SQL INJECTION

Secure Software Development Course
Computer Eng. Depart. - Sharif Univ.

Ahmad Boorghany
boorghany@ce.sharif.edu

Spring 1392
**SQL INJECTION**

- **Impact:** **SEVERE**
  - Steal/change/delete data
  - System access

- **GET**
  - `http://.../?name=Bobby';DROP TABLE users --`

- **POST**
  - `http://login.example.com`
  - `name=' OR ''=''&password=' OR ''=''`

- **What happens**
  - `SELECT * FROM users WHERE name='myname' AND password='mypass'`
  - `SELECT * FROM users WHERE name=' OR ''='' AND password='' OR ''=''`
THE BAD

Python:

```python
cconn = MySQLdb.connect(host = "localhost", user = "uname", passwd = "pass")
cursor = conn.cursor()
query = "SELECT * FROM user_info WHERE email = "" + email + ""
cursor.execute(query)
```

PHP:

```php
$link = mysql_connect('localhost', 'uname', 'pass');
mysql_query("SELECT * FROM user_info WHERE email = "" . $email . """);
```
THE GOOD

Python:

```python
conn = MySQLdb.connect(host = "localhost", user = "uname", passwd = "pass"
cursor = conn.cursor()
cursor.execute("SELECT * FROM user_info WHERE email = %s", email)
```

PHP:

```php
$link = mysql_connect('localhost', 'uname', 'pass');
mysql_query("SELECT * FROM user_info WHERE email = '' .
    mysql_real_escape_string($email) . """");
```
SQLI SOLUTIONS

- Validate and escape inputs
  - `mysql_real_escape_string`
- Consider all inputs:
  - GET variables
  - POST variables
  - HTTP Request Headers
    - Cookie
    - Referrer
    - User Agent
- Do NOT validate by hand!
  - `str_replace("'", "\'", $_GET["name"]);`
SQLi SAMPLES

- SQLi in User Agent:

```php
$link = mysql_connect('localhost', 'uname', 'pass');
mysql_query("INSERT INTO stat (uagent) VALUES (" .
            $_SERVER['HTTP_USER_AGENT'] . ")");
```

- SQLi in Cookie:

```php
$link = mysql_connect('localhost', 'uname', 'pass');
mysql_query("SELECT * FROM user_info WHERE user = " .
            $_COOKIE['user'] . ")");
```
Parameterized statements

Python

```python
conn = MySQLdb.connect(host = "localhost", user = "uname", passwd = "pass")
cursor = conn.cursor()
cursor.execute("SELECT * FROM user_info WHERE email = %s", email)
```

Java

```java
java.sql.PreparedStatement prep = connection.prepareStatement(
    "SELECT * FROM `users` WHERE USERNAME = ? AND PASSWORD = ?");
prep.setString(1, username);
p prep.setString(2, password);
p prep.executeQuery();
```
Be careful of LIKE statements.
Whenever LIKE is used, in addition to regular escaping:
- % and _ must be escaped from user input.
Validating functions DO NOT do this for you.
- Like mysql_real_escape_string in PHP.
- Or like cursor.execute escaping in Python.
You must do it by hand.
Parameterized Stored Procedures

NOT all stored procedures are safe.

Insecure sample:

```sql
CREATE PROCEDURE VerifyUser @username varchar(50), @password varchar(50) AS
BEGIN
    DECLARE @sql nvarchar(500);
    SET @sql = 'SELECT * FROM UserTable
                WHERE UserName = '''' + @username + ''''
                AND Password = '''' + @password + ''''
                ;
    EXEC(@sql);
END
GO
```
Secure one:

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

Do NOT use `mysql_real_escape_string` for integers. Cast them instead. Bad usage:

```php
$link = mysql_connect('localhost', 'uname', 'pass');
mysql_query("SELECT name FROM user_info WHERE id = ".
    mysql_real_escape_string($id));
```

Performance Issue: Do not write ‘12’ instead of 12. Otherwise the index is not used properly.
SQLI CONCLUSION

- Use parameterized queries everywhere:
  - NO excepted query!
  - NO excepted parameter!
REFERENCES