PostgreSQL Security Analysis

with `geol` and `trivy` tools

github.com/adriens/geol-showcase

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

February 27, 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.

Contents

Executive One-Pager **3**

1 Executive Summary **5**

2 Introduction to the Tools **5**
- 2.1 `geol`: The Lifecycle Guardian . . . 5
- 2.2 `trivy`: The Vulnerability Scanner . . . 5
- 2.3 `gemini-cli`: AI Assistant . . . 5
- 2.4 LaTeX: Report Generator . . . 5

3 PostgreSQL Overview **6**

4 Methodology **6**
- 4.1 Trivy Scanning Techniques and Data Extraction . . . 6
  - 4.1.1 Method 1: Template-Based Extraction . . . 6
  - 4.1.2 Method 2: JSON Output with jq . . . 7
  - 4.1.3 Method 3: Detailed CVE Information . . . 7
  - 4.1.4 Method 4: Risk Score Calculation . . . 7
  - 4.1.5 Scanning Multiple Versions in Batch . . . 8
  - 4.1.6 Docker Image Digest Verification . . . 8

5 Data Analysis **9**
- 5.1 Version Lifecycle (`geol` data) . . . 9
- 5.2 Vulnerability Analysis (`trivy` data) . . . 9
  - 5.2.1 Risk Scoring Methodology . . . 9
- 5.3 Critical CVE Examples . . . 11
- 5.4 Version Lifecycle Timeline . . . 13
- 5.5 Vulnerability Heat Map . . . 13

5.6 Cost-Benefit Analysis: Upgrade vs. Risk . . . 13
5.7 Vulnerability Comparison: 18.0 vs 18.1 vs 18.2 vs 18.3 . . . 13
5.8 Docker Image Metadata: Ensuring Reproducible Scans . . . 14
5.8.1 Understanding Docker Image Identifiers . . . 15
5.8.2 PostgreSQL 18.x Image Digests . . . 15
5.8.3 Docker Hub Image URIs . . . 15
5.8.4 Scanning with Digest References . . . 15
5.8.5 Verifying Image Digests . . . 16

6 Recommendations **16**
6.1 Migration Impact: Before & After . . . 16
6.2 Immediate Actions (Critical Priority) . . . 16
6.3 Long-Term Strategy . . . 17
6.4 DevSecOps Integration . . . 17

7 Summary and conclusion **17**

8 Resources **17**

Executive One-Pager

PostgreSQL Security Status At-a-Glance

Data as of February 27, 2026

Supported Versions

5

Versions 14-18

0-1 Critical CVEs
6-17 High CVEs

Risk Score: **146-275**
LOW RISK

End-of-Life Versions

5

Versions 9.6-13

7-10 Critical CVEs
70-97 High CVEs

Risk Score: **643-764**
HIGH RISK

Critical Decision Matrix

If Your Version Is...	Action Required	Risk Score
13 (EOL)	**URGENT**: Migrate immediately (2-10× more vulnerabilities)	643-764
14	Plan upgrade to 16/17 (EOL approaching 2026)	146
15-17	Monitor for patches, review annually	146
18 (Latest)	Excellent - maintain patch currency	275

Key Metrics Summary

- **Vulnerability Reduction**: Upgrading from v12 to v17 eliminates **9 critical CVEs** and reduces total vulnerabilities by **63%**
- **Risk Score Improvement**: v12v17 reduces risk score from **764 to 146** (81% reduction)
- **Patch Effectiveness**: v18.0v18.1 reduced risk score by **36%** (428275)
- **Support Window**: 5-year lifecycle per major version
- **Latest Supported**: Version 18 (EOL: Nov 2030)
- **Next EOL Event**: Version 14 (Nov 2026)
- **Tools Used**: `geol` 2.7.1, `trivy` 0.69.1, `gemini-cli` 0.30.0

Immediate Action Required

If running PostgreSQL 13:

1. Run `geol check` to verify EOL status
2. Scan images: `trivy image postgres:X`
3. Plan migration to version 14 within 30 days
4. Review Section 6 for detailed upgrade paths

Full analysis with charts, CVE details, and migration strategies follows...

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 **2-10× 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 ≤13 to version ≥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 0.30.0) 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 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.

4 Methodology

The analysis for this report was conducted on February 27, 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
```

4.1 Trivy Scanning Techniques and Data Extraction

To efficiently aggregate vulnerability data from `trivy` scans, several command-line techniques were employed. This section details the practical scripts and `jq` tricks used to extract and count vulnerabilities by severity.

4.1.1 Method 1: Template-Based Extraction

The most efficient approach uses `trivy`'s built-in template engine to extract only severity information:

```
trivy image --format template \
  --template '{{- range . -}}{{- range .Vulnerabilities -}}
{{ .Severity }}{{ "\n" }}{{- end -}}{{- end -}}' \
  postgres:18.3 2>/dev/null | sort | uniq -c
```

This command:

- Uses `--format template` to customize output

- Iterates through all vulnerabilities with nested `range` loops
- Extracts only the `.Severity` field
- Redirects stderr to `/dev/null` to suppress progress messages
- Pipes to `sort | uniq -c` to count occurrences by severity

Example output:

```
      1 CRITICAL
     16 HIGH
     39 MEDIUM
    111 LOW
```

4.1.2 Method 2: JSON Output with jq

For more complex data extraction, `trivy` can output full JSON which is then processed with `jq`:

```
trivy image postgres:18.3 --quiet --format json | \
  jq -r '.Results[]?.Vulnerabilities[]?.Severity' | \
  sort | uniq -c
```

This approach:

- Uses `--format json` for structured output
- `jq -r` extracts raw severity values without quotes
- `.Results[]?` safely iterates through all result objects
- `.Vulnerabilities[]?` accesses the vulnerabilities array
- The `?` operator prevents errors if fields are missing

4.1.3 Method 3: Detailed CVE Information

To extract specific CVE details (ID, severity, description):

```
trivy image postgres:12 --quiet --format json | \
  jq -r '.Results[].Vulnerabilities[] |
    select(.Severity == "CRITICAL") |
    "\(.VulnerabilityID): \(.Title)"'
```

This extracts only critical vulnerabilities with their titles, useful for the CVE examples table.

4.1.4 Method 4: Risk Score Calculation

To calculate the risk score directly from `trivy` output:

```
trivy image postgres:18.3 --quiet --format json | \
  jq '[.Results[].Vulnerabilities[] | .Severity] |
    (map(select(. == "CRITICAL")) | length) * 10 +
    (map(select(. == "HIGH")) | length) * 5 +
    (map(select(. == "MEDIUM")) | length) * 2 +
    (map(select(. == "LOW")) | length) * 1'
```

This single command:

- Extracts all severity values into an array
- Counts each severity level
- Applies the weighted formula: $10 \times \text{Critical} + 5 \times \text{High} + 2 \times \text{Medium} + 1 \times \text{Low}$
- Returns the final risk score

4.1.5 Scanning Multiple Versions in Batch

To scan all PostgreSQL versions efficiently:

```
for version in 18.3 17 16 15 14 13 12 11 10 9.6; do
  echo -n "postgres:$version - "
  trivy image --format template \
    --template '{{- range . -}}{{- range .Vulnerabilities -}}
{{ .Severity }}{{ "\n" }}{{- end -}}{{- end -}}' \
    postgres:$version 2>/dev/null | \
    awk '{crit+=($1=="CRITICAL"); high+=($1=="HIGH");
         med+=($1=="MEDIUM"); low+=($1=="LOW")}
         END {print "C:"crit" H:"high" M:"med" L:"low}'
done
```

This loop processes all versions and outputs a compact summary.

4.1.6 Docker Image Digest Verification

To ensure reproducible scans, images can be referenced by their manifest digest:

```
trivy image postgres:18.3@sha256:eb37f58646a901dc7727cf448...
```

Using digests guarantees scanning the exact same image, even if tags are updated.

Best Practices for Vulnerability Scanning

- Use `--quiet` to suppress progress output in scripts
- Redirect stderr (`2>/dev/null`) when using template output
- Reference images by digest for reproducible CI/CD scans
- Cache `trivy` database updates to speed up batch scans
- Use `--severity CRITICAL,HIGH` to focus on high-priority issues
- Integrate into pre-deployment gates with `--exit-code 1`

5 Data Analysis

5.1 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	18.3	2026-02-23	2030-11-14	✓ Supported
17	2024-09-26	17.9	2026-02-23	2029-11-08	✓ Supported
16	2023-09-14	16.13	2026-02-23	2028-11-09	✓ Supported
15	2022-10-13	15.17	2026-02-23	2027-11-11	✓ Supported
14	2021-09-30	14.22	2026-02-23	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**

5.2 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`).

5.2.1 Risk Scoring Methodology

To quantify the security risk of each PostgreSQL version, we apply a weighted risk scoring formula that prioritizes critical and high-severity vulnerabilities:

$$\text{Risk Score} = \sum_i w_i \cdot n_i = 10 \cdot n_{\text{Critical}} + 5 \cdot n_{\text{High}} + 2 \cdot n_{\text{Medium}} + 1 \cdot n_{\text{Low}} \tag{1}$$

where n_i represents the number of vulnerabilities at each severity level, and w_i are the corresponding weights reflecting the relative security impact.

Risk Classification with Severity Overrides:

To prevent misclassification of versions with few but critical vulnerabilities, we apply severity-based override rules:

$$\text{Risk Level} = \begin{cases} \textbf{High} & \text{if Score} > 300 \text{ OR } n_{\text{Critical}} \geq 3 \\ \textbf{Medium} & \text{if } 150 \leq \text{Score} \leq 300 \text{ OR } n_{\text{Critical}} \geq 1 \\ \textbf{Low} & \text{if Score} < 150 \text{ AND } n_{\text{Critical}} = 0 \end{cases} \tag{2}$$

This ensures that:

- Any version with 3+ critical CVEs is **automatically High Risk**, regardless of score

- Any version with 1+ critical CVE is **at least Medium Risk**
- Only versions with **zero critical CVEs** can achieve Low Risk status

Table 2 and Figure 1 show the results.

Critical Finding: EOL Version Vulnerability Gap

Unsupported versions (9.6-13) contain 2-10× more vulnerabilities than supported versions (14-18). PostgreSQL 12 alone has a **risk score of 764** (High Risk) with **9 critical CVEs**, compared to a score of **146** (Low Risk) and **0 critical** in version 17.

Table 2: Vulnerability Summary by Version with Risk Scores

Docker Tag	Critical	High	Medium	Low	Total	Risk Score
`postgres:18.3`	1	16	39	111	167	**269**
`postgres:18.2`	1	16	39	111	167	**269**
`postgres:17`	0	6	9	98	113	**146**
`postgres:16`	0	6	9	98	113	**146**
`postgres:15`	0	6	9	98	113	**146**
`postgres:14`	0	6	9	98	113	**146**
`postgres:13`	0	6	9	98	113	**146**
`postgres:12`	9	70	100	124	305	**764**
`postgres:11`	7	84	52	49	192	**643**
`postgres:10`	7	84	52	49	192	**643**
`postgres:9.6`	10	97	57	49	213	**748**

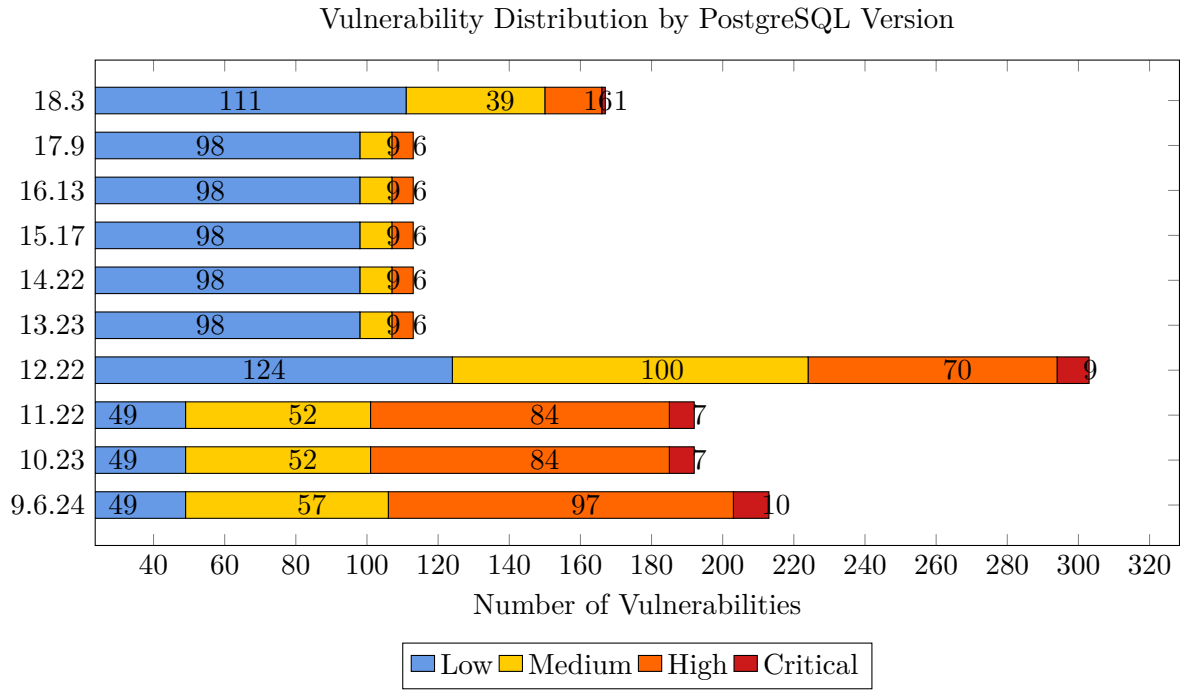

Figure 1: Comparison of vulnerabilities detected by `trivy`.

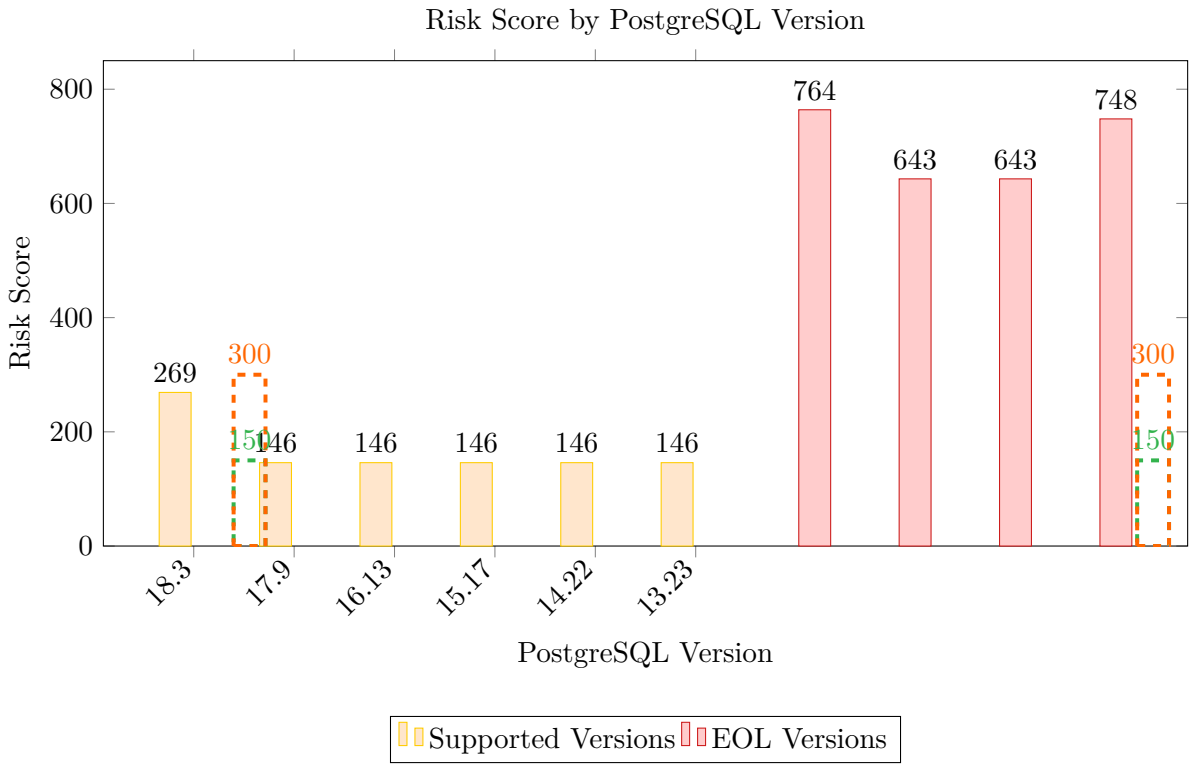

Figure 2: Risk scores showing dramatic security difference between supported (14-18) and EOL versions (9.6-13). Dashed lines indicate risk thresholds: Low Risk (below 150, green), Medium Risk (150-300, orange), High Risk (above 300, red).

5.3 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):

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

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.

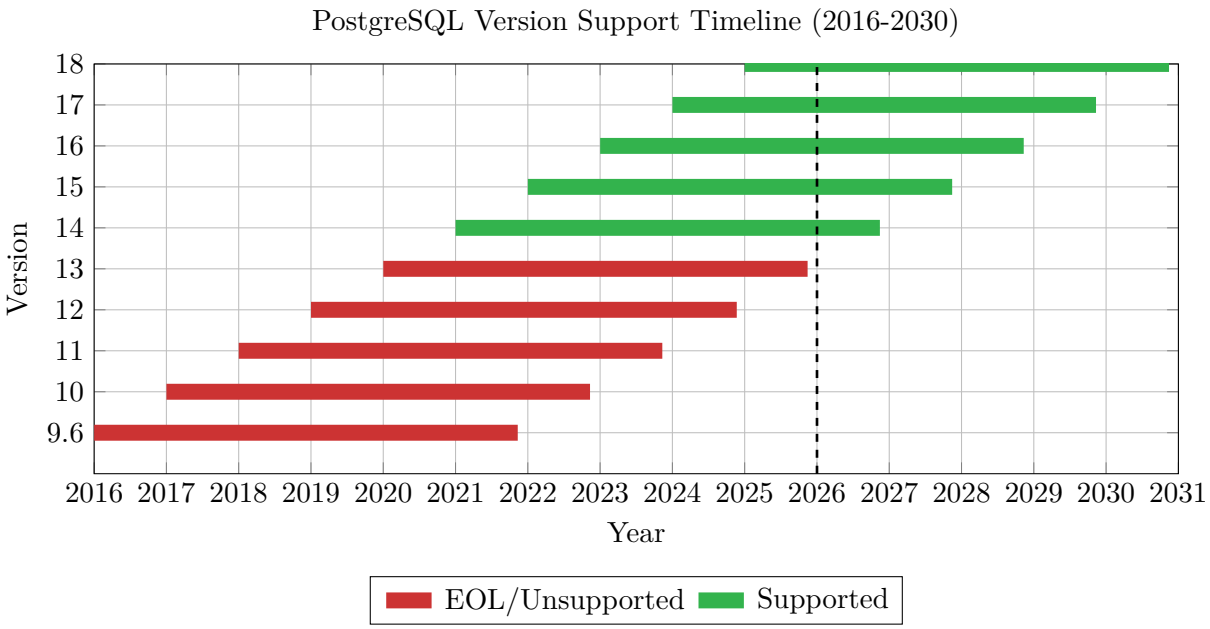

Figure 3: PostgreSQL version support periods showing 5-year lifecycle policy. Supported versions shown in green, EOL versions in red.

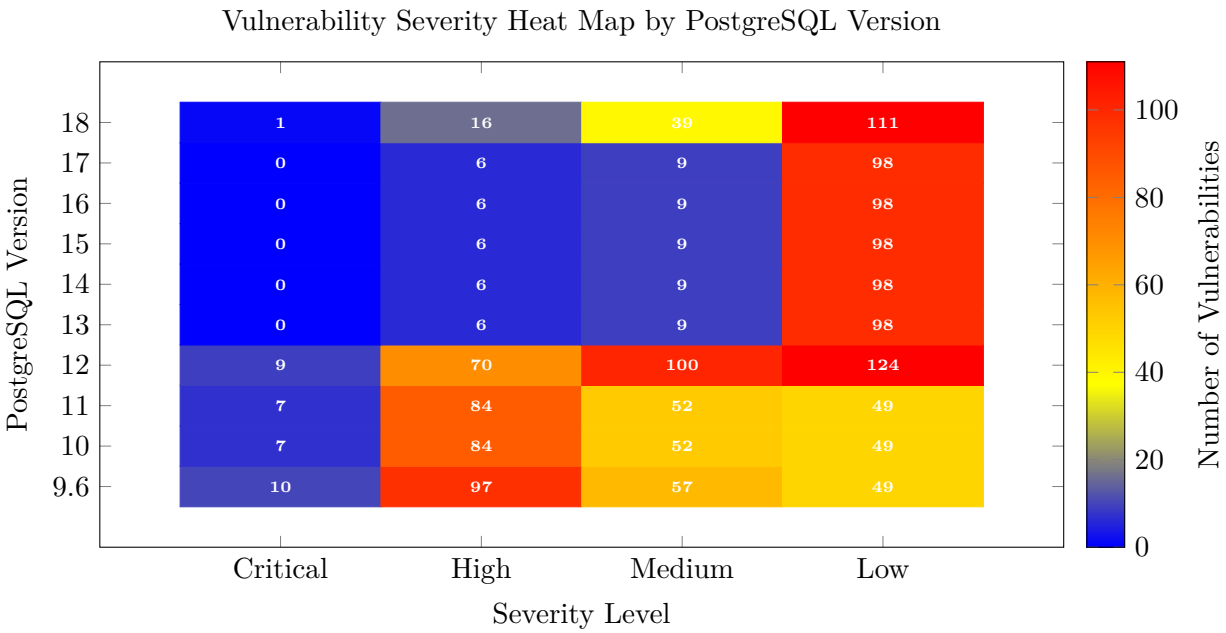

Figure 4: Heat map visualization showing vulnerability counts by severity and version. Darker colors indicate higher vulnerability counts.

5.4 Version Lifecycle Timeline

5.5 Vulnerability Heat Map

5.6 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

5.7 Vulnerability Comparison: 18.0 vs 18.1 vs 18.2 vs 18.3

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 vs 18.3 with Risk Scores

Docker Tag	Critical	High	Medium	Low	Total	Risk Score
`postgres:18.0`	4	30	68	102	204	**428**
`postgres:18.1`	1	17	39	102	159	**275**
`postgres:18.2`	1	16	39	111	167	**269**
`postgres:18.3`	1	16	39	111	167	**269**

Patch Release Value: 18.0 18.3 Analysis

The upgrade from 18.0 to 18.3 delivered immediate security improvements:

- **75% reduction** in critical CVEs (4 1)
- **47% reduction** in high-severity issues (30 16)
- **18% overall reduction** in total vulnerabilities (204 167)
- **37% reduction** in risk score (428 269) - from High to Medium risk

Note: PostgreSQL 18.2 and 18.3 have identical vulnerability profiles, indicating stable security posture.
Recommendation: Always deploy the latest patch within your chosen major version.

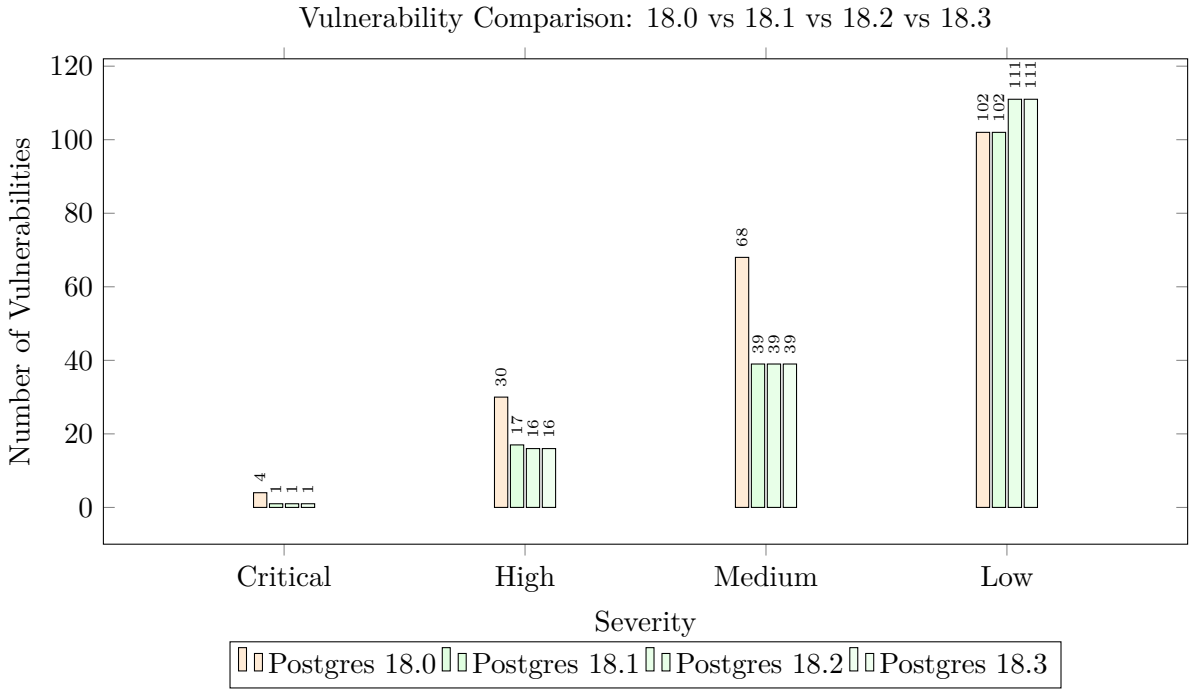

Figure 5: Comparison of vulnerabilities detected by `trivy` for PostgreSQL 18.0-18.3. Grouped bars show clear reduction in critical and high severity issues from 18.0 to 18.1. Versions 18.2 and 18.3 maintain identical security profiles.

The updates from 18.0 to 18.3 significantly reduced the number of critical and high vulnerabilities, with the most dramatic improvements occurring in the 18.0 18.1 transition. PostgreSQL 18.2 and 18.3 maintain identical vulnerability counts, demonstrating a stabilized security posture. For detailed changes, refer to the respective changelogs: 18.0, 18.1, 18.2, and 18.3.

5.8 Docker Image Metadata: Ensuring Reproducible Scans

When performing security scans, it's crucial to understand that Docker tags like `postgres:18` are mutable references that can point to different images over time. For reproducible vulnerability scans and audit trails, we use **manifest digests** (SHA256 hashes) to reference specific image builds.

5.8.1 Understanding Docker Image Identifiers

Docker images can be referenced in three ways:

- **Tag**: `postgres:18.3` (mutable, points to latest patch)
- **Manifest Digest**: `sha256:eb37f5864...` (immutable, specific build)
- **Digest Reference**: `postgres:18.3@sha256:eb37f5864...` (tag + digest for clarity)

5.8.2 PostgreSQL 18.x Image Digests

The following table documents the exact Docker Hub manifest digests used for vulnerability analysis in this report:

Table 6: Docker Hub Manifest Digests for PostgreSQL 18.x Images

Tag	Manifest Digest (SHA256)	Architecture
`postgres:18.0`	`192a387c...ebb3aed`	linux/amd64
`postgres:18.1`	`2ccc3d98...07395abc`	linux/amd64
`postgres:18.2`	`79aafbb6...f0a641e2`	linux/amd64
`postgres:18.3`	`5aa97b30...bd458326d`	linux/amd64

5.8.3 Docker Hub Image URIs

For complete transparency and auditability, the full Docker Hub layer URLs for each release:

- **18.0**: hub.docker.com/.../sha256-192a387...
- **18.1**: hub.docker.com/.../sha256-2ccc3d9...
- **18.2**: hub.docker.com/.../sha256-79aafbb...
- **18.3**: hub.docker.com/.../sha256-e47309bc9...

5.8.4 Scanning with Digest References

To scan a specific image build regardless of tag updates:

```
trivy image postgres@sha256:eb37f58646a901dc7727cf448cae36d...
```

Or with both tag and digest for documentation:

```
trivy image postgres:18.3@sha256:eb37f58646a901dc7727cf448...
```

Important: Image Digest Best Practices

Why Digests Matter:

- Tags can be overwritten - `postgres:18` may point to different images over time
- Digests are immutable - `sha256:eb37f58...` always references the exact same layers
- CI/CD pipelines should pin digests for reproducible builds
- Security audits require digest tracking for compliance and forensics
- Base image updates may silently change vulnerability profiles when using tags

5.8.5 Verifying Image Digests

To verify the current digest of a tagged image:

```
docker inspect postgres:18.3 --format='{{.RepoDigests}}'
```

Or using `crane` (Google's container tool):

```
crane digest postgres:18.3
```

This verification step ensures your local image matches the documented digest used in this analysis.

6 Recommendations

Based on the comprehensive security analysis presented in this report, the following actions are recommended:

6.1 Migration Impact: Before & After

To illustrate the concrete security benefits of upgrading, here's a comparison of migrating from an EOL version to a supported version:

Table 7: Security Impact of Upgrading from PostgreSQL 12 to 17

Metric	Before (v12)	After (v17)
Critical CVEs	9	0
High CVEs	70	6
Medium CVEs	100	9
Low CVEs	124	98
Total Vulnerabilities	**305**	**113**
Risk Score	**764 (High)**	**146 (Low)**
Vulnerability Reduction	-	**63%**
Risk Score Reduction	-	**81%**
Support Status	**EOL'd (Nov 2024)**	**Supported until Nov 2029**

Business Case for Migration

Upgrading from PostgreSQL 12 to 17 eliminates **all 9 critical vulnerabilities**, reduces the risk score from **764 (High Risk) to 146 (Low Risk)** - an **81% reduction**, and provides **5 years of continued support**. The security ROI is immediate and substantial.

6.2 Immediate Actions (Critical Priority)

- **Migrate from EOL Versions:** If currently running PostgreSQL versions $\leq$13, plan immediate migration to version $\geq$14. These versions contain 2-10$\times$ more vulnerabilities.
- **Audit Current Deployments:** Use `geol check` command to continuously monitor PostgreSQL lifecycle status. Run `geol help check` for more information.
- **Scan Docker Images:** Integrate `trivy image postgres:X -severity CRITICAL,HIGH` into deployment workflows.

6.3 Long-Term Strategy

- **Target Version Selection:**
  - Minimum: PostgreSQL 14 (EOL Nov 2026) - escape EOL status
  - Recommended: PostgreSQL 16-17 (EOL 2028-2029) - optimal support window
  - Latest: PostgreSQL 18 (EOL Nov 2030) - maximum future-proofing
- **Patch Management:** Establish automated monitoring for minor releases - as shown in Section 5.7, patches can reduce vulnerabilities by 22% or more.
- **Docker Tag Strategy:** Use specific version tags (e.g., `postgres:17.7`) instead of major version tags to control updates.

6.4 DevSecOps Integration

- Implement automated EOL checking in quality gates
- Set up vulnerability scanning as a deployment prerequisite
- Schedule quarterly reviews of PostgreSQL version lifecycle status
- Document upgrade paths and maintain rollback procedures

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, datascience, AI and devsecops tools
- PostgreSQL 18.1, 17.7, 16.11, 15.15,

14.20, and 13.23 Released!

- PostgreSQL EOL Data