SQL injection attack, querying the database type and version on MySQL and Microsoft(sqli)

Executive Summary

A **SQL Injection** vulnerability was discovered in the product category filter of the application. This flaw allows an attacker to manipulate SQL queries and retrieve sensitive information from the database. In this case, we successfully extracted the database version using a UNION-based SQL injection. This issue is classified as **High** severity due to its potential for full data exposure or database compromise.

Introduction

The test focused on identifying injection flaws in the product filter feature. By tampering with the category parameter, it was possible to inject SQL commands and retrieve information directly from the backend database.

Methodology

1. **Intercepted the request** using Burp Suite where the category parameter was set.

2. **Tested for SQL injection** by injecting UNION queries to identify the number of returned columns.

3. **Confirmed two columns**, both supporting text output.

4. **Used a UNION SELECT query** to retrieve the database version from the `@@version` variable.

Vulnerability Findings

- **Type**: SQL Injection

- **Parameter**: `category`

- **Severity:** High

- **Location**: Product listing filter

Description

The application failed to properly sanitize user input in the `category` parameter. By injecting SQL payloads, an attacker could alter the query and extract sensitive backend data.

Proof of Concept (PoC)

1. Intercept the following request:

   sql

   CopyEdit

   `GET /filter?category=Gifts HTTP/1.1`

2. Modify the category parameter to test for columns:

   bash

   CopyEdit

   `' UNION SELECT 'abc','def'#`

3. Then retrieve the database version:

   makefile

   CopyEdit

   `' UNION SELECT @@version, NULL#`

4. Observe the database version string displayed in the response (e.g., `MySQL 5.7.32` or similar).

Impact Assessment

A successful SQL injection allowed attackers to:

- View database metadata

- Extract sensitive data

- Potentially modify or delete record

- Escalate to full database compromise

Recommendations

1. Use parameterized queries or prepared statements to prevent SQL injection.

2. Validate and sanitize user input on the server side.

3. Implement a Web Application Firewall (WAF) to detect and block common injection patterns.

4. Apply least privilege principles to the database account used by the application.

Conclusion

The application is vulnerable to SQL injection through the category filter. This allowed the retrieval of the backend database version using a UNION-based payload. Proper input validation and secure coding practices are necessary to mitigate this critical issue.