SQLi-1

Author: Eyad Islam El-Taher

Date: January 21, 2026

How the Researcher Exploited the Vulnerability

This section explains step by step how the researcher exploited the SQL injection vulnerability in the Formidable Pro WordPress plugin.

1. Unauthenticated AJAX Endpoint

The website exposed the following WordPress AJAX endpoint:

`/wp-admin/admin-ajax.php`

Normally, sensitive AJAX actions require authentication. However, the vulnerable action did not:

`action=frm_forms_preview`

This endpoint could be accessed by anyone, even without logging in.

2. User-Controlled HTML and Shortcodes

The endpoint accepted parameters such as:

- `after_html`
- `before_html`

These parameters were rendered directly into the page output and allowed the use of WordPress shortcodes. Example:

`after_html=Hello World`

Critically, the endpoint also allowed plugin-defined shortcodes.

3. Vulnerable Shortcode: `[display-frm-data]`

The Formidable Pro plugin provides the shortcode:

`[display-frm-data]`

This shortcode is used to display form submission data and accepts parameters such as:

- `id` – Form ID
- `order_by` – SQL column(s)
- `order` – Sorting direction (`ASC` or `DESC`)

Vulnerability: These parameters were inserted directly into an SQL query without sufficient sanitization.

4. SQL Injection via the `order` Parameter

The researcher injected arbitrary SQL through the `order` parameter.

Example request:

```
action=frm_forms_preview&
after_html=XXX[display-frm-data id=835 order_by=id limit=1 order=zzz]YYY
```

This caused SQL errors on the backend, confirming the presence of an SQL injection vulnerability.

5. Blind SQL Injection Using ORDER BY

The injection was a **blind SQL injection**:

- Query results were not directly visible
- The attacker could control row ordering
- This allowed extraction of data one bit at a time

The injection occurred inside the `ORDER BY` clause of the SQL query.

6. Bypassing Query Manipulation

Formidable Pro internally modified SQL queries, especially handling commas (`,`), which normally breaks injected queries.

To bypass this, the researcher:

- Used `sqlmap`
- Applied a custom `--eval` expression
- Used the `commalesslimit` tamper script

This neutralized the plugin's query rewriting logic and allowed valid injected SQL queries to execute.

7. Database Extraction

Using the refined `sqlmap` configuration, the researcher successfully:

- Identified the DBMS (MySQL)
- Enumerated database tables
- Extracted sensitive information, including:
  - Administrator usernames
  - Password hashes
  - Personal data from form submissions
  - Webroot path

All data was retrieved via blind SQL injection.

8. Privilege Escalation via iThemes Sync (Optional)

The researcher observed that another plugin, `iThemes Sync`, stored:

- User ID
- Authentication key

These values were stored in plaintext in the WordPress database.

Why This Matters

The iThemes Sync plugin authenticates requests using the following logic:

$$\text{hash} = \text{SHA256}(\text{user\_id} \,||\, \text{action} \,||\, \text{args} \,||\, \text{key} \,||\, \text{salt})$$

If an attacker obtains the `user_id` and `key`, they can:

- Generate valid authentication hashes
- Send authenticated administrative requests
- Perform actions such as:
  - Adding new administrator users
  - Installing or activating plugins
  - Achieving remote code execution

9. Attack Chain Summary

```
Unauthenticated AJAX endpoint
        ↓
Shortcode injection
        ↓
SQL Injection (ORDER BY)
        ↓
Blind SQL data exfiltration
        ↓
Read WordPress database
        ↓
(Optional) Admin access via iThemes Sync
        ↓
Potential Remote Code Execution
```