Database Security – SQL Injection Questions

1.Consider the following PHP script for a login page:

```
$username = $_GET[user];
$password = $_GET[pwd];
$sql = "SELECT * FROM usertable WHERE username = '$username'
AND password = '$password' ";
$result = $db->query($sql);
if ($result->num_rows > 0) { /* Success */ }
else { /* Failure */ }
```

(a) Explain why a URL where user is set to " ' or 1 = 1 -- " will result in a successful login.

(b) Suppose we change lines 1 and 2 to

$username = addslashes($ GET[user])

$password = addslashes($ GET[pwd])

The addslashes function adds a slash before every quote. That is addslashes("a'b") will output the string "a\'b". Explain why this prevents the attack from part (a).

(c) Does addslashes completely solve the problem? Consider the GBK Chinese unicode character set. Some characters in GBK are single bytes while others are double bytes. In particular, the following table shows a few GBK characters:

0x 5c = \

0x 27 = '

0x bf 27 = ¿'

0x bf 5c = 縗

That is, the database interprets 0x bf27 as two characters, but interprets 0x bf5c as a single chinese character. Show that using addslashes as in part (b) leads to a SQL injection attack. What value of user will result in a successful login?

(d) Because addslashes did not eliminate the problem of SQL injection, we decide to employ some form of input filtering. A recommends the use of JavaScript to validate the input before passing it over to the server. Is this enough to ensure that SQL injection will not happen? Why?

(e) As an extension to input filtering, we decide to strip the -- characters in the input so that attackers cannot inject a comment and thereby execute an injection. Can an attacker bypass this restriction? How?

(f) Stored Procedures have a reputation for being able to defend against SQl injection attacks. With this in mind, we decide to employ the following stored procedure:

```
CREATE PROCEDURE VerifyUser
        @username varchar(50),
        @password varchar(50)
AS
BEGIN
        SELECT * FROM UserTable
        WHERE UserName = @username
        AND Password = @password;
END
GO
```

The stored procedure is called from the PHP file using the following code:

```
$sql = "CALL VerifyUser ("+ $_GET[user] + "," + $_GET[pwd]+")";
```

Is the web application susceptible to an SQL injection? Why or why not?

2. An application allows users to self-register by specifying their username and password. Every user is assigned a privilege level of 5 by default, which is the lowest privilege level. The query that inserts the data of newly registered users into the database is:
INSERT INTO users (username, password, privilege) VALUES (‘Paul’, ‘Secret’,5);
Suggest an input that can elevate a new user’s privilege level to 0, which is the highest.
Your answer should clearly mention the input supplied to the username and password fields, as well as how the supplied input modifies the internal query.

3. Consider an application which allows users to update their passwords. For changing the password, the users have to supply their current password as a precaution. Internally, the query that resets the password is:
UPDATE users SET password=’newsecret’
WHERE user=’marcus’ AND password=’secret’;

(a) Suggest an input that can change the password of the admin, without actually knowing the current password.
(b) Assume that an attacker supplies the following input:
Username: admin’ or 1=1--
Password:123456
What effect does the input have on the database?

4. A company has deployed a hand-made defense mechanism against SQL injection which removes any SQL keywords it recognizes in the input string. For example, an input designed to launch a piggybacked query in the search field:
Books’; DROP TABLE users;
Gets reduced to:
Books’; TABLE users;
Which makes no sense and cannot launch an attack at the database site.
Suggest a modification of the input which can accomplish the same objective, which is to drop the user table.

5. Consider the following code for a stored procedure:

```
CREATE PROCEDURE VerifyUser
        @username varchar(50),
        @password varchar(50)
AS
BEGIN
        SELECT * FROM UserTable
        WHERE UserName = @username
        AND Password = @password;
END
GO
```

The stored procedure is called from the PHP file using the following code:
$sql = “CALL VerifyUser (“+ $_POST[“username”] + ”,” + $_POST[password]+”)”;
Is the web application susceptible to an SQL injection? Why or why not?

6. Consider the following PHP code which generates a query to login into a web application:
$sql = " SELECT id, name, eid, salary, birth, ssn, phonenumber, address, email, nickname, Password FROM credential WHERE eid= ’ “ + $input_eid + “ ’ and password=’ “ + md5($input_pwd)+’ ";
Will an input of the form:
EID : admin
Password : 123456’ or 1=1 --
Enable an attacker to login to the application? Why or why not?

7. You have found a SQL injection vulnerability but have been unable to carry out any useful attacks, because the application rejects any input containing whitespace. How can you work around this restriction?

8. You have found a SQL injection vulnerability in a login function, and you try to use the input ‘ or 1=1-- to bypass the login. Your attack fails, and the resulting error message indicates that the -- characters are being stripped by the application’s input filters. How could you circumvent this problem?

9. Consider the following code for a stored procedure which executes a search query for a web application. In case no search query is provided, the query results the details of all the products in the database. If a query is provided, it returns all products which have the search term in the product name.

```
CREATE PROCEDURE SP_ProductSearch @prodname varchar(400) = NULL AS
DECLARE @sql nvarchar(4000)
SELECT @sql = ' SELECT ProductID, ProductName, Category, Price ' +
' FROM Product Where '
IF @prodname IS NOT NULL
SELECT @sql = @sql + ' ProductName LIKE '" + @prodname + ""
EXEC (@sql)
```

Is the stored procedure susceptible to an SQL injection? Why or why not?

10.Consider the following PHP script for a login page:

```
$username = addslashes($ GET[user])
$password = addslashes($ GET[pwd])
$sql = "INSERT INTO usertable VALUES(’$username’ , ’$password’) ";
$result = $db->query($sql);
```

Once the user has logged in, the same web application also allows the user to view his profile. For doing this, the web application extracts the username of the currently logged in member from the session table, and then uses that username to extract the details.

```
$sql = “SELECT username FROM sessiontable WHERE session
=’” +sessionid + ““““;
$username= $db->query($sql);
$sql = “SELECT * FROM users WHERE username=’”+username+”””;
$result = $db->query($sql);
```

The application developer is confident that the web application is secure from SQL injection because all the input fields are strings, and the use of the single-quote (‘) has been rendered ineffective by the addquotes() function. Is the developer’s assumption valid? Why?

11. You are creating security test cases to check for SQL injection on an input field that allows up to 5 alphanumeric characters. You are planning to apply equivalence partitioning to reduce the number of test cases you will need to execute. Given this information, which of the following is the minimum set of inputs you would need to use to test this field?

a. bbbbb, 12345, ‘
b. %, ‘, @, ab123
c. ‘, ab123
d. ‘

Answer: C is correct as this has one test for SQL injection and one for a valid input. This is the minimum number of tests. A and B have more than the minimum number and D doesn’t have enough tests because it doesn’t test the valid input. It would be advisable to do more testing on the various characters that can support SQL injection, but this question is asking to apply EP and get the minimum number of test cases.

12. *You are engaged in a penetration-test where you are attempting to gain access to a protected location. You are presented with this login screen:*

What are some examples of you how you would attempt to gain access?

13. Given this HTML and php code:

HTML:

```
<form action="sql.php" method="POST"/>
  <p>Username: <input type="text" name="login" /><br />
     Password: <input type="text" value="password" /></p>
</form>
```

PHP:

```
<?php $query = "SELECT * FROM users WHERE username = '{$_POST['login']} AND
     password= '{$_POST['password']}";  $result = mysql_query($query); echo "$_POST['login'] attempted"}>
```

Assume that logins are allowed if $result above is simply checked as being non-empty.

1. Given this PHP code, give a username and password that will allow you to login without knowing the password.
2. Give a username and password that allows you to login without knowing a valid username.
3. Give a username and password that allows a cross-site scripting attack.

XML Security - Questions

1. Assume that multiple groups simultaneously query the same XML document. Each user group has a different access-control policy. How to specify access policies at various levels of granularity? How to efficiently enforce those access policies?
2. Compare the XACML, XACL models of XML security?
3. Give example for access rule.
4. Due to the hierarchical nature of XML, how to apply the access rule?
5. Discuss different approaches to perform conflict resolution?