SQL Injection (SQLi)

SQL Injection (SQLi) is **a web security vulnerability that allows attackers to manipulate an application's SQL queries by injecting malicious input**. This can result in unauthorized access, data leaks, and even complete compromise of the database.

1. How SQL Injection Works

a. Vulnerable Query

When user input is directly incorporated into an SQL query without proper validation or sanitization, it creates a vulnerability.

- Example of Vulnerable Code

```
SELECT * FROM users WHERE username = 'user' AND password = 'pass';
```

- If input is

```
username: ' OR 1=1 --
password: anything
```

- The resulting query becomes `SELECT * FROM users WHERE username = '' OR 1=1 -- ' AND password = 'anything';`
- The condition OR 1=1 always evaluates as true, bypassing authentication.

2. Exploitation and Impact

a. Common Exploits

1. **Authentication Bypass**

- Using inputs like

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

- Allows attackers to bypass login pages.

2. **Data Extraction**

- Exploiting vulnerable forms to retrieve sensitive data

```
UNION SELECT username, password FROM users;
```

3. **Database Enumeration**

- Attackers identify database structure (tables, columns) using:

```
UNION SELECT table_name FROM information_schema.tables;
```

4. **Remote Code Execution**

- Advanced SQLi can execute OS-level commands (e.g., with MySQL's xp_cmdshell).

b. Impact

- **Data Breach**
  - Sensitive data (e.g., usernames, passwords) is exposed.
- **Database Corruption**
  - Attackers can delete or modify data.
- **Privilege Escalation**
  - Exploiting SQLi in administrative interfaces.
- **System Compromise**
  - Leveraging SQLi to execute shell commands and take control of servers.

3. Person-in-the-Browser (Malware)

Attackers can use person-in-the-browser (PITB) malware to facilitate SQLi by injecting malicious scripts into web sessions.

How PITB Works

1. Delivery

- Malware (e.g., through Flash or Java applets) infects the victim's browser.
- It intercepts browser traffic and manipulates web forms or requests.

2. SQLi Injection

- The malware modifies user inputs or hidden fields in web forms to include SQLi payloads.

3. Exploitation

- Injected payloads are sent to the server, exploiting SQLi vulnerabilities.

Examples

- A compromised Flash-based ad runs malicious JavaScript in the browser.
- The applet intercepts form submissions, adding SQLi payloads.

4. Validation and Sanitization of Web Forms

Proper validation and sanitization of user input is the primary defense against SQLi.

a. Best Practices

1. **Parameterized Queries/Prepared Statements**

- Use placeholders for user input to prevent injection.
- Example (in Python with SQLite)

```
cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?",
(username, password))
```

2. **Input Validation**

- Allow only expected input formats (e.g., reject special characters in usernames).
- Example
    - Regex for usernames

```
^[a-zA-Z0-9_]{3,20}$
```

3. **Output Encoding**

- Encode dynamic data before displaying it to prevent script execution in cases where injection leads to stored scripts.

4. **Limit Permissions**

- Restrict database user privileges to minimize the impact of SQLi (e.g., use read-only accounts where possible).

5. **Web Application Firewalls (WAFs)**

- Deploy WAFs to detect and block SQLi attempts in HTTP traffic.

6. **Sanitize User Input**

- Remove or escape dangerous characters like:

```
'
"
--
;
```

- Avoid relying solely on escaping as it can be bypassed.

b. Examples of Poor Validation

- Unvalidated Input
    - Accepting any characters in a form field, allowing SQL injection payloads.

- Improper Escaping
    - Simply escaping single quotes without using parameterized queries:

```
SELECT * FROM users WHERE username = 'O\'Reilly'; -- Vulnerable to bypass
techniques.
```

5. SQL Injection Types

a. Classic SQLi

- Directly injecting malicious SQL code into input fields.
- Example

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

b. Blind SQLi

- The server doesn't display errors but responds differently based on injected conditions.
- Example

```
' AND 1=1 --
' AND 1=0 --
```

c. Time-Based Blind SQLi

- Uses time delays to infer information.
- Example (MySQL)

```
' OR IF(1=1, SLEEP(5), 0) --
```

d. Error-Based SQLi

- Relies on error messages to extract data.
- Example

```
' UNION SELECT NULL, table_name FROM information_schema.tables --
```

6. Preventing SQL Injection

a. Use an ORM

- Object-Relational Mappers (ORMs) like SQLAlchemy or Hibernate abstract SQL queries, reducing the risk of manual query injection.

b. Avoid Dynamic SQL

- Avoid concatenating user input directly into queries

```
SELECT * FROM users WHERE username = '" + user_input + "';
```

c. Employ Secure Defaults

- Disable dangerous features like SQL command execution in the database.

d. Regular Security Audits

- Use automated tools to scan for SQLi vulnerabilities
  - **SQLMap**: Automates SQL injection detection and exploitation.
  - **Burp Suite**: Identifies SQLi vulnerabilities in web applications.

7. Summary

Aspect	Details
What is SQLi?	Exploiting insecure SQL queries to inject malicious commands.
Common Exploits	Authentication bypass, data extraction, remote code execution.
Role of PITB	Malware in the browser injects SQLi payloads into form submissions.
Primary Defense	Use parameterized queries and input validation.
Advanced Mitigations	WAFs, ORMs, and database user privilege restrictions.

SQL Injection remains one of the most critical and prevalent web vulnerabilities. **Proper implementation of validation, sanitization, and parameterized queries can effectively mitigate this threat**. Additionally, awareness of emerging attack vectors like person-in-the-browser (PITB) malware underscores the need for comprehensive security measures at both the client and server levels.