Vulnerability Analysis: CVE-2026-72898 Metabase - Unauthenticated SQL Injection

Author: Synthex **Site:** denizhalil.com CVSS 10 HIGH **Date:** August 2026

INTRODUCTION

In modern enterprise architectures, business intelligence (BI) platforms such as Metabase serve as central data gateways, aggregating access to production databases, data warehouses, and identity management systems. Consequently, vulnerabilities within BI platforms pose severe systemic risks to an organization's entire digital infrastructure. In early August 2026, a critical security flaw identified as **CVE-2026-72898** (GitHub Advisory ID: `GHSA-vwf4-m7j8-wcjf`) was publicly disclosed and documented as an active zero-day threat being exploited in the wild, triggering an immediate CISA Known Exploited Vulnerabilities (KEV) catalog listing. Assigned a maximum CVSS score of **10.0 (Critical)**, this vulnerability allows unauthenticated remote attackers to execute arbitrary SQL commands directly against the core Metabase application database without needing prior credentials or session tokens. By doing so, an attacker can hijack administrative privileges, modify application settings, and extract stored connection credentials for connected enterprise databases, leading to complete infrastructure compromise.

LEARNING OBJECTIVES

- **Understanding SQL Injection Mechanics:** Analyze how unauthenticated inputs processed at public endpoints lead to SQL Injection in Clojure-based applications.
- **Examining the Attack Lifecycle:** Track the multi-stage exploitation chain—from public payload delivery to core application DB mutation and administrative takeover.
- **Evaluating Software Impact:** Identify vulnerable versus patched release streams for both Open Source and Enterprise editions of Metabase.
- **Detecting Indicators of Compromise (IoCs):** Recognize HTTP request signatures and log anomalies that indicate active exploitation attempts.
- **Executing Remediation Steps:** Implement immediate security patches, network-level mitigations, and post-exploitation incident response actions.

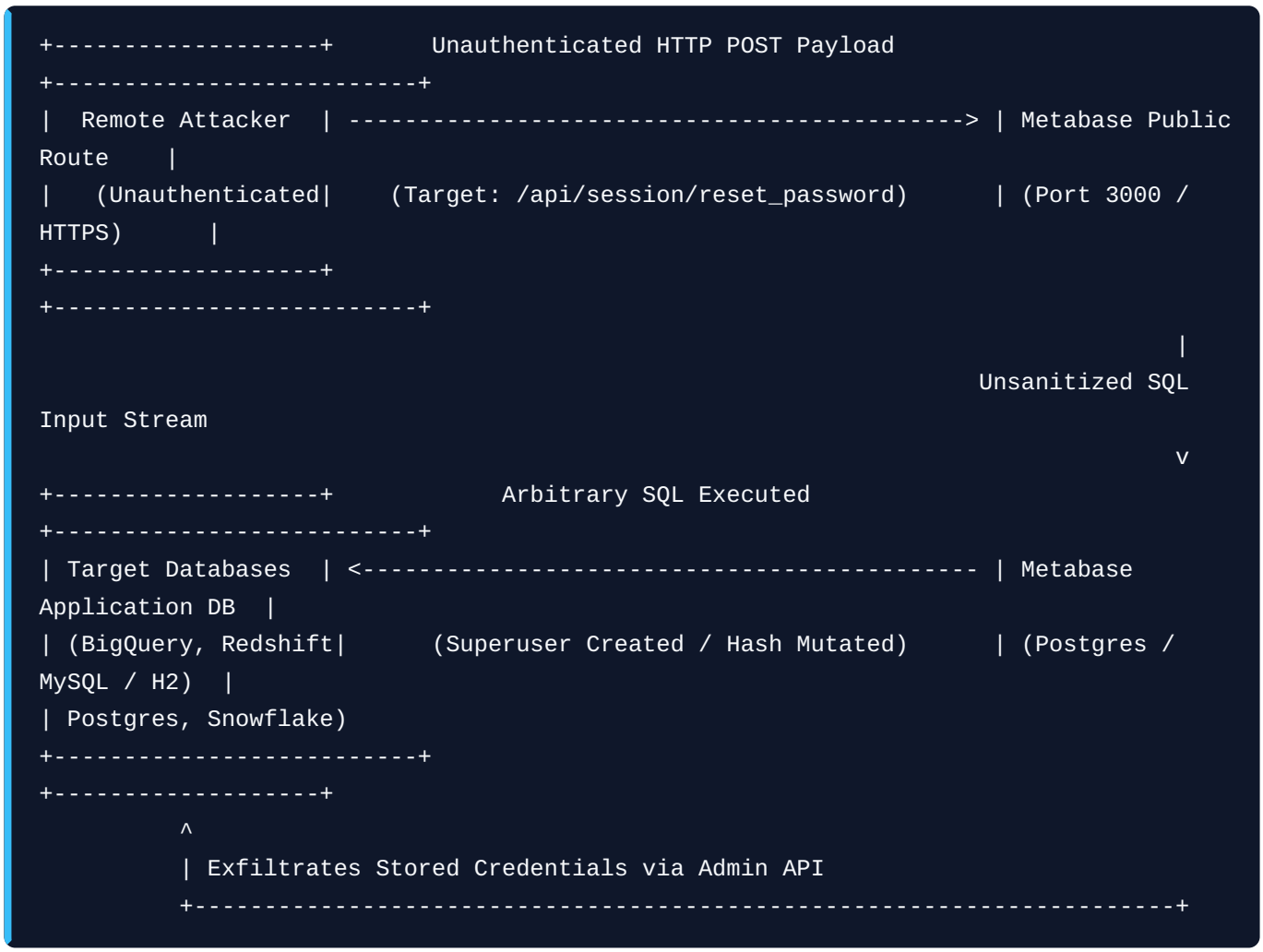
WHAT IS METABASE - UNAUTHENTICATED SQL INJECTION CVE-2026-72898

Metabase is a widely deployed, open-source business intelligence and data visualization platform engineered to connect directly with enterprise data sources—ranging from traditional relational databases like PostgreSQL and MySQL to cloud data warehouses such as Snowflake, Amazon Redshift, and Google BigQuery. By abstracting complex SQL queries into intuitive visual dashboards, Metabase functions as a centralized gateway for data analytics across entire organizations. However, because it centralizes access to high-value infrastructure and stores saved connection secrets, any security flaw in

its application logic presents an immediate threat to the wider data ecosystem. Tracked under **CVE-2026-72898** (and GitHub Advisory ID `GHSA-vwf4-m7j8-wcjf`), this vulnerability is a critical, unauthenticated SQL Injection (SQLi) flaw rated **10.0 (Critical)** on the CVSS scale. The bug resides within the public API endpoint responsible for handling user password reset requests—specifically `POST /api/session/reset_password`. Because the endpoint must remain publicly accessible to assist locked-out users, it operates completely outside the boundary of session authentication checks, exposing the vulnerable backend code path directly to the public internet.

When an attacker submits a specially crafted HTTP request to this unauthenticated route, malicious SQL payloads passed inside request parameters are improperly sanitized and evaluated directly by the underlying Metabase application database engine. This flaw allows external, unauthenticated threat actors to execute arbitrary database queries, alter internal user records, and effectively bypass authentication controls without ever possessing a valid user account, password, or session token.

- **Unauthenticated Initial Access:** The target API endpoint (`POST /api/session/reset_password`) requires no active session, API key, or valid user credentials to be reached and exploited.
- **Direct Application Database Injection:** Malicious input is concatenated directly into SQL queries, giving the attacker query execution privileges on the backend storage database (PostgreSQL, MySQL, or H2).
- **Administrative Privilege Takeover:** Threat actors can manipulate internal records within the `core_user` table to set their account's `is_superuser` flag to TRUE or overwrite an existing administrator's password hash.
- **Secondary Infrastructure Exfiltration:** Upon securing administrative rights, attackers gain access to stored connection strings and credentials for connected target databases, leading to full enterprise data exfiltration.



TECHNICAL DETAIL: HOW THE VULNERABILITY WORKS

The underlying root cause of **CVE-2026-72898** resides in how incoming API parameters are handled within the backend routing layer during password reset handling. When a client initiates a password reset via `POST /api/session/reset_password`, the endpoint accepts a JSON payload containing parameters such as `token` or `email`. In vulnerable versions of the application, these user-controlled input strings were passed directly into dynamic database query construction functions without undergoing rigorous type enforcement, input sanitization, or prepared statement parameter binding. Because Metabase utilizes standard database interfaces (such as HoneySQL and JDBC abstractions in Clojure), query components are typically expected to be safely parameterized. However, flawed logic within the reset handler interpolated the untrusted input directly into raw string fragments prior to handing the query off to the driver execution layer. Consequently, when an attacker sends standard SQL control characters—such as single quotes (`'`), semicolons (`;`), or comment operators (`--`)—the database parser interprets those characters as active SQL commands rather than literal string parameters.

Once the injected SQL syntax breaks out of its intended data context, the backend application database engine (PostgreSQL, MySQL, or H2) executes the attacker's arbitrary SQL commands under the security context of the Metabase application user. Threat actors leverage this execution control to query internal metadata, dump application tables, or directly modify user records inside the `core_user` system table.

By overwriting password hashes or elevating privileges on an existing account, the attacker bypasses all upper-layer authentication mechanisms and achieves complete administrative control.

- **Input Interpolation Failure:** User-supplied strings inside the `POST /api/session/reset_password` JSON body are dynamically concatenated into raw SQL query strings instead of utilizing parameterized placeholders.
- **Database Driver Execution:** The backend DBMS parses the untrusted parameter input as valid SQL, allowing blind, error-based, or stacked query execution.
- **Application DB Mutation:** Attackers execute `UPDATE` queries against the internal Metabase database (e.g., setting `is_superuser = TRUE` or updating `password_hash` in `core_user`).
- **Credential Exfiltration & Lateral Movement:** Armed with full administrator privileges, the attacker uses Metabase's admin REST APIs to decrypt stored connection strings and credentials for connected production databases.

Flawed vs. Secure Code Pattern (Conceptual SQL / Query Logic)

The vulnerable implementation fails by directly formatting untrusted user input into raw SQL queries:

```
-- ❌ VULNERABLE: Direct string concatenation creates SQL Injection
-- User input passed to token parameter:  ' OR '1'='1' --
SELECT * FROM core_user
WHERE reset_token = '' OR '1'='1' --' AND reset_token_created_at > NOW() - INTERVAL
'1 hour';
```

In contrast, the patched logic enforces strict parameter binding, treating all user input purely as literal data:

```
-- ✅ SECURE: Parameterized query (Prepared Statement) prevents code execution
SELECT * FROM core_user
WHERE reset_token = $1 AND reset_token_created_at > NOW() - INTERVAL '1 hour';
-- Parameter $1 is safely bound as literal string: "' OR '1'='1' --"
```

Detection Pattern (Log Signature): Inspect web server and WAF access logs for a `POST /api/session/reset_password` returning an **HTTP 400 or 500** status code containing SQL injection signatures, followed immediately by a `GET /api/user/current` or `POST /api/session` returning an **HTTP 200** status code from the same source IP address.

AFFECTED SOFTWARE & PLUGINS

The vulnerability impacts a wide range of self-hosted deployments across both **Metabase Community (Open Source)** and **Metabase Enterprise / Pro** editions. Because the flawed password reset logic resides in the core API handling layer shared between open-source and commercial distributions, all

active release branches prior to the August 2026 security patches are susceptible to unauthenticated exploitation. Organizations operating self-hosted Docker containers, JAR deployments, or Kubernetes clusters must review their current version against the official fixed releases below and upgrade immediately to prevent unauthorized administrative compromise.

Both **Metabase Open Source Edition** (`0.x` releases) and **Metabase Enterprise Edition** (`1.x` releases) are affected across all maintenance branches prior to the security patches.

Release Branch	Affected / Vulnerable Versions	Patched Version (Safe)
v0.58 / v1.58	`0.58.0` – `0.58.23` / `1.58.0` – `1.58.23`	**`0.58.24` / `1.58.24`**
v0.59 / v1.59	`0.59.0` – `0.59.20` / `1.59.0` – `1.59.20`	**`0.59.21` / `1.59.21`**
v0.60 / v1.60	`0.60.0` – `0.60.16` / `1.60.0` – `1.60.16`	**`0.60.17` / `1.60.17`**
v0.61 / v1.61	`0.61.0` – `0.61.10` / `1.61.0` – `1.61.10`	**`0.61.11` / `1.61.11`**
v0.62 / v1.62	`0.62.0` – `0.62.8` / `1.62.0` – `1.62.8`	**`0.62.9` / `1.62.9`**
v0.63 / v1.63	`0.63.0` – `0.63.4` / `1.63.0` – `1.63.4`	**`0.63.5` / `1.63.5`**

CONCLUSION

CVE-2026-72898 serves as a critical case study on how input sanitization failures in unauthenticated API routes can destabilize an enterprise's entire security posture. In modern cloud environments, business intelligence platforms like Metabase occupy a uniquely privileged position because they centralize data warehouse tokens, database credentials, and service accounts. When an unauthenticated SQL injection vulnerability is exposed on a public route like `/api/session/reset_password`, remote threat actors gain an immediate avenue to bypass authentication checks and convert public network access into core application control.

The risk profile associated with this vulnerability extends well beyond the internal Metabase user database. By manipulating application state to secure superuser access, attackers inherit the full spectrum of privileges granted to the BI instance. This allows malicious actors to extract stored connection secrets, inspect sensitive query histories, and pivot directly to underlying backend databases such as Snowflake, BigQuery, PostgreSQL, or Amazon Redshift. In effect, a single unauthenticated SQL injection vulnerability on the web tier can trigger a cascade of data exfiltration across connected data warehouses.

To defend against active exploitation, administrators of self-hosted Metabase instances must prioritize updating their deployments to the safe patch versions corresponding to their release stream, such as `v0.63.5` or `v1.63.5`. In environments where emergency maintenance windows or strict change management protocols delay immediate patching, network administrators should deploy temporary

mitigations at the reverse proxy or Web Application Firewall layer. Specifically, blocking or tightly restricting external HTTP requests directed toward `POST /api/session/reset_password` effectively closes the exposure window while permanent patches are tested and rolled out.

Following the application of security patches, security operations teams must execute a thorough post-incident audit and remediation protocol. This process should include immediately terminating all active user sessions, resetting administrator passwords, and rotating all backend database credentials stored within Metabase configuration files. Furthermore, security analysts should inspect web server access logs for suspicious requests to password reset endpoints and analyze connected data warehouse logs for unusual query activity that may indicate secondary data exfiltration prior to patching.