SQL Injection Attack

12.1. 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.

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

12.2. This problem is similar to Problem 12.1., except that the hash value is not calculated inside the SQL statement; it is calculated in the PHP code using PHP's `hash()` function. Does this modified program have a SQL injection problem?

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

12.3. What if the SQL statement is constructed in the following way (with a line break in the WHERE clause), can you still launch an effective SQL injection attack?

```
    SELECT * FROM employee
    WHERE  eid= '$eid' AND
       password='$password'
```

12.4. The following SQL statement is sent to the database to add a new user to the database, where the content of the `$name` and `$passwd` variables are provided by the user, but the `EID` and `Salary` field are set by the system. How can a malicious employee set his/her salary to a value higher than `80000`?

```
$sql = "INSERT INTO empoyee (Name, EID, Password, Salary)
        VALUES ('$name', 'EID6000', '$passwd', 80000)";
```

12.5. The following SQL statement is sent to the database to modify a user's name and password, where the content of the `$eid`, `$name`, `$oldpwd` and `$newpwd` variables are provided by the user. You want to set your boss Bob's salary to $1 (using the `Salary` field), while setting his password to something that you know, so you can later log into his account.

```
$hashed_newpwd = hash('sha256', $newpwd);
$hashed_oldpwd = hash('sha256', $oldpwd);
$sql = "UPDATE employee
        SET name='$name', password='$hashed_newpwd'
        WHERE eid = '$eid' and password='$hashed_oldpwd'
```

12.6. The following SQL statement is sent to the database, where `$eid` and `$passwd` contain data provided by the user. An attacker wants to try to get the database to run an arbitrary SQL statement. What should the attacker put inside `$eid` or `$passwd` to achieve that goal. Assume that the database does allow multiple statements to be executed.

```
$sql = "SELECT * FROM employee
        WHERE eid='$eid' and password='$passwd'"
```

12.7. MySQL does allow us to put two SQL statements together, separated by a semicolon. Can we use a SQL injection vulnerability to get the victim server to run an arbitrary SQL statement?

12.8. 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, and keywords reserved for SQL statements. Assume that the filtering logic does it job, and can remove all the code from the data; is this solution able to defeat SQL injection attacks?

12.9. Is the following PHP code 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();

   ...
}
```

12.10. Please modify the following program using the prepared statement.

```
$sql = "UPDATE employee SET password='$newpwd'
        WHERE eid ='$eid' and password='$oldpwd'";
```

12.11. SQL Injection allows remote users to execute code on databases. In a typical setup, the database is only accessible to the web application server, not to remote users, so there is no direct path for users to interact with the database. How can users inject code to the database?

12.12. To defeat code injection attacks when a C program needs to invoke an external program, we should not use `system()`; instead, we should use `execve()`. Please describe the similarity between this countermeasure and the prepared statement, which is a countermeasure against SQL injection attacks.