Advisory:

Issue:	**SQL injection**
Severity:	High
Confidence:	Firm
Host:	http://www.digitalocean.com
Path:	/cdn-cgi/challenge-platform/h/g/orchestrate/captcha/v1

Issue detail

The URL path folder 1 appears to be vulnerable to SQL injection attacks. The payload ' was submitted in the URL path folder 1, and a database error message was returned. You should review the contents of the error message, and the application's handling of other input, to confirm whether a vulnerability is present.

The database appears to be PostgreSQL.

Remediation detail

The application should handle errors gracefully and prevent SQL error messages from being returned in responses.

Issue background

SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query.

A wide range of damaging attacks can often be delivered via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges within the database and taking control of the database server.

Remediation background

The most effective way to prevent SQL injection attacks is to use parameterized queries (also known as prepared statements) for all database access. This method uses two steps to incorporate potentially tainted data into SQL queries: first, the application specifies the structure of the query, leaving placeholders for each item of user input; second, the application specifies the contents of each placeholder. Because the structure of the query has already been defined in the first step, it is not possible for malformed data in the second step to interfere with the query structure. You should review the documentation for your database and application platform to determine the appropriate APIs which you can use to perform parameterized queries. It is strongly recommended that you parameterize *every* variable data item that is incorporated into database queries, even if it is not obviously tainted, to prevent oversights occurring and avoid vulnerabilities being introduced by changes

elsewhere within the code base of the application.

You should be aware that some commonly employed and recommended mitigations for SQL injection vulnerabilities are not always effective:

- One common defense is to double up any single quotation marks appearing within user input before incorporating that input into a SQL query. This defense is designed to prevent malformed data from terminating the string into which it is inserted. However, if the data being incorporated into queries is numeric, then the defense may fail, because numeric data may not be encapsulated within quotes, in which case only a space is required to break out of the data context and interfere with the query. Further, in second-order SQL injection attacks, data that has been safely escaped when initially inserted into the database is subsequently read from the database and then passed back to it again. Quotation marks that have been doubled up initially will return to their original form when the data is reused, allowing the defense to be bypassed.
- Another often cited defense is to use stored procedures for database access. While stored procedures can provide security benefits, they are not guaranteed to prevent SQL injection attacks. The same kinds of vulnerabilities that arise within standard dynamic SQL queries can arise if any SQL is dynamically constructed within stored procedures. Further, even if the procedure is sound, SQL injection can arise if the procedure is invoked in an unsafe manner using user-controllable data.

References

- Web Security Academy: SQL injection
- Using Burp to Test for Injection Flaws
- Web Security Academy: SQL Injection Cheat Sheet

Vulnerability classifications

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- CWE-94: Improper Control of Generation of Code ('Code Injection')
- CWE-116: Improper Encoding or Escaping of Output
- CAPEC-66: SQL Injection

Request:

```
GET /cdn-cgi/challenge-platform/h/g/orchestrate/captcha/v1?ray=7d6185dd6ed721ef HTTP/2
Host: www.digitalocean.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.62 Safari/537.36
Connection: close
Cache-Control: max-age=0
Referer: http://www.digitalocean.com/
Cookie: __cf_bm=
e4CUbOKMsIZQHDcMvIxq26XxE3tlqt0QuucSJH4UptE-1686566921-0-AaCNvV11e7kaBmiS8yarXT2vqHhMEUlzBjXEs4nR1dOARPLgINNSroZGesP5aPkEh1JF1LRbpdD57q038rhYXKX5gEExQtpwHE4eY
hHeOMrS
```

Response:

```
<a class="LazyLink___StyledA-sc-yi29t7-0 dSqHbP DefaultItemStyles__StyledLink-sc-p6sww0-2 foQqdK" href="
/products/managed-databases-postgresql" target="">
  PostgreSQL
</a>
```