PostgreSQL Security Analysis

with `geol` and `trivy` tools
github.com/adriens/geol-showcase

Gemini CLI | Adrien SALES (**X** `@rastadidi`)

February 13, 2026

Abstract

This article presents a concise analysis of the security and lifecycle of the PostgreSQL database versions.

Using the `geol` tool to check End-of-Life dates and `trivy` to scan vulnerabilities in official Docker images, I establish a risk profile for currently supported and unsupported versions.

The goal is to demonstrate the crucial importance of using maintained versions and the value of combining generative AI with optimally designed CLI tools to automate and enrich this type of analysis.

1 Executive Summary

Key Findings:

- **5 Supported Versions:** PostgreSQL versions 14-18 are actively maintained with EOL dates ranging from 2026 to 2030.
- **Fresh EOL Alert:** PostgreSQL 13 recently reached end-of-life on November 13, 2025, joining versions 9.6-12 as unsupported.
- **Security Gap:** Unsupported versions contain **2-10Œ more vulnerabilities** than supported versions, with critical CVEs present only in EOL versions.
- **Patch Impact:** Minor updates are crucial - PostgreSQL 18.1 reduced vulnerabilities by 22% compared to 18.0 (204 159 total vulnerabilities).
- **Critical Recommendation:** Migrate immediately from any version $\leq$13 to version $\geq$14. The security risk of unsupported versions is unacceptable for production environments.

Risk Profile Summary:

- ✓ **Low Risk:** Versions 14-18 (0-1 critical, 6-17 high vulnerabilities)
- × **High Risk:** Versions 9.6-13 (7-10 critical, 70-97 high vulnerabilities)

2 Introduction to the Tools

Maintaining a secure software infrastructure relies on two fundamental pillars:

- **Actively supported versions**
- **Awareness of vulnerabilities** present in the components we deploy

Below is a quick overview of the tools used for this analysis.

2.1 `geol`: The Lifecycle Guardian

`geol` (version 2.7.1) is a tool that queries the endoflife.date API to instantly retrieve software End-of-Life dates.

2.2 `trivy`: The Vulnerability Scanner

`trivy` (version 0.69.1) is an open-source scanner that detects vulnerabilities (CVEs) in container images, file systems, and Git repositories. The vulnerability database is version 2.

2.3 `gemini-cli`: AI Assistant

`gemini-cli` (version 0.28.2) is an open-source AI agent that brings Gemini's power directly to the terminal.

2.4 LaTeX: Report Generator

LaTeX is a document composition system that produces high-quality technical and scientific reports. We use `xelatex` for compilation. It is particularly suited for structuring, formatting, and presenting security analysis results clearly and professionally.

3 Methodology

The analysis for this report was conducted on February 13, 2026. The data was gathered using the following open-source tools and commands:

- **Lifecycle Data:** PostgreSQL version lifecycle information was retrieved using the `geol` CLI with the command:

```
geol product extended psql -n0
```

- **Vulnerability Scanning:** Docker images for each major PostgreSQL version were scanned for vulnerabilities using the `trivy` CLI. An example command for a single version is:

```
trivy image postgres:18
```

4 Data Analysis

4.1 Version Lifecycle (`geol` data)

The first step is to determine which versions are officially supported.

An unsupported version is a **gateway to unpatched vulnerabilities.**

Table 1: PostgreSQL Version Lifecycle

Version	Release Date	Latest	Latest Release	End of Support (EOL)	Status
18	2025-09-25	18.2	2026-02-09	2030-11-14	✓ Supported
17	2024-09-26	17.7	2025-11-10	2029-11-08	✓ Supported
16	2023-09-14	16.11	2025-11-10	2028-11-09	✓ Supported
15	2022-10-13	15.15	2025-11-10	2027-11-11	✓ Supported
14	2021-09-30	14.20	2025-11-10	2026-11-12	✓ Supported
13	2020-09-24	13.23	2025-11-10	2025-11-13	× **Unsupported**
12	2019-10-03	12.22	2024-11-18	2024-11-21	× **Unsupported**
11	2018-10-18	11.22	2023-11-06	2023-11-09	× **Unsupported**
10	2017-10-05	10.23	2022-11-07	2022-11-10	× **Unsupported**
9.6	2016-09-29	9.6.24	2021-11-08	2021-11-11	× **Unsupported**

4.2 Vulnerability Analysis (`trivy` data)

The second step is to analyze the "attack surface" of Docker images. It's important to note that while we use major version Docker tags (e.g., `postgres:18`), these tags typically point to the latest patch release within that major version series (e.g., `postgres:18` currently refers to `postgres:18.1`).

Table 2 and Figure 1 show the results.

Table 2: Vulnerability Summary by Version

Docker Tag	Critical	High	Medium	Low	Total
`postgres:18.2`	1	17	39	102	159
`postgres:17`	0	6	9	98	117
`postgres:16`	0	6	9	98	113
`postgres:15`	0	6	9	98	113
`postgres:14`	0	6	9	98	113
`postgres:13`	0	6	9	98	113
`postgres:12`	9	70	100	124	305
`postgres:11`	7	84	52	49	192
`postgres:10`	7	84	52	49	192
`postgres:9.6`	10	97	57	49	213

4.3 Critical CVE Examples

To illustrate the concrete security risks of using unsupported versions, here are real critical vulnerabilities detected in PostgreSQL 12 (EOL November 2024):

Key Takeaway: These critical CVEs affect core dependencies (OpenSSL, libxml2, SQLite) bundled in the Docker image. Unsupported versions will never receive patches for these vulnerabilities, leaving systems permanently exposed.

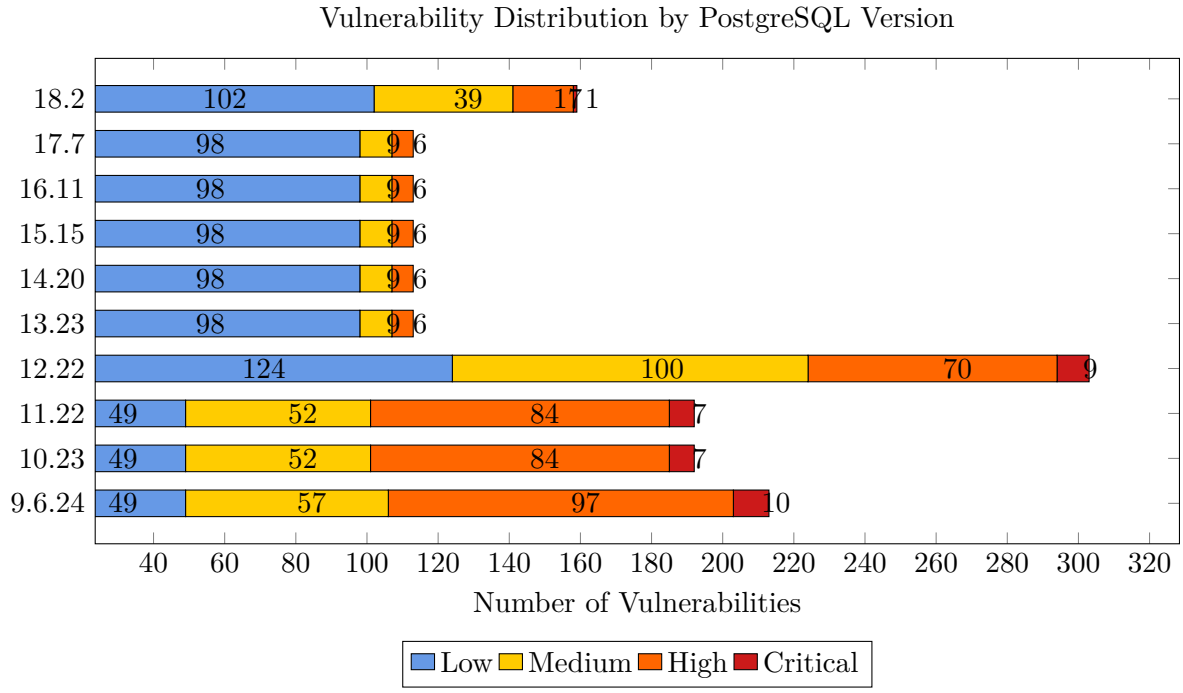

Figure 1: Comparison of vulnerabilities detected by `trivy`.

Table 3: Sample Critical CVEs in Unsupported Version (postgres:12)

CVE ID	Description
CVE-2025-15467	**OpenSSL**: Remote code execution or Denial of Service via oversized Initialization Vector in CMS parsing
CVE-2024-56171	**libxml2**: Use-After-Free vulnerability leading to potential remote code execution
CVE-2025-49794	**libxml**: Heap use-after-free (UAF) leads to Denial of Service (DoS)
CVE-2025-7458	**sqlite**: Integer overflow vulnerability enabling potential exploitation
CVE-2025-6965	**sqlite**: Integer truncation vulnerability in SQLite library

4.4 Version Lifecycle Timeline

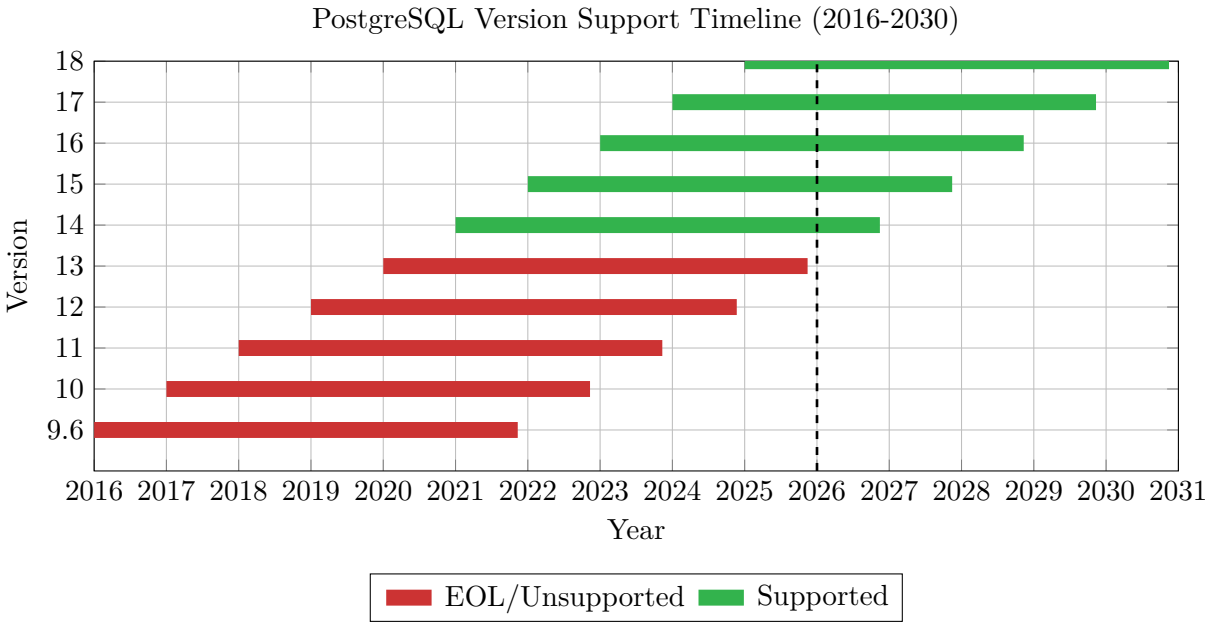

Figure 2: PostgreSQL version support periods showing 5-year lifecycle policy. Supported versions shown in green, EOL versions in red.

4.5 Vulnerability Heat Map

4.6 Cost-Benefit Analysis: Upgrade vs. Risk

Table 4: Upgrade Effort vs. Security Risk Assessment

Migration Path	Effort	Risk Reduction	Notes
13 → 14	Low	High	Single major version jump, similar features
12 → 14	Medium	Very High	2 major versions, eliminates 9 critical CVEs
11 → 15	Medium	Very High	4 major versions, significant feature changes
10 → 16	High	Critical	6 major versions, requires thorough testing
9.6 → 17/18	Very High	Critical	8-9 major versions, extensive compatibility review needed

Recommendation Strategy:

- **Step 1:** Upgrade to the minimum supported version (14) immediately to escape EOL status
- **Step 2:** Plan migration to version 16 or 17 for long-term support (EOL 2028-2029)
- **Step 3:** Establish regular minor update process to stay current within major version

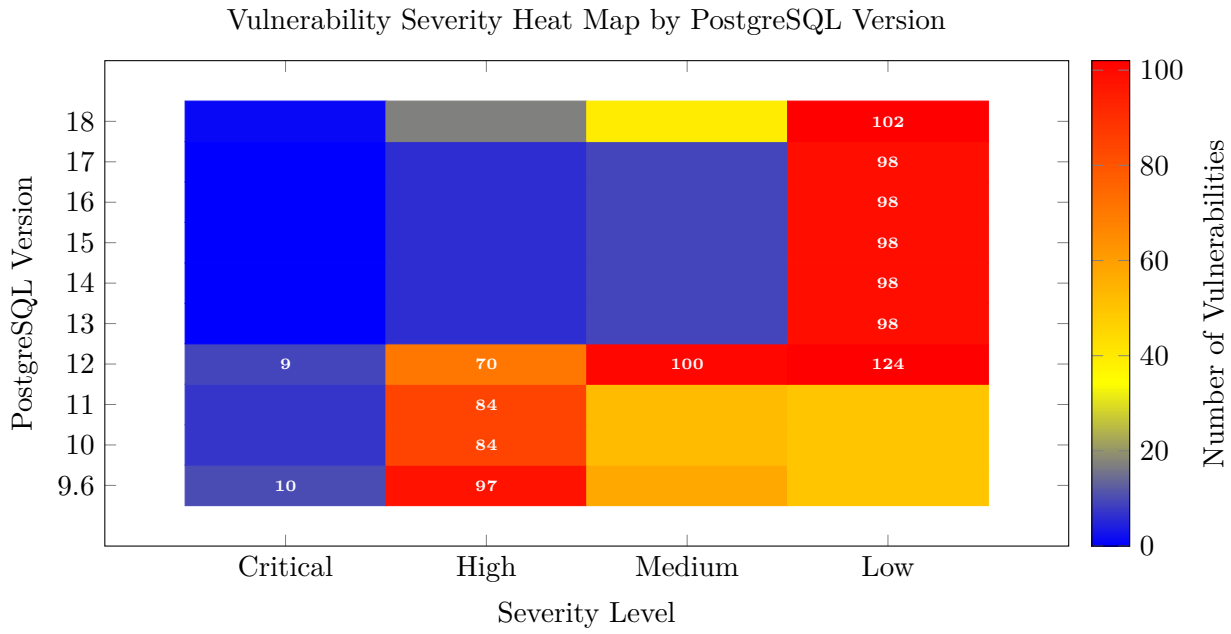

Figure 3: Heat map visualization showing vulnerability counts by severity and version. Darker colors indicate higher vulnerability counts.

5 Migration Guide

5.1 Recommended Upgrade Paths

PostgreSQL's 5-year support policy means careful planning is essential. Here are tested migration strategies:

For Production Systems on EOL Versions:

1. **Assess Current State**
   - Run `pg_dump` to ensure backup capability
   - Document custom configurations and extensions
   - Review application compatibility with target version
2. **Choose Target Version**
   - Minimum: PostgreSQL 14 (EOL 2026) - shortest migration path
   - Recommended: PostgreSQL 16 or 17 (EOL 2028-2029) - better long-term support
   - Latest: PostgreSQL 18 (EOL 2030) - maximum future-proofing
3. **Migration Strategy**

```
# Backup current database
pg_dump -U postgres -Fc mydb > mydb.dump

# Deploy new PostgreSQL version
docker run -d --name postgres-new postgres:17

# Restore data
pg_restore -U postgres -d mydb mydb.dump
```

4. **Testing Checklist**

   - Verify all queries execute correctly
   - Test application endpoints under load
   - Validate custom functions and stored procedures
   - Check extension compatibility (`pg_upgrade -check`)

Automation Best Practices:
Integrate version checking into CI/CD pipelines:

```
# Check PostgreSQL EOL status
geol check psql --version 13
# Exit code 1 if EOL, integrate into quality gates

# Scan Docker images before deployment
trivy image postgres:18 --severity CRITICAL,HIGH --exit-code 1
```

5.2 Vulnerability Comparison: 18.0 vs 18.1 vs 18.2

To illustrate the importance of patch releases, we compare the vulnerabilities found in `postgres:18` versions.

Table 5: Vulnerability Comparison: PostgreSQL 18.0 vs 18.1 vs 18.2

Docker Tag	Critical	High	Medium	Low	Total
`postgres:18.0`	4	30	68	102	204
`postgres:18.1`	1	17	39	102	159
`postgres:18.2`	1	17	39	102	159

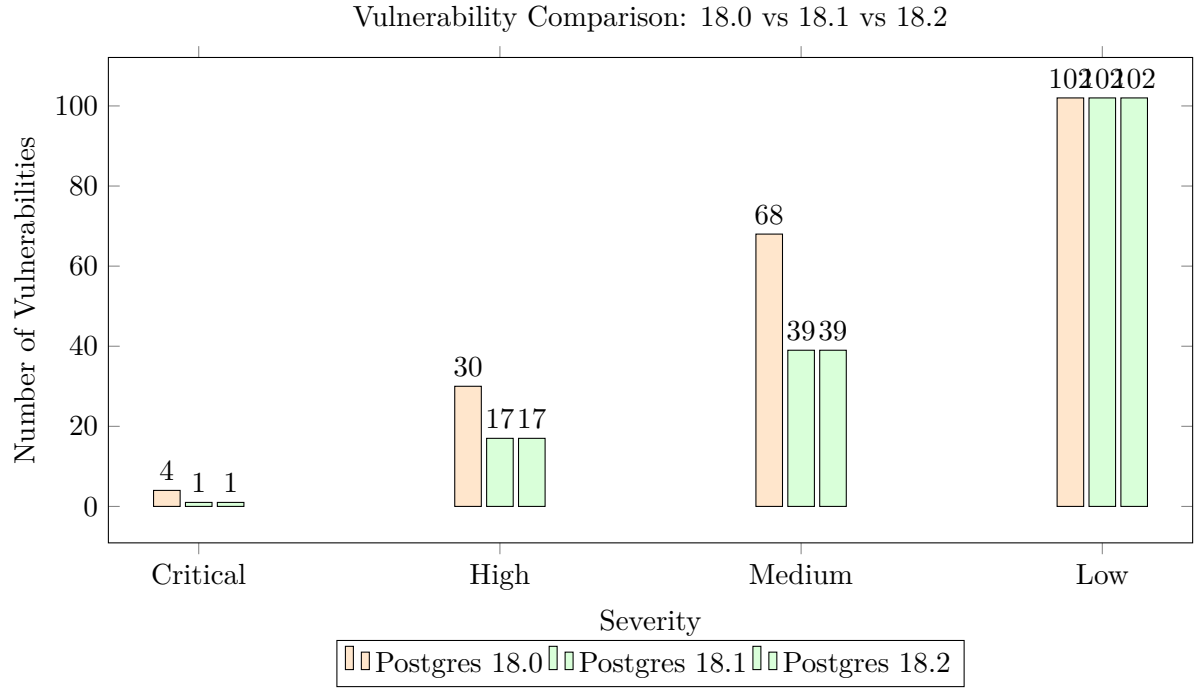

Figure 4: Comparison of vulnerabilities detected by `trivy` for PostgreSQL 18.0, 18.1 and 18.2.

The updates from 18.0 to 18.1 and then to 18.2 significantly reduced the number of critical, high, and medium vulnerabilities, demonstrating the effectiveness of minor releases in addressing security issues. For detailed changes, refer to the respective changelogs: 18.0, 18.1, and 18.2.

6 PostgreSQL Overview

PostgreSQL https://www.postgresql.org/, also known as Postgres, is a free and open-source relational database management system (RDBMS) emphasizing extensibility and technical standards compliance.

Postgres recommends that all users run the latest available minor release for whatever major version is in use.

The PostgreSQL Global Development Group supports a major version for 5 years after its initial release. After its five-year anniversary, a major version will have one last minor release containing any fixes and will be considered end-of-life (EOL) and no longer supported.

The Release roadmap https://www.postgresql.org/developer/roadmap/ lists upcoming minor and major releases. If the release team determines that a critical bug or security fix is too important to wait until the regularly scheduled minor release, it may make a release available outside the minor release roadmap.

A Feature Matrix https://www.postgresql.org/about/featurematrix/ documents feature availability against major releases.

7 Summary and conclusion

The combined data analysis is clear. Figure 1 strikingly illustrates this divergence:

- **The danger of unsupported versions:** Versions that have reached their end of life (12, 11, 10, 9.6) accumulate a dangerous number of vulnerabilities, including several **critical** ones.
- **The security of supported versions:** In contrast, images of maintained versions (14 to 18) show no critical vulnerabilities and a low, consistent risk profile. Note that PostgreSQL 13 is now unsupported.
- **Recommendation:** The choice of PostgreSQL version must be for an actively supported version. The security risk of using an obsolete version is real and high.

Tools like `geol` and `trivy` are essential in a modern DevSecOps approach. This analysis of PostgreSQL perfectly illustrates how abandoning software support directly leads to a drastic increase in security flaws. Using up-to-date versions is not just a recommendation, but a necessity for the security of any infrastructure.

8 Resources

- geol, the cli to efficiently manage EOLs like a boss
- geol - Gérer la fin de vie (notebookLM slideshow) v1.3.0 - "for dummies" edition
- geol 1.3.0 unboxing - the check command
- MVP Unboxing geol - a devops secops cli to manage EOLs and product lifecycle

- geol-showcase, A set of resources to showcase what could be achieved with geol, data-science, AI and devsecops tools
- PostgreSQL 18.1, 17.7, 16.11, 15.15, 14.20, and 13.23 Released!
- PostgreSQL EOL Data