New issue

cross_browser SQL Injection Vulnerability in Legacy MySQL Fingerprint Details Endpoint #24

Open

wing3e opened 2 weeks ago Owner

cross_browser SQL Injection Vulnerability in Legacy MySQL Fingerprint Details Endpoint

1) CNA / Submission Type

- Submission type: Report a vulnerability (CVE ID request)
- Reporter role: Independent security researcher
- Report date: March 20, 2026

2) Reporter Contact (fill before submit)

- Reporter name: `Winegee`
- Reporter email: `Winegee@zju.edu.cn`
- Permission to share contact with vendor: `Yes`

3) Vendor / Product Identification

- Vendor: Song-Li
- Product: cross_browser
- Repository: https://github.com/Song-Li/cross_browser
- Reviewed local repository path: `cross_browser`
- Affected component(s):
- `flask/uniquemachine_app.py`

- HTTP route: `/details`

4) Vulnerability Type

- CWE: CWE-89 (Improper Neutralization of Special Elements used in an SQL Command)
- Short title: SQL injection in legacy fingerprint details lookup

5) Affected Versions

- Confirmed affected: repository HEAD commit `ca690f0fe6954fd9bcda36d071b68ed8682a786a` dated January 17, 2022, when deployments use `flask/uniquemachine_app.py`
- Earliest confirmed vulnerable code presence: commit `d9f32224d4f4ceae1a7e5a0a5630846814e0f686` dated January 13, 2017
- Suspected affected range: all repository revisions that include the legacy MySQL backend file `flask/uniquemachine_app.py` with the same `/details` query construction
- Fixed version: Not available at time of report (March 20, 2026)
- Release-note caveat: the repository has no Git tags or packaged release versions in the reviewed repository tree, so affected versions should be mapped by the maintainer to published builds

6) Vulnerability Description

The legacy MySQL-backed Flask application in cross_browser contains an SQL injection vulnerability in the `/details` endpoint implemented in `flask/uniquemachine_app.py`. The handler reads `ID` from a JSON request body and directly concatenates it into a SQL `SELECT` statement without parameterization or escaping. An attacker who can reach this endpoint can submit crafted input to alter the SQL query and retrieve unintended database records, and in some MySQL deployment configurations may be able to perform broader data exfiltration or blind SQL injection techniques.

7) Technical Root Cause

1. The `/details` handler accepts attacker-controlled JSON input:
   - `flask/uniquemachine_app.py:32`
   - `ID = request.get_json()["ID"]`
2. The input is concatenated into a SQL statement:
   - `flask/uniquemachine_app.py:35`
   - `sql_str = "SELECT * FROM features WHERE browser_fingerprint = '" + ID +"'"`
3. The resulting statement is executed directly:
   - `flask/uniquemachine_app.py:36`
   - `cursor.execute(sql_str)`
4. The project's client-side details page shows a realistic attacker-controlled path into the endpoint:
   - `client/fingerprint/js/details.js:287` reads `ID` from the page URL
   - `client/fingerprint/js/details.js:295` sends `{"ID": ID}` to `/details`

5. The vulnerable backend is a legacy MySQL implementation that remains in the repository even though the current `flask/server.py` default entrypoint loads `uniquemachine_experimental_app.py`. The vulnerable file can still be deployed directly under Flask or Apache/WGI-style setups referenced by the project documentation.

8) Attack Prerequisites

- The deployment must expose the legacy backend in `flask/uniquemachine_app.py`, or otherwise route requests to its `/details` handler.
- The attacker must be able to send an HTTP `POST` request with JSON content to `/details`.
- The application must have access to the configured MySQL database `uniquemachine`.
- No authentication or authorization checks are present in the vulnerable handler.

9) Proof of Concept / Reproduction Guidance

This proof of concept assumes a deployment where the vulnerable `flask/uniquemachine_app.py` backend is exposed at `http://TARGET/details`.

1. Reproduction request

```
curl -i http://TARGET/details \
  -H 'Content-Type: application/json' \
  --data "{\"ID\":\"' OR 1=1 #\"}"
```

2. Expected vulnerable behavior

- The server constructs and executes a query equivalent to:

```
SELECT * FROM features WHERE browser_fingerprint = '' OR 1=1 #'
```

- Instead of requiring an exact `browser_fingerprint` match, the application returns the first row selected from the `features` table.
- In deployments with sufficient data and permissive database behavior, an attacker can adapt the payload for boolean-based, error-based, union-based, or time-based SQL injection testing.

10) Security Impact

- Confidentiality: High. Attackers can retrieve unintended rows from the fingerprint database and may be able to enumerate or exfiltrate additional data depending on MySQL permissions and injection technique.
- Integrity: Low to Medium. The reviewed code path performs a `SELECT`, but impact can increase if the database connector or server configuration permits stacked or write-capable statements.

- Availability: Low to Medium. Time-based or expensive injected queries can delay responses and degrade service.
- Scope: Unchanged.

11) CVSS v3.1 Suggestion

- Suggested vector for deployments exposing the legacy `/details` endpoint to untrusted callers: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L`
- Suggested base score: 8.6 (High)
- If the vulnerable backend is only reachable on an internal or operator-restricted interface, adjust `PR` and deployment assumptions accordingly.

12) Workarounds / Mitigations

- Do not deploy `flask/uniquemachine_app.py`; use the non-SQL experimental backend or a patched parameterized implementation instead.
- Restrict access to `/details` to trusted users and trusted network segments only.
- Place the service behind an application firewall or reverse proxy rule set that blocks obvious SQL injection payloads. This is only a partial mitigation.
- Use a database account with the minimum required privileges.

13) Recommended Fix

- Replace string concatenation with a parameterized query, for example using placeholders supported by the MySQL driver.
- Validate `ID` against the expected fingerprint format and reject unexpected characters and unreasonable lengths before querying.
- Remove the unnecessary `db.commit()` in the read-only code path.
- Add regression tests for payloads such as `' OR 1=1 #`, quote-breaking input, and time-based SQL injection probes.
- If the legacy backend is no longer supported, remove it from the shipped repository or mark it clearly as unsafe and deprecated until patched.

14) References

- Repository: https://github.com/Song-Li/cross_browser
- Project README: `cross_browser/README.md`
- Reviewed file: `cross_browser/flask/uniquemachine_app.py`
- Client request path reference: `cross_browser/client/fingerprint/js/details.js`
- CWE-89: https://cwe.mitre.org/data/definitions/89.html

15) Credits

- Discoverer: `Winegee`
- Discovery method: Static analysis plus repository code audit

16) Additional Notes for Form Mapping

- This submission concerns the legacy MySQL backend file `flask/uniquemachine_app.py` and its `/details` route specifically.
- In the reviewed repository state, `flask/server.py` imports `uniquemachine_experimental_app.py` by default, so the issue should be described as affecting deployments that use the legacy MySQL backend component still present in the repository.
- No public fix, security advisory, or tagged fixed release was found in the reviewed local repository snapshot.

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Code with agent mode

No branches or pull requests

Participants