ETX Signal-X

Daily Intelligence Digest

Sunday, March 1, 2026

4	4
ARTICLES	TECHNIQUES

Article 1

Bypassing Login via NoSQL Operator Injection: A MongoDB Authentication Hack

Source: securitycipher

Date: 18-May-2025

URL: https://infosecwriteups.com/bypassing-login-via-nosql-operator-injection-a-mongodb-authentication-hack-b895211f60e0

T1 MongoDB NoSQL Operator Injection for Authentication Bypass

💉 Payload

```
POST /login HTTP/1.1
Host: targetsite.com
Content-Type: application/json
Content-Length: 52

{
  "username": { "$ne": null },
  "password": { "$ne": null }
}
```

⚔️ Attack Chain

1. Identify the login endpoint that accepts JSON payloads (e.g., /login).
2. Craft a payload where the username and password fields are replaced with MongoDB query operators (e.g., { "$ne": null }).
3. Send the payload to the login endpoint.
4. If the backend is vulnerable, authentication is bypassed, granting access as the first matched user.

🔍 Discovery

Observed that the login endpoint accepted JSON input and suspected that user input was being used directly in MongoDB queries without sanitization. Tested with NoSQL operators to confirm injection.

🔓 Bypass

The use of the $ne (not equal) operator in both username and password fields causes the MongoDB query to match any document where both fields are not null, effectively bypassing authentication checks.

🔗 Chain With

Can be chained with privilege escalation if the first user in the database is an admin or has elevated permissions. Potential for session fixation or further NoSQL injection attacks if other endpoints are similarly vulnerable.

T2 Boolean-Based NoSQL Injection with $or Operator

💉 Payload

```
POST /login HTTP/1.1
Host: targetsite.com
Content-Type: application/json
Content-Length: 77

{
  "username": "victim",
  "password": { "$ne": "wrongpassword" },
  "$or": [ { "isAdmin": true }, { "isAdmin": { "$exists": false } } ]
}
```

⚔️ Attack Chain

1. Target the login endpoint with a payload that includes the $or operator.
2. Set the username to a known user and the password to a $ne operator to bypass password checks.
3. Inject an $or clause to manipulate additional query logic, such as forcing isAdmin to true or bypassing admin checks.
4. If successful, gain access as the targeted user, potentially with escalated privileges.

🔍 Discovery

After confirming NoSQL injection was possible, experimented with additional MongoDB operators to manipulate query logic and privilege checks.

🔓 Bypass

The $or operator allows the attacker to alter the query logic, bypassing both authentication and privilege checks by satisfying at least one condition in the array.

🔗 Chain With

Can be used to escalate privileges during login or to target specific users (e.g., admin accounts). May also be leveraged to enumerate user roles or attributes if error messages are verbose.

Article 2

How I find vulnerability can make X(Twitter) lose millions of dollars

Source: securitycipher

Date: 30-Jul-2025

URL: https://l4zyhacker.medium.com/how-i-find-vulnerability-can-make-x-twitter-lose-millions-of-dollars-ae34d713254f

⚠️ No actionable techniques found

Article 3

Log Me Maybe: When Log Files Leaked Secrets I Wasn’t Meant to See

Source: securitycipher

Date: 20-Jun-2025

URL: https://infosecwriteups.com/log-me-maybe-when-log-files-leaked-secrets-i-wasnt-meant-to-see-%EF%B8%8F-a3db4d2624b7

T1 Direct Access to Exposed Log Files for Secret Extraction

💉 Payload

```
GET /debug/server-error.log HTTP/1.1
Host: api.victim.com
```

⚔️ Attack Chain

1. Use subdomain enumeration tools (e.g., subfinder) to collect all subdomains of the target (victim.com).
2. Use tools like httpx and gau to enumerate all accessible endpoints and files, filtering for interesting file extensions (e.g., .log).
3. Identify accessible log files, such as `https://api.victim.com/debug/server-error.log`.
4. Directly request the log file via browser, curl, or Burp Suite.
5. Review the log file contents for sensitive information such as credentials, secrets, tokens, or internal error messages.

🔍 Discovery

Mass recon using automated tools (subfinder, httpx, gau) with a focus on unusual or sensitive file extensions (.log) in publicly accessible locations. The presence of a `.log` file in the output triggered manual review.

🔓 Bypass

N/A (relies on misconfiguration and lack of access controls on log files).

🔗 Chain With

Use credentials, tokens, or secrets found in logs to escalate privileges, pivot to internal systems, or perform authenticated attacks (e.g., API abuse, lateral movement). Combine with endpoint fuzzing to discover additional exposed files or directories.

Article 4

When APIs Don’t Check Roles: Broken Authorization in Customer Deletion Endpoint

Source: securitycipher

Date: 01-Jun-2025

URL: https://callgh0st.medium.com/when-apis-dont-check-roles-broken-authorization-in-customer-deletion-endpoint-8b318fd4c8f6

T1 Role Bypass via JWT in Customer Deletion API

💉 Payload

```
DELETE /3.0/contact/2 HTTP/2
Host: api.freepalestine.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:124.0) Gecko/20100101 Firefox/124.
0 Bugcrowd Callgh0st
Accept: application/json
Accept-Language: en-GB
Accept-Encoding: gzip, deflate, br
Authorization: Bearer TOKEN_HERE
Api-Version: Redacted
Content-Type: application/json
Origin: https://office.freepalestine.com
Referer: https://office.freepalestine.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Te: trailers
```

⚔️ Attack Chain

1. Log in as a low-privilege (non-admin) user to the SaaS platform.
2. Navigate to a section in the UI that triggers API calls (e.g., viewing a specific resource).
3. Click the "Email Redacted" button or similar UI element to trigger an API request.
4. Intercept the subsequent GET request to `https://api.freepalestine.com/2.0/redacted/{redacted_id}` and extract the `Authorization: Bearer` JWT token from the request headers.
5. Craft a DELETE request to `https://api.freepalestine.com/3.0/contact/{contact_id}` using the extracted JWT token, replacing `{contact_id}` with any valid/target contact ID (sequential IDs are valid).
6. Send the crafted DELETE request as a low-privilege user and observe successful deletion of the contact, regardless of role.

🔍 Discovery

While testing API endpoints as a low-privilege user, the researcher noticed that certain UI features were restricted but corresponding API endpoints could still be accessed. By reviewing Burp Suite logs, a JWT token was discovered in an API call triggered by a UI action. This token was then reused to test privileged API actions, leading to the discovery that role checks were not enforced server-side for the DELETE operation.

🔓 Bypass

The API failed to enforce server-side role checks on the DELETE endpoint. Even though the UI hid the functionality from non-admin users, possession of a valid JWT token (obtainable via legitimate user actions) allowed privilege escalation. The attack bypasses UI-based access controls and leverages direct API interaction.

🔗 Chain With

If other sensitive endpoints (e.g., user management, order deletion) are similarly protected only by UI and not by backend role checks, the same JWT token can be used to escalate privileges across multiple API actions. Sequential or predictable contact IDs allow for bulk or automated deletion attacks. If the JWT token is long-lived or not scoped per action, it can be reused for further lateral movement or privilege escalation.