In-Depth Notes: Common Places to Find SQL Injection (SQLi) Vulnerabilities

SQL Injection (SQLi) occurs when an attacker injects malicious SQL queries into input fields, allowing them to manipulate the database. Below is an in-depth explanation of the most common places to find SQLi vulnerabilities, along with examples, testing methodologies, and tools for finding these vulnerabilities.

Made by: shadowkage69

---

1. GET Parameters

- **Description**: GET parameters are passed in the URL after a `?` symbol. They are commonly used for filtering, sorting, or retrieving specific data.
- **Example**: `https://example.com/page?id=1`.
- **How SQLi Occurs**: If the `id` parameter is directly used in a SQL query without proper sanitization, an attacker can inject malicious SQL.
- **Testing**:
  - Append SQLi payloads to the parameter.
  - Example: `https://example.com/page?id=1' OR 1=1 --`.

    - If the application is vulnerable, it may return all rows from the database.
- **Tools for Finding**:
    - **ParamSpider**: Extracts parameters from URLs.
        - Example: `python3 paramspider.py -d example.com`.
    - **Arjun**: Fast parameter discovery tool.
        - Example: `arjun -u https://example.com/page?id=1`.
    - **Burp Suite**: Use the **Param Miner** extension to guess parameters.
- **Tools for Testing**:
    - **Manual Testing**: Use browser or `curl`.
    - **Automated Testing**: Use `sqlmap` (`sqlmap -u https://example.com/page?id=1`).

---

2. POST Parameters

- **Description**: POST parameters are sent in the body of an HTTP request, typically in forms (e.g., login, registration).
- **Example**: A login form with `username` and `password` fields.
- **How SQLi Occurs**: If the input fields are not sanitized, an attacker can inject SQL queries.
- **Testing**:
    - Use tools like Burp Suite to intercept the request.

    - Inject payloads like `' OR '1'='1` into the `username` or `password` fields.
    - Example:

```
POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
username=admin' OR '1'='1&password=password
```

- **Tools for Finding**:
    - **Burp Suite**: Intercept and analyze POST requests.
    - **Arjun**: Can also discover POST parameters.
        - Example: `arjun -u https://example.com/login --data="username=admin&password=password"`.
- **Tools for Testing**:
    - **Burp Suite**: Intercept and modify POST requests.
    - **sqlmap**: Test POST requests (`sqlmap -u https://example.com/login --data="username=admin&password=password"`).

---

3. Headers

- **Description**: HTTP headers like `Cookie`, `User-Agent`, and `Referer` can be exploited if they are used in SQL queries.
- **Example**: `Cookie: session_id=12345`.

- **How SQLi Occurs**: If the `session_id` is used in a SQL query without sanitization, an attacker can inject malicious SQL.
- **Testing**:
  - Use Burp Suite to modify headers.
  - Inject payloads like `' OR 1=1 --` into headers.
  - Example:

```
GET /dashboard HTTP/1.1
Host: example.com
Cookie: session_id=12345' OR 1=1 --
```

- **Tools for Finding**:
  - **Burp Suite**: Use the **Param Miner** extension to guess headers.
  - **Arjun**: Can also test headers for parameters.
- **Tools for Testing**:
  - **Burp Suite**: Modify headers and test for vulnerabilities.
  - **sqlmap**: Test headers (`sqlmap -u https://example.com/dashboard --headers="Cookie: session_id=12345"`).

---

4. Cookies

- **Description**: Cookies are small pieces of data stored in the browser and sent with every request to the server.

- **Example**: `Cookie: user_id=12345`.
- **How SQLi Occurs**: If the `user_id` is used in a SQL query, an attacker can manipulate it.
- **Testing**:
  - Modify cookies using browser dev tools or Burp Suite.
  - Inject payloads like `' OR '1'='1`.
  - Example:

```
GET /profile HTTP/1.1
Host: example.com
Cookie: user_id=12345' OR '1'='1
```

- **Tools for Finding**:
  - **Burp Suite**: Intercept and analyze cookies.
  - **Cookie Editor**: Browser extension to modify cookies.
- **Tools for Testing**:
  - **Burp Suite**: Intercept and modify cookies.
  - **sqlmap**: Test cookies (`sqlmap -u https://example.com/profile --cookie="user_id=12345"`).

---

5. JSON/API Endpoints

- **Description**: APIs often accept JSON payloads, which can be exploited if not properly sanitized.

- **Example**: `{"username":"admin","password":"password"}`.
- **How SQLi Occurs**: If the JSON input is used directly in SQL queries, an attacker can inject malicious SQL.
- **Testing**:
  - Use tools like Burp Suite or Postman to send JSON payloads.
  - Inject payloads like `' OR 1=1 --` into JSON fields.
  - Example:

```
POST /api/login HTTP/1.1
Content-Type: application/json
{"username":"admin' OR 1=1 --
","password":"password"}
```

- **Tools for Finding**:
  - **Burp Suite**: Intercept and analyze JSON payloads.
  - **Postman**: Manually test API endpoints.
- **Tools for Testing**:
  - **Burp Suite**: Intercept and modify JSON payloads.
  - **sqlmap**: Test JSON endpoints (`sqlmap -u https://example.com/api/login --data='{"username":"admin","password":"password"}'`).

---

6. File Uploads

- **Description**: File upload features may use filenames or metadata in SQL queries.
- **Example**: Uploading a file named `test'.jpg`.
- **How SQLi Occurs**: If the filename is used in a SQL query, an attacker can inject malicious SQL.
- **Testing**:
  - Upload files with SQLi payloads in the filename or metadata.
  - Example: `test' OR 1=1 --.jpg`.
- **Tools for Finding**:
  - **Burp Suite**: Intercept file upload requests.
  - **OWASP ZAP**: Analyze file upload functionality.
- **Tools for Testing**:
  - **Manual Testing**: Use browser or `curl`.
  - **Burp Suite**: Intercept and modify file upload requests.

---

7. Search Fields

- **Description**: Search bars or filters often process user input and use it in SQL queries.
- **Example**: Searching for `test' OR 1=1 --`.
- **How SQLi Occurs**: If the search term is not sanitized, an attacker can inject SQL.
- **Testing**:
  - Enter SQLi payloads into the search bar.

    - Example: `test' OR 1=1 --`.
- **Tools for Finding**:
    - **Burp Suite**: Intercept and analyze search requests.
    - **Arjun**: Can discover parameters in search fields.
- **Tools for Testing**:
    - **Manual Testing**: Use browser.
    - **sqlmap**: Test search fields (`sqlmap -u https://example.com/search?q=test`).

---

8. Hidden Form Fields

- **Description**: Hidden fields in HTML forms can be manipulated by attackers.
- **Example**: `<input type="hidden" name="user_id" value="12345">`.
- **How SQLi Occurs**: If the hidden field value is used in a SQL query, an attacker can modify it.
- **Testing**:
    - Use browser dev tools or Burp Suite to modify hidden fields.
    - Inject payloads like `' OR 1=1 --`.
- **Tools for Finding**:
    - **Burp Suite**: Intercept and analyze hidden fields.
    - **Browser Dev Tools**: Inspect and modify hidden fields.
- **Tools for Testing**:

- **Burp Suite**: Intercept and modify hidden fields.

---

9. URL Paths

- **Description**: Dynamic parts of the URL path can be exploited.
- **Example**: `https://example.com/users/12345`.
- **How SQLi Occurs**: If the path parameter is used in a SQL query, an attacker can inject SQL.
- **Testing**:
  - Modify the URL path to include SQLi payloads.
  - Example: `https://example.com/users/12345' OR 1=1 --`.
- **Tools for Finding**:
  - **Burp Suite**: Intercept and analyze URL paths.
  - **xnLinkFinder**: Extract dynamic paths from a website.
- **Tools for Testing**:
  - **Manual Testing**: Use browser or `curl`.
  - **sqlmap**: Test URL paths (`sqlmap -u https://example.com/users/12345`).

---

10. Second-Order SQLi

- **Description**: Inputs stored in the database and executed later (e.g., user registration fields).
- **Example**: Registering with a username like `admin' --`.
- **How SQLi Occurs**: The payload is executed when the stored data is used in a SQL query.
- **Testing**:
  - Submit SQLi payloads during registration or input forms.
  - Observe behavior when the data is used later.
- **Tools for Finding**:
  - **Burp Suite**: Monitor and analyze stored inputs.
  - **Manual Testing**: Use browser and monitor application behavior.
- **Tools for Testing**:
  - **Manual Testing**: Use browser and monitor application behavior.

Summary of Tools:

- **Manual Testing**: Browser, `curl`, browser dev tools.
- **Automated Testing**: `sqlmap`, Burp Suite, Arjun, ParamSpider, xnLinkFinder.

By understanding these common places and using the right tools, you can effectively identify and exploit SQLi vulnerabilities. Always ensure you have proper authorization before testing.