SQL Injection Attack

W4.1. 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? If so, please describe how to exploit this vulnerability.

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

Yes, this program does have a SQL injection problem. Looking at the code above, we can see that the eid and password variables are directly used in the WHERE clause. So it would work if we assume that eid = 500 is valid and we do 500' # without using the password. So the attacker would be able to access the account without the password.

W4.2. This problem is similar to Problem W4.1., 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'";
```

No, this modified program does not have the SQL injection problem. In this program, we are not using the eid directly in the SELECT statement.

W4.3. 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'
```

Yes, we can still launch the SQL injection attack. We can effortlessly launch the attack by making the user equal to 'OR 1=1/*' and the password to be */#. The attacker would be able to enter the user account.

W4.4. 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?

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

To increase the employee's salary and set it to a higher value, we can use the SQL injection in the password. So in this SQL statement, if we assume that the user password is "seed," we can set the user password to seed',90000)#, which would put the employee's salary to 90000.

W4.5. The following SQL statement is sent to the database to modify a user's name and pass- word, 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'
```

In this code, we can see that the new pwd and the old pwd are hashed before the SQL UPDATE statement, so we cannot use this to make the attack. We can use the name variable to launch the attack and set bob salary to 1.

So we are going to set the name to Bob',password='hashseed',Salary=1 Where name='bob'

Our SQL statement would look like this:
$sql = "UPDATE employee SET name=' Bob',password='hashseed',Salary=2 Where name='bob'""

W4.6. 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'"
```

Within $eid, we can put "OR 1=1/*". Or 1=1 is an attempt to make a query succeed because the statement 1=1 is always true, while /* is to start a quarry where the rest of the query is ignored. Within $passwd, we can put */#, and this is because we are already commenting out the rest of the query within $eid by using /*, and */# is just a different way of commenting.

W4.7. 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?

Yes, suppose a SQL client application was designed poorly enough to accept inputs from the user without any validation and filtering and have it passed into the SQL server. In that case, an arbitrary SQL statement could be run.

W4.8. 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?

No, filtering alone is not a solution to defeat SQL injection attacks. There are many ways to bypass filtering. one such method can arise from quotations that are not always necessary if you inject them into numerical data fields or column names.

W4.9. 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();

   ...
}
```

No, the above PHP code is not secure. That is because the eid is being read from the user's point of view, making it susceptible to attacks.

W4.10. Please modify the following program using the prepared statement.

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

answer:

```
$stmt = $conn->prepare("UPDATE employee SET
       password=? WHERE eid =? and password=?");
$stmt->bind_param("ss", $eid, $pwd);

$stmt->execute();
```

W4.11. 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 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?

SQL injection is used to access the database indirectly, even if the user does not have direct access to the database. This is one of the hacking methods that is used to extract data or insert data into the database using the input field on the web page.

W4.12. 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 counter- measure against SQL injection attacks.

The similarity between the countermeasure and the prepared statement is that they both have different trusted codes from the user input.