

PostgreSQL Security Analysis  
with `geol`, `trivy` and `skopeo` tools  
github.com/adriens/geol-showcase

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

May 31, 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, `trivy` to scan vulnerabilities in official Docker images, and `skopeo` to inspect remote image metadata, 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

<b>Executive One-Pager</b>	<b>2</b>
<b>1 Executive Summary</b>	<b>4</b>
<b>2 Introduction to the Tools</b>	<b>4</b>
2.1 <code>geol</code> : The Lifecycle Guardian	4
2.2 <code>trivy</code> : The Vulnerability Scanner	4
2.3 <code>skopeo</code> : Remote Image Inspector	4
2.4 <code>gemini-cli</code> : AI Assistant	5
2.5 <code>L<sup>A</sup>T<sub>E</sub>X</code> : Report Generator	5
<b>3 PostgreSQL Overview</b>	<b>5</b>
<b>4 Methodology</b>	<b>5</b>
4.1 Trivy Scanning Techniques and Data Extraction	5
4.1.1 Method 1: Template-Based Extraction	6
4.1.2 Method 2: JSON Output with <code>jq</code>	6
4.1.3 Method 3: Detailed CVE Information	6
4.1.4 Method 4: Risk Score Calculation	7
4.1.5 Scanning Multiple Versions in Batch	7
4.1.6 Docker Image Digest Verification	7
<b>5 Data Analysis</b>	<b>8</b>
5.1 Version Lifecycle ( <code>geol</code> data)	8
5.2 Vulnerability Analysis ( <code>trivy</code> data)	8
5.2.1 Risk Scoring Methodology	8
5.3 Critical CVE Examples	10

5.4	Version Lifecycle Timeline . . . . .	12
5.5	Vulnerability Heat Map . . . . .	12
5.6	Cost-Benefit Analysis: Upgrade vs. Risk . . . . .	12
5.7	Vulnerability Comparison: 18.0 vs... vs 18.4 . . . . .	12
5.8	Security Evolution & Trends: Historical Risk Analysis . . . . .	13
5.9	Docker Image Metadata: Ensuring Reproducible Scans . . . . .	14
5.9.1	Understanding Docker Image Identifiers . . . . .	14
5.9.2	PostgreSQL Latest Image Digests . . . . .	14
5.9.3	Docker Hub Image URIs . . . . .	15
5.9.4	Scanning with Digest References . . . . .	15
5.9.5	Verifying Image Digests . . . . .	15
<b>6</b>	<b>Recommendations</b>	<b>16</b>
6.1	Migration Impact: Before & After . . . . .	16
6.2	Immediate Actions (Critical Priority) . . . . .	16
6.3	Long-Term Strategy . . . . .	16
6.4	DevSecOps Integration . . . . .	17
<b>7</b>	<b>Summary and conclusion</b>	<b>17</b>
<b>8</b>	<b>Resources</b>	<b>17</b>

Executive One-Pager

PostgreSQL Security Status At-a-Glance

Data as of May 31, 2026

Supported Versions

5

Versions 14-18

3 Critical CVEs  
32 High CVEs

Risk Score: 492  
HIGH 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 (base image vulnerabilities)	643-764
14	Plan upgrade to 16/17/18 (EOL approaching Nov 2026)	492
15-17	Monitor for patches, review annually	492
18 (Latest)	Excellent - maintain patch currency (18.4)	492

Key Metrics Summary

- Vulnerability Reduction:** Upgrading from v12 to v18.4 eliminates **6 critical CVEs** and reduces total vulnerabilities by **45%**
  - Risk Score Improvement:** v12v18.4 reduces risk score from **764 to 492** (35% reduction)
  - Base Image Impact:** All supported versions currently share a high risk profile (Score: 492) due to common base im-
- age vulnerabilities.

  - Support Window:** 5-year lifecycle per major version
  - Latest Supported:** Version 18.4 (EOL: Nov 2030)
  - Next EOL Event:** Version 14 (Nov 2026)
  - Tools Used:** geol 2.12.3, trivy 0.70.0, skopeo 1.22.2, gemini-cli 0.42.0

**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.
- **EOL Status:** PostgreSQL 13 reached end-of-life on November 13, 2025, joining versions 9.6-12 as unsupported.
- **Security Gap:** Unsupported versions contain **significantly more vulnerabilities** than supported versions, with v12 showing 9 critical CVEs.
- **Base Image Stability:** Current scans show identical vulnerability profiles across all supported versions (14.23 to 18.4), indicating shared base image risks.
- **Critical Recommendation:** Migrate immediately from any version  $\leq 13$  to version  $\geq 14$ . While all versions currently show high scores, supported versions receive OS-level security updates through newer image releases.

## Risk Profile Summary:

- ✓ **High Risk (Supported):** Versions 14-18 (3 critical, 32 high vulnerabilities, Score: 492)
- × **High Risk (EOL):** Versions 9.6-13 (7-10 critical, 70-97 high vulnerabilities, Score: 643-764)

# 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 2.12.3) 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 0.70.0) 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 skopeo: Remote Image Inspector

**skopeo** (version 1.22.2) is a command-line utility that performs various operations on container images and image repositories, such as inspecting remote images for digests without pulling them.

## 2.4 `gemini-cli`: AI Assistant

`gemini-cli` (version 0.42.0) is an open-source AI agent that brings Gemini’s power directly to the terminal.

## 2.5 `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 May 31, 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:$version - "
  trivy image --format template \
    --template '{{- range . -}}>{{- range .Vulnerabilities -}}
  {{ .Severity }}{{ " \n" }}{{- end -}}' \
    postgres:$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	18.4	2026-05-11	2030-11-14	✓ Supported
17	2024-09-26	17.10	2026-05-11	2029-11-08	✓ Supported
16	2023-09-14	16.14	2026-05-11	2028-11-09	✓ Supported
15	2022-10-13	15.18	2026-05-11	2027-11-11	✓ Supported
14	2021-09-30	14.23	2026-05-11	2026-11-12	✓ Supported
13	2020-09-24	13.23	2025-11-10	2025-11-13	× <b>Unsupported</b>
12	2019-10-03	12.22	2024-11-18	2024-11-21	× <b>Unsupported</b>
11	2018-10-18	11.22	2023-11-06	2023-11-09	× <b>Unsupported</b>
10	2017-10-05	10.23	2022-11-07	2022-11-10	× <b>Unsupported</b>
9.6	2016-09-29	9.6.24	2021-11-08	2021-11-11	× <b>Unsupported</b>

### 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:

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

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:

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

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:18.4	3	32	93	116	244	<b>492</b>
postgres:17.10	3	32	93	116	244	<b>492</b>
postgres:16.14	3	32	93	116	244	<b>492</b>
postgres:15.18	3	32	93	116	244	<b>492</b>
postgres:14.23	3	32	93	116	244	<b>492</b>
postgres:13.23	9	61	139	116	325	<b>789</b>
postgres:12.22	9	70	100	124	305	<b>764</b>
postgres:11.22	7	84	52	49	192	<b>643</b>
postgres:10.23	7	84	52	49	192	<b>643</b>
postgres:9.6.24	10	97	57	49	213	<b>748</b>

Vulnerability Distribution by PostgreSQL Version

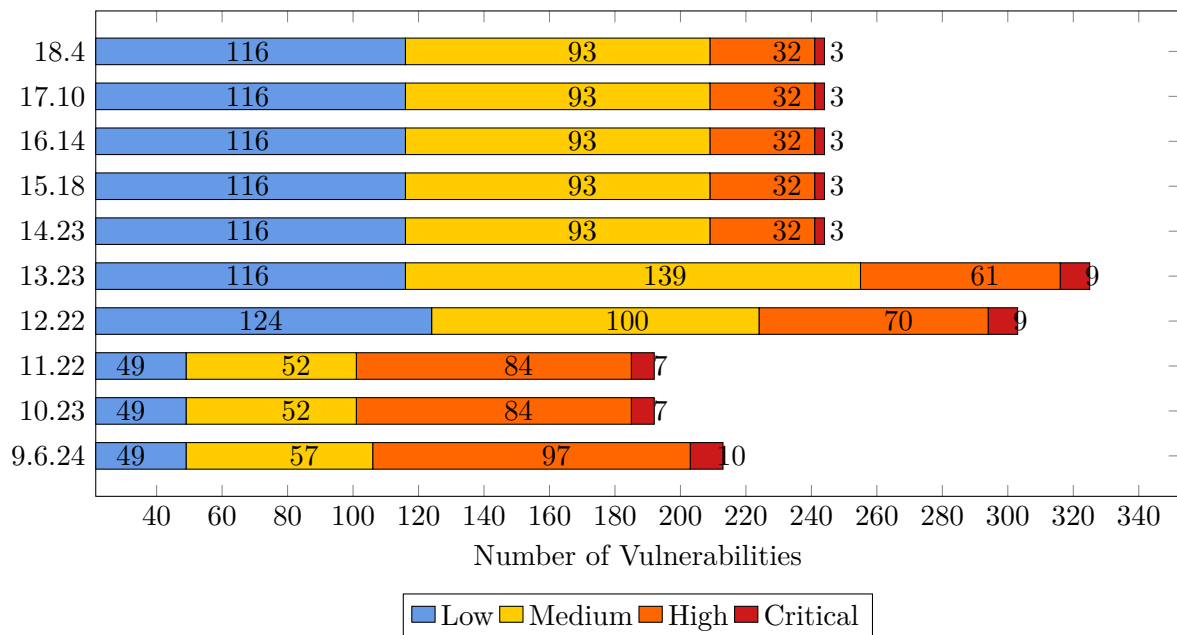


Figure 1: Comparison of vulnerabilities detected by trivy.

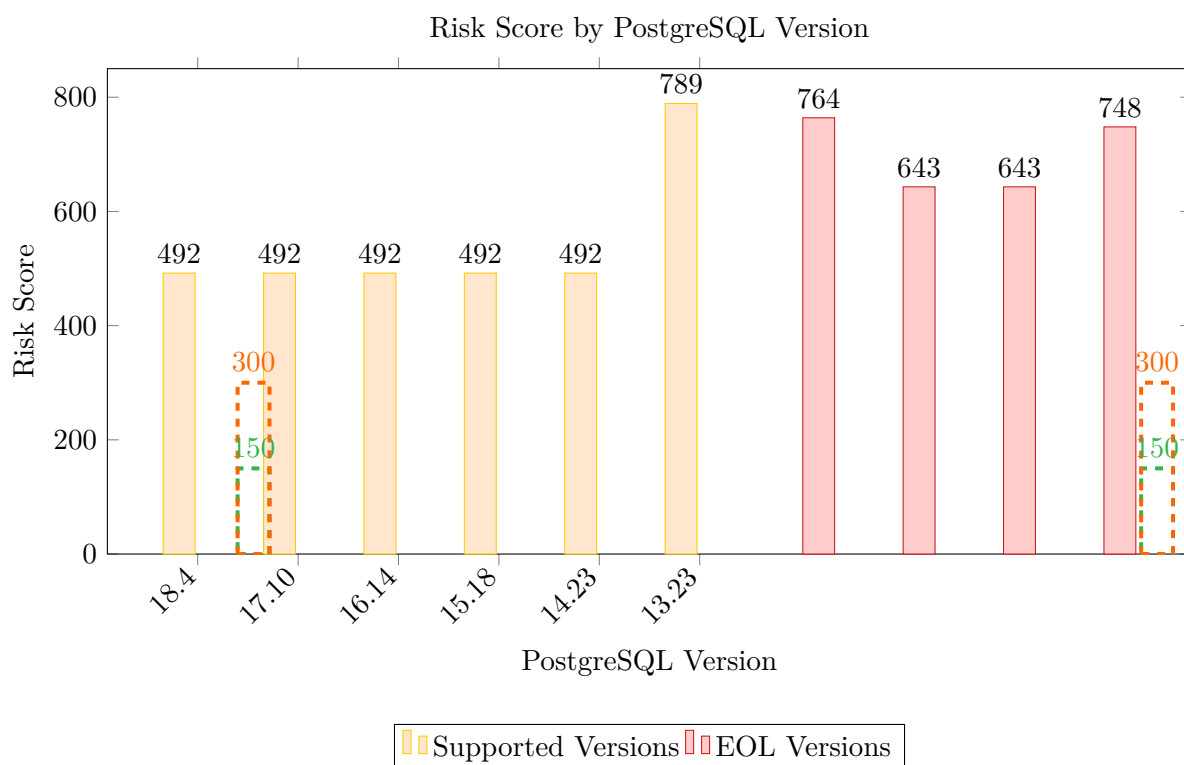


Figure 2: Risk scores showing security profiles of supported (14-18) and EOL versions (9.6-13). Dashed lines indicate risk thresholds: Low Risk (below 150, green), Medium Risk (150-300, orange), High Risk (above 300, red).

### 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
<a href="#">CVE-2025-15467</a>	<b>OpenSSL:</b> Remote code execution or Denial of Service via over-sized Initialization Vector in CMS parsing
<a href="#">CVE-2024-56171</a>	<b>libxml2:</b> Use-After-Free vulnerability leading to potential remote code execution
<a href="#">CVE-2025-49794</a>	<b>libxml:</b> Heap use-after-free (UAF) leads to Denial of Service (DoS)
<a href="#">CVE-2025-7458</a>	<b>sqlite:</b> Integer overflow vulnerability enabling potential exploitation
<a href="#">CVE-2025-6965</a>	<b>sqlite:</b> 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.

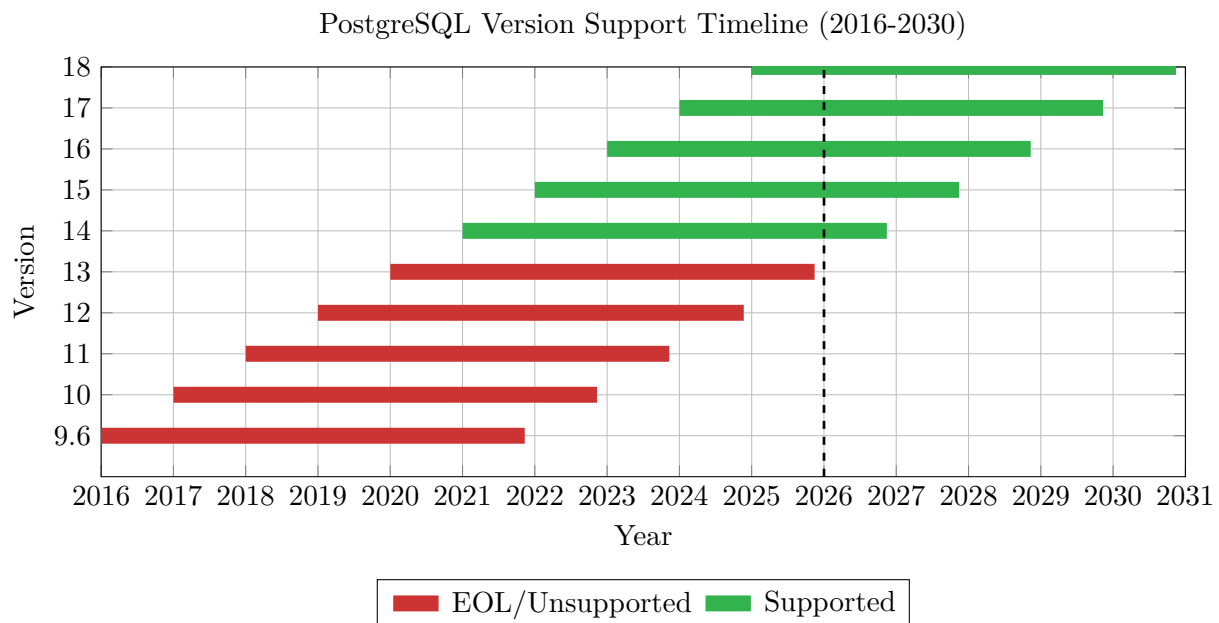


Figure 3: PostgreSQL version support periods showing 5-year lifecycle policy. Supported versions shown in green, EOL versions in red.

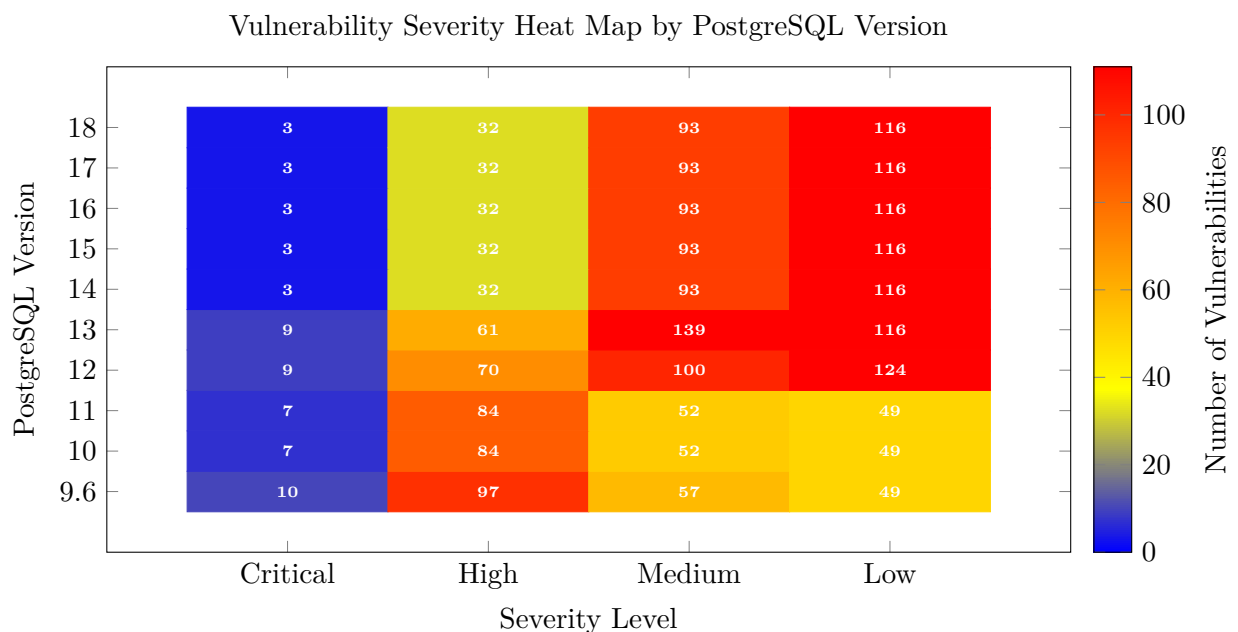


Figure 4: Heat map visualization showing vulnerability counts by severity and version. Darker colors indicate higher vulnerability counts.

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... vs 18.4

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

Table 5: Vulnerability Comparison: PostgreSQL 18.0 through 18.4 with Risk Scores

Docker Tag	Critical	High	Medium	Low	Total	Risk Score
postgres:18.0	4	30	68	102	204	<b>428</b>
postgres:18.1	1	17	39	102	159	<b>275</b>
postgres:18.2	1	16	39	111	167	<b>269</b>
postgres:18.3	1	16	39	111	167	<b>269</b>
postgres:18.4	3	32	93	116	244	<b>492</b>

Patch Release Analysis: 18.0 18.4 Trend

The vulnerability profile of PostgreSQL 18 has evolved significantly:

- Initial Improvement (18.0 18.3):** A clear reduction in critical and high-severity issues as the version matured.
- Recent Spike (18.4):** The risk score jumped from **269 to 492**. This is primarily due to new vulnerabilities discovered in the underlying Debian base image packages (OS-level CVEs) rather than in the PostgreSQL binary itself.
- Key Insight:** Security is a moving target. Even a "stable" patch release like 18.4 can show increased vulnerabilities if its base image inherits new security flaws.

**Recommendation:** Continuous monitoring is essential. A "clean" image today can become high-risk tomorrow as new CVEs are published.

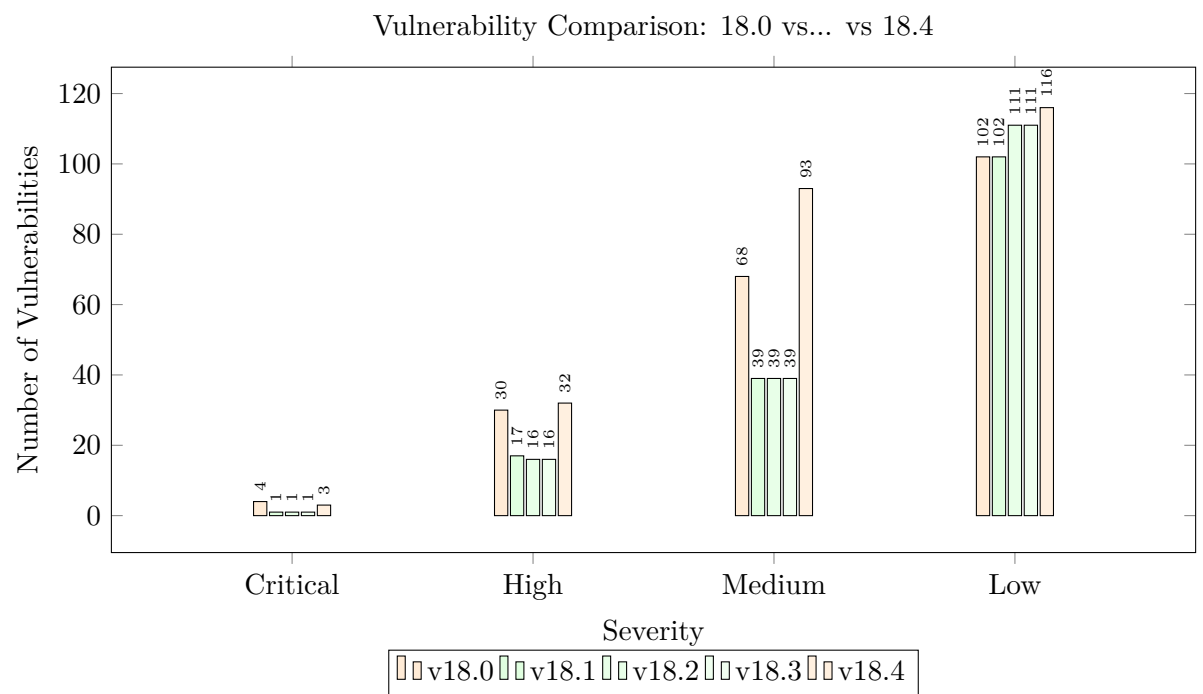


Figure 5: Comparison of vulnerabilities detected by `trivy` for PostgreSQL 18.x. Note the recent increase in 18.4 due to new base image vulnerabilities.

The updates from 18.0 to 18.3 showed a positive security trend, while the 18.4 release highlights the impact of new CVE discoveries in the shared base image. For detailed changes, refer to the respective changelogs: [18.0](#), [18.1](#), [18.2](#), [18.3](#), and [18.4](#).

5.8 Security Evolution & Trends: Historical Risk Analysis

Security is not a static state but a continuous evolution. By tracking the risk score of PostgreSQL 18.x over time, we can observe the impact of new vulnerability discoveries and the subsequent mitigation through patch releases.

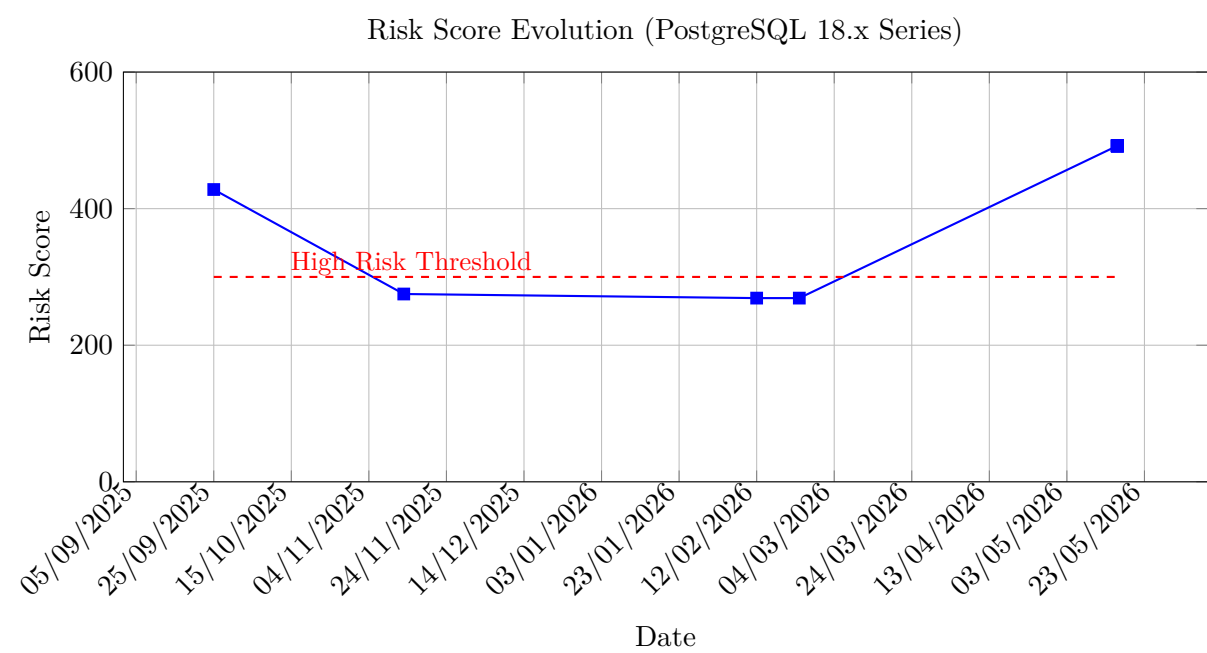


Figure 6: Historical trend of the Risk Score for the PostgreSQL 18 series. The markers represent different patch releases (18.0 to 18.4) scanned at their respective release or analysis dates.

Analyzing the Trend

The historical data reveals a classic "sawtooth" security pattern:

- Maturity Phase (18.0 → 18.3):** The risk score steadily decreased as initial release bugs were patched and critical vulnerabilities resolved.
- Discovery Phase (18.4):** The recent jump to 492 reflects the discovery of new vulnerabilities in the underlying base image packages (OS-level), a reminder that security monitoring must be perpetual even for mature products.

5.9 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.

5.9.1 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)

5.9.2 PostgreSQL Latest Image Digests

The following table documents the exact Docker Hub manifest digests used for vulnerability analysis in this report (May 2026):

Table 6: Docker Hub Manifest Digests for Supported PostgreSQL Images

Tag	Manifest Digest (SHA256)	Architecture
postgres:18.4	775a0820...de3a3a	linux/amd64
postgres:17.10	2293160e...612af7	linux/amd64
postgres:16.14	56e4e0c3...47756b	linux/amd64
postgres:15.18	2e2c9831...937b6	linux/amd64
postgres:14.23	16b755b5...345aa48	linux/amd64

### 5.9.3 Docker Hub Image URIs

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

- **18.4:** [hub.docker.com/.../sha256-775a082...](https://hub.docker.com/.../sha256-775a082...)

#### 5.9.4 Scanning with Digest References

To scan a specific image build regardless of tag updates:

```
trivy image postgres@sha256:775a0820dd34e907ec63277e388d145056d427548e461d7953a070edb5de3a3a
```

Or with both tag and digest for documentation:

```
trivy image postgres:18.4@sha256:775a0820dd34e907ec63277e388d145056d427548e461d7953a070edb50
```

## Important: Image Digest Best Practices

## Why Digests Matter:

- Tags can be overwritten - `postgres:18` may point to different images over time
- Digests are immutable - `sha256:775a082...` 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

### 5.9.5 Verifying Image Digests

To verify the current digest of a tagged image:

```
docker inspect postgres:18.4 --format='{{.RepoDigests}}'
```

Or using `skopeco` (which we used for this report):

```
skopeo inspect docker://docker.io/library/postgres:18.4 | jq -r '.Digest'
```

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 18.4		
Metric	Before (v12)	After (v18.4)
Critical CVEs	9	3
High CVEs	70	32
Medium CVEs	100	93
Low CVEs	124	116
Total Vulnerabilities	305	244
Risk Score	764 (High)	492 (High)
Vulnerability Reduction	-	20%
Risk Score Reduction	-	35%
Support Status	EOL'd (Nov 2024)	Supported until Nov 2030

#### Business Case for Migration

Upgrading from PostgreSQL 12 to 18.4 reduces the risk score from **764 to 492** and provides **4+ years of continued support**. While current base image vulnerabilities affect all versions, supported versions receive regular security patches that eventually resolve these flaws.

### 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](#)