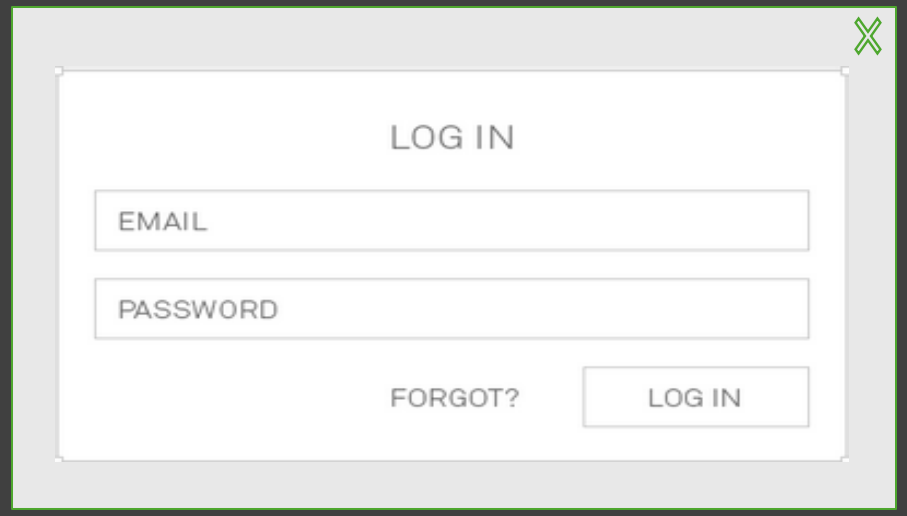
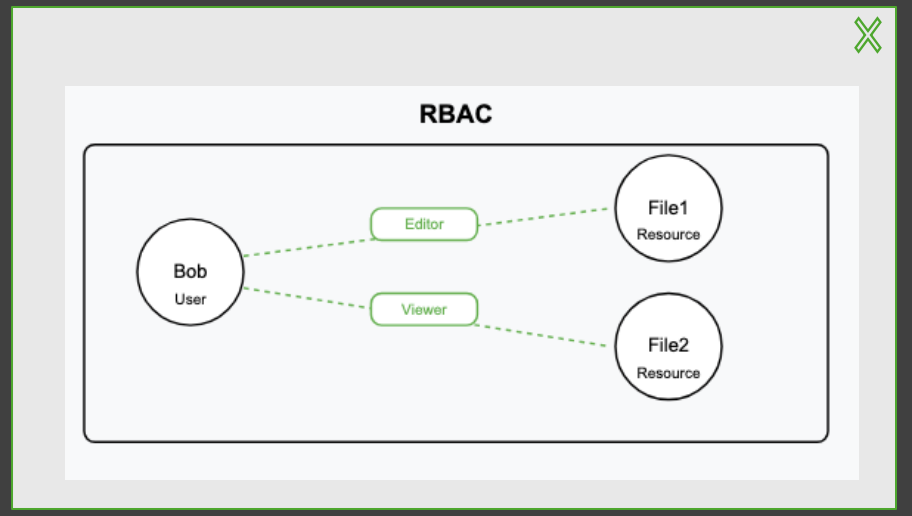
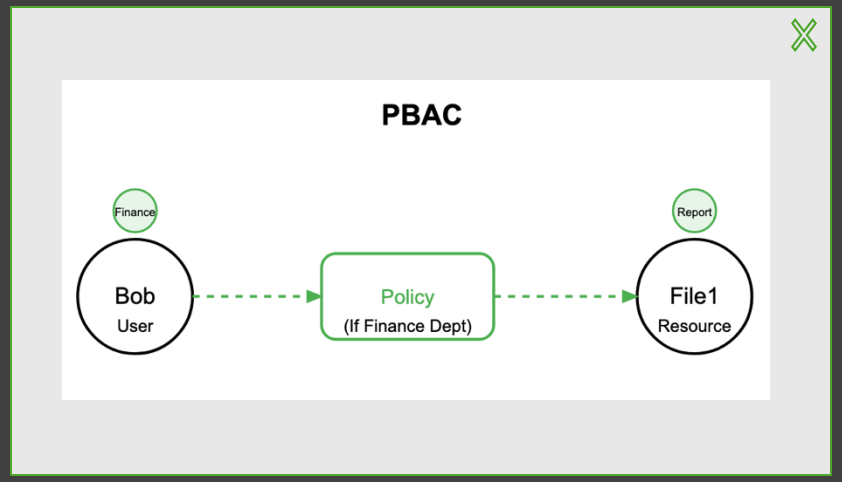
Protecting PII & AI Workloads in PostgreSQL

DEV RAJ GAUTAM

Secured Apps



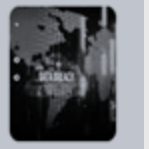




Unsecured Database

BleepingComputer

Insecure Database Leads to Over 800 Million Records Data Breach



An unprotected 140+ GB MongoDB database led to the discovery of a huge collection of 808539939 email records, with many of them also...

Mar 8, 2019

The Hacker News

Microsoft Confirms Server Misconfiguration Led to 65,000+ Companies' Data Leak

Microsoft this week confirmed that it inadvertently exposed information related to thousands of customers following a security lapse.

Oct 21, 2022

Help Net Security

A PostgreSQL zero-day was also exploited in US Treasury hack (CVE-2025-1094)

The suspected Chinese state-sponsored hackers who breached workstations of several US Treasury employees in December 2024 did so by leveraging not one, but two...

Feb 17, 2025

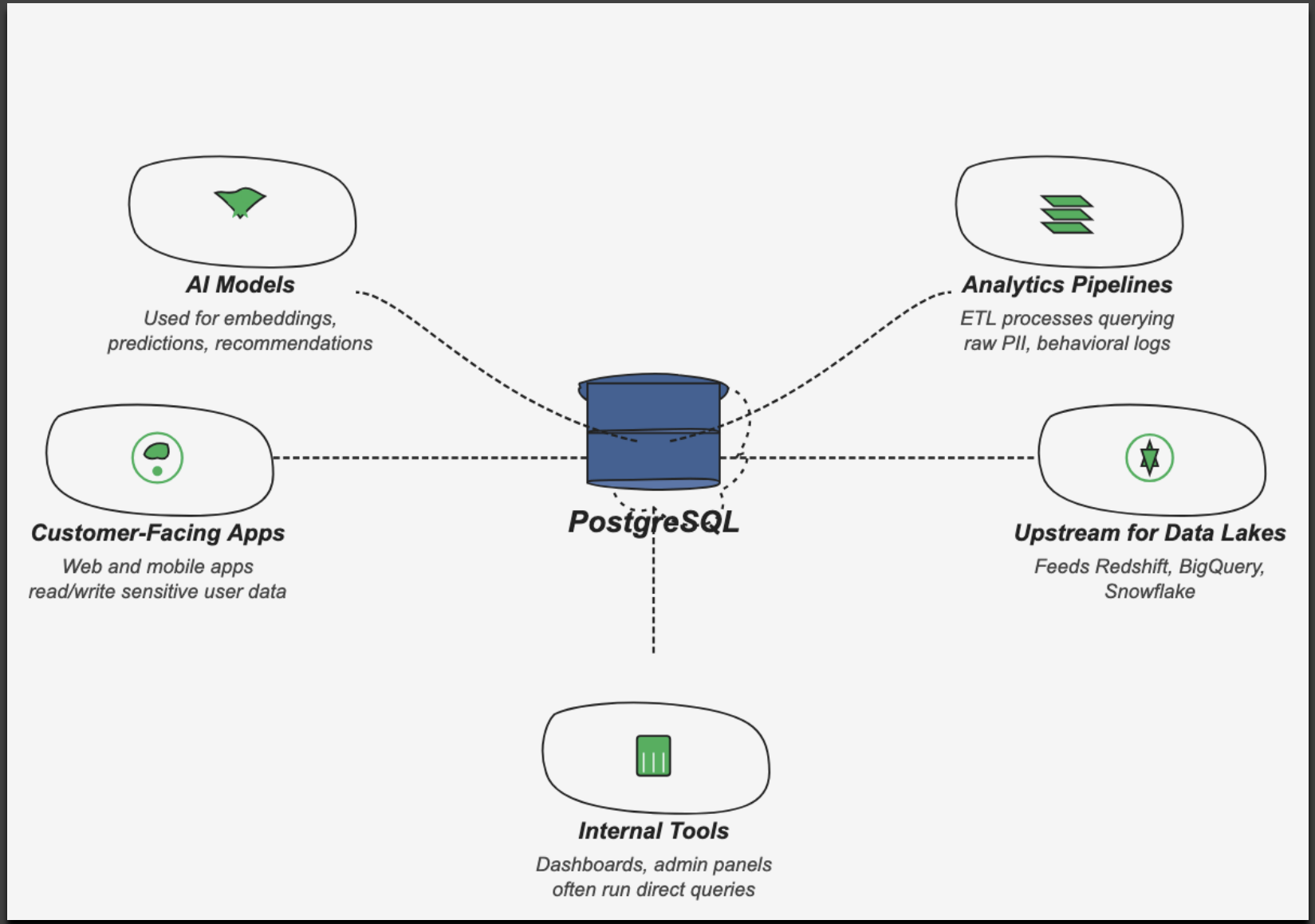
GBHackers News

PostgreSQL Vulnerability Allows Hackers To Execute Arbitrary SQL Functions

A critical vulnerability identified as CVE-2024-7348 has been discovered in PostgreSQL, enabling attackers to execute arbitrary SQL functions.

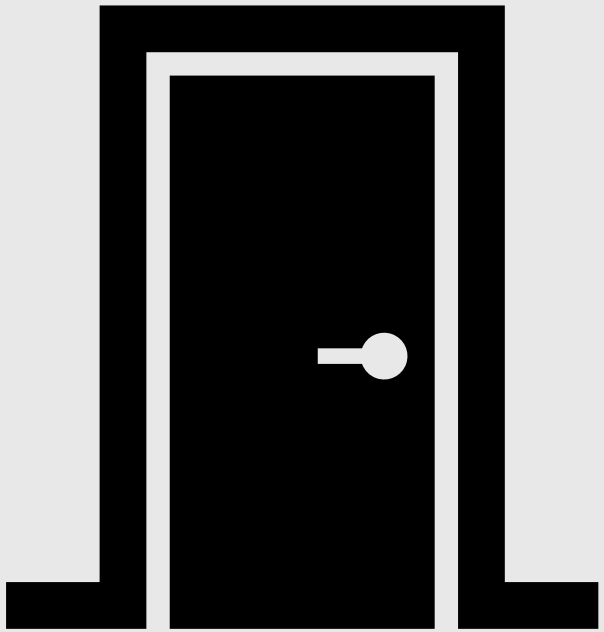
Aug 13, 2024

The Unsecured Core of the Modern Stack

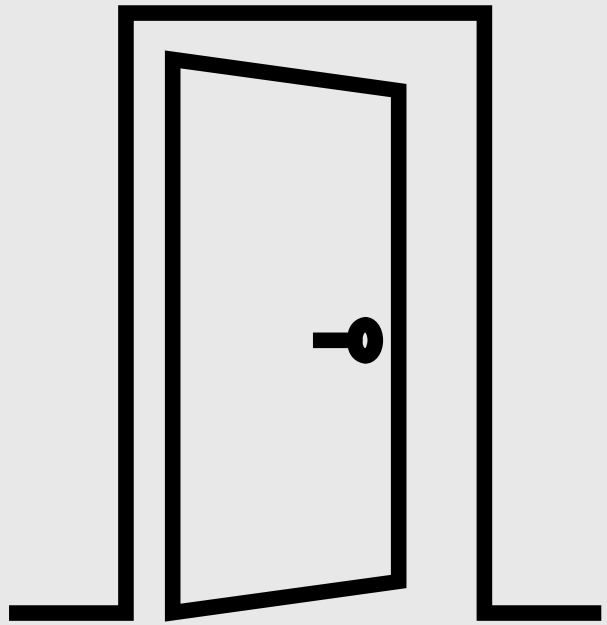


Securing Database & Data is Equally Important

Front Door



Back Door



Agenda

- Introduction: Why PII & AI Security Matters
- Understanding PII in PostgreSQL
- Introduction: Why PII & AI Security Matters
- PostgreSQL Security Capabilities
- Anonymization & Data Masking Techniques
- Handling AI Workloads Securely
- Compliance Mapping & Audit Trails
- Q&A and Wrap-Up

About Me

15+ Years of Industry Experience
Developer, Team Lead, Researcher

Experience in Leading and Managing Projects (10+)
Registered Scrum Master

MBA , MSCIT

CQI IRCA Certified Management System Auditor
Information Security
Personal Information
Artificial Intelligence *PECB

Row Level Security

- Scenario
  - An queries AI training data in user_profiles. They return names, emails, and embedding vectors for all users.
- Problem
  - Data Leak
- Solution
  - RLS by user, roles and other objects.
  - Enable RLS on user_profiles WHERE user_id = current_user

Column-Level Protection

- Scenario
  - A shared PostgreSQL database holds email, phone, and national ID, which are used in an AI model and queried by analysts across teams.
- Problem
  - AI pipelines process raw PII
  - Analysts can query or export sensitive data
  - If the DB is hacked or backed up improperly — all PII is leaked
- Solution
  - Use pgcrypto for column-level encryption
    pgp_sym_encrypt(national_id, key)
  - Decrypt only when needed — with access control + audit

Limit By Role

- Scenario
  - You have multiple teams using the same PostgreSQL instance Data Scientists, Business Analysts, Backend Developers. All roles connect to the same database; many have broad read access.
- Problem
  - Developers can see user emails
  - Analysts can access salary or health info
  - AI pipelines run with superuser privileges
- Solution
  - Create granular roles: analyst_read, dev_basic, ml_pipeline
  - Grant access only to needed tables/columns
  - Use SET ROLE and session auditing for traceability

Anonymize Before you Analyze

- Scenario
  - Marketing team requests user data to analyze trends. Data includes age, zip code, and purchase history tied to names.
- Problem
  - Analysts don’t need real names or emails
  - Combining zip, age, and gender can re-identify users
  - Risk of exposing PII during model training or sharing
- Solution
  - Apply pseudonymization (user_id → token)
  - Apply anonymization (name → NULL, zip → region)
  - Use views or transformation scripts before exporting

Know Who Touched It

- Scenario
  - An AI pipeline fails and needs retraining. A developer reruns a data export script on the customer_insights table, which includes purchase history and emails. Later, a CSV of raw customer PII is found on the dark web; no one knows who exported it.
- Problem
  - No access trail for data exports
  - No visibility into how sensitive data was used
- Solution
  - Enable pgAudit to log read/write access to PII tables
  - Use native logs to track export scripts, session activity
  - Correlate logs with roles and users for full traceability

Securing Vector Search with pgvector

- Scenario
  - You're using pgvector to store embeddings from customer support chats. AI models query nearest neighbors to generate auto-replies. The vector data still links back to real users and can be reverse-engineered.
- Problem
  - Vectors may encode PII (names, context)
  - No constraints on how embeddings are queried
  - Potential for model inversion or unauthorized inference
- Solution
  - Restrict access to vector tables using RLS or views
  - Log and audit AI model access to embedding queries
  - Limit query results (e.g., top 3 only) to reduce leakage risk
  - Mask metadata linking vectors to user identity

Protect Data in Transit with SSL/TLS

- Scenario
  - An AI dashboard queries user data from PostgreSQL over a public cloud network. Traffic includes login credentials and customer records. The connection is unencrypted and vulnerable to sniffing or man-in-the-middle attacks.
- Problem
  - Login credentials exposed
  - PII (e.g., names, IDs) can be intercepted
- Solution
  - Enforce SSL (ssl = on in postgresql.conf)
  - Use sslmode = require or verify-full in app connections
  - Implement client certificates for stricter validation

Data Classification First

Level	Example Columns	Controls
High	national_id, card_no	RLS **AND** pgcrypto, pgAudit, encrypted backups
Medium	email, age	RLS **OR** masked views, audit reads
Low	aggregates, k-anonymised ages	None beyond RBAC

Encrypted Backups & DR

- Backups often outlive production data—#1 source of breach after prod
- pg_basebackup –K or --waldir + server-side GPG (gpg --symmetric).
- Store keys in KMS (AWS KMS, Hashicorp Vault).
- Disaster-Recovery drill: restore + re-key to prove encryption isn’t ornamental.
- Automate integrity check: pg_verifybackup.

Performance & Cost Tips

- RLS: add predicate indexes → CREATE INDEX … WHERE tenant_id IS NOT NULL;
- pgcrypto adds CPU; benchmark with pg_stat_kcache.
- pgAudit: direct logs to csvlog; ship to SIEM—avoid bloating pg_log.
- pgvector: use HNSW index (Postgres 16) + max_connections tuning to keep latency < 50 ms.

Thank You