# 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.

Yes.  
  
Assuming that 1000 is a valid EID, the user can enter  
  
1000, 256)'#  
  
as the EID, which would then make the SQL statement  
  
SELECT \* FROM employee  
WHERE eid='SHA2(1000, 256)'#,256)' and ....  
  
allowing the attacker to login as EID 1000 without a password.

# 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’";

No. Since $eid & $passwd aren't used directly in the query, there is no risk of attack here.

# 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. Most SQL databases support multi-line comments enclosed in //. By setting the $eid to   
  
' OR 1=1/\*  
  
and the $password to \*/#  
  
we can make the SQL query look like this  
  
SELECT \* FROM employee  
WHERE eid='' OR 1=1/\*' AND  
password='\*/#'  
  
Filtering out the comments gives us  
  
SELECT \* FROM employee  
WHERE eid='' OR 1=1  
  
which will allow an attacker to login successfully.

# 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)";

Let's assume the user wants username as john and password as iheartburgers. By setting $name to john and $passwd to   
  
iheartburgers', 2000000)#  
  
a malicious employee can set his/her salary to 2M

# 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’

Though oldpwd & newpwd are hashed, name is untouched, and we can use that to exploit the query. Setting name to  
  
Bob', password='precalculated\_hash', Salary=1 WHERE name='Bob'#  
  
will cause the SQL statement to become  
  
UPDATE employee SET name='Bob', password='precalculated\_hash', Salary=1 WHERE name='Bob'   
  
and the rest will be ignored as they're commented out.

# 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’"

In SQL, ; is the statement delimiter. Therefore by setting the password to some random string but suffixing it with  
  
'; DELETE FROM employee WHERE eid != 1000  
  
will cause the SELECT statement to run, return no results, and then followed by the DELETE statement.

# 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?

We can with the command: alice’; Select 1 # but

The database will block multiple SQL statement. Actually, this is countermeasure that is written in PHP code. The PHP use API to invoke database. The API support to one single SQL statement.

The query() function of the mysqli extension in php does not allow multiple statements to be run in the database. This is a protective measure against sql injection

If you want to run multiple sql statements, you can use mysqli's multi\_query()

# 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?

It is possible to carefully construct the injection and bypass any filtering, therefore filtering alone is not a recommended solution.

# 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. While the password parameter is part of a prepared SQL statement, the EID itself is directly read from user input, and is susceptible to the attack,

eid reads directly from user input and is vulnerable to attacks

# 10: Please modify the following program using the prepared statement. $sql = "UPDATE employee SET password=’$newpwd’ WHERE eid =’$eid’ and password=’$oldpwd’";

$sql = “UPDATE employee SET password=? WHERE eid =? and password=?”;

$sql = "UPDATE employee SET password=? WHERE eid =? and password=?";

# 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?

SQL injection attacks only modify the query string passed from the client (browser) to the web server. The actual query runs on the web server, so it will be executed even if access is restricted.

# 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

Code is separated from user input.