, code written in memory safe languages.

Part D: Discuss 2-3 limitations to fuzz testing

Discussion Question 3: SQL Injection Attacks

Part A: Assume that a database only stores the sha256 value for the password and eid columns. The following SQL statement is sent to the database, where the values of the $passwd and $eid variables are provided by users. Does this program have a SQL injection problem? If so, please describe how to exploit this vulnerability.

```
mysql> SELECT * FROM employee  WHERE eid=SHA2('$eid', 256)
                                    and password=SHA2('$passwd', 256);
```

Part B: Assume now that the hash value is not calculated inside the SQL statement; instead it is calculated in the PHP code using PHP's hash() function. Does this modified program have an SQL injection vulnerability?

```
mysql> $hashed_eid = hash('sha256', $eid);
mysql> $hashed_passwd = hash('sha256', $passwd);
mysql> $sql = "SELECT * FROM employee WHERE eid='$hashed_eid' AND
               password='$hashed_passwd'";
```

Part C: To defeat SQL injection attacks, a web application has implemented a filtering scheme at the client side: basically, on the page where users type their data, a filter is implemented using JavaScript. It removes any special character found in the data, such as apostrophe, characters for comments. Assume that the filtering logic does its job; is this solution able to defeat SQL injection attacks?

Part D: Comment on whether you think the following PHP code is secure.

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN FROM employee WHERE eid= '$eid' and
password=?";
       if ($stmt = $conn->prepare($sql)) {
            $stmt->bind_param("s", $pwd);
            $stmt->execute(); ...
       }
```