POSETTE:
An Event for Postgres 2026

From Dev to Prod: Securing Postgres the Right Way

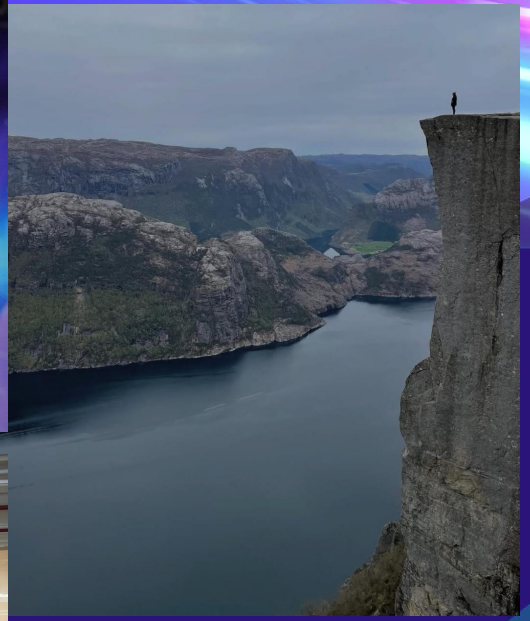
Sakshi Nasha

Hosted by

Microsoft | June 16-18, 2026 • posetteconf.com

$whoami

👩‍💻 Software Engineer @Cohesity
⭐ Open Source Contributor
🗣️ 25+ International talks on Go, APIs, Security, PostgreSQL and open-source technologies
🏸 🧗‍♀️ ✈️



Why This Talk Matters

- Many teams **focus on performance and shipping features first**, while security remains **default-driven until incidents happen**.
- This includes:
  - Misconfigured privileges
  - Overexposed schemas
  - Lack of auditing

Goal is to Shift security from reactive to proactive

Common PostgreSQL Security Blind Spots

Defaults that work in development can create risk in production.

01	**Public Schema Access**	• Default public schema was overly permissive (<PG15) • Any user could create objects unless explicitly restricted • Increased risk of privilege misuse
02	**Overprivileged Roles**	• Excessive superuser usage • Shared admin credentials • Weak permission boundaries
03	**Missing TLS**	• Unencrypted client-server traffic • Credential exposure risk • Sensitive data vulnerable in transit
04	**No Query Auditing**	• Limited native visibility • Difficult forensic analysis

Recent Security Lesson: CVE-2025-1094

What Happened

PG escaping functions (PQescapeString) were trusted to sanitize user input. **Under encoding edge cases,** escaping protections failed, creating SQL injection.

Mistakes Made

Reliance on escaping APIs, manual SQL construction & client-side safety assumptions instead of prepared statements, parameterized queries & strict input validation.

Impact

Affected scripts, automation workflows, psql command construction, & legacy applications -> relying on manually built SQL

Lesson

Production security depends on layered defenses, not assumptions.

1. Least Privilege

- Separate application, migration, and admin roles : Limit blast radius
- Revoke unnecessary default privileges early
- Define clear permission boundaries
- Avoid superuser for applications

```
REVOKE ALL ON SCHEMA public FROM PUBLIC;
CREATE ROLE app_read;
CREATE ROLE app_write;
GRANT CONNECT ON DATABASE mydb TO app_read, app_write;
```

Every service should have only the permissions it absolutely needs.

2. Schema Isolation

Why It Matters

- Shared `public` schema increases exposure
- Weak namespace boundaries expand attack surface
- Default object creation permissions create unnecessary risk

Production-Grade Strategy

- Multi-schema design (`app`, `admin`, `audit`)
- Locked-down `public` schema
- Restrict `CREATE` privileges
- Separate operational boundaries by function

```
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
CREATE SCHEMA app;
CREATE SCHEMA admin;
CREATE SCHEMA audit;
```

“As privilege models mature, namespace control becomes your next security layer.”

3. Search Path Security & SECURITY DEFINER

Risks

- Weak search_path: redirect function execution
- Public schema abuse
- Privilege escalation

```
-- Insecure setup: public schema is writable, weak search_path
CREATE FUNCTION public.calculate_bonus(integer)
RETURNS integer
LANGUAGE sql
AS 'SELECT $1 * 2';

CREATE FUNCTION admin.grant_bonus(integer)
RETURNS integer
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
BEGIN
    RETURN calculate_bonus($1); -- Unqualified function call
END;
$$;
```

3. Search Path Security & SECURITY DEFINER

Production-Grade Practices

- Harden the search_path
- Qualified object names
- Controlled DEFINER functions

```
-- Attacker creates malicious function in public schema
CREATE FUNCTION public.calculate_bonus(integer)
RETURNS integer
LANGUAGE sql
AS 'SELECT 999999';

-- If search_path includes public first:
-- grant_bonus() may execute attacker-controlled function

-- Production hardening:
ALTER FUNCTION admin.grant_bonus(integer)
SET search_path = admin, pg_catalog;

REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE EXECUTE ON FUNCTION admin.grant_bonus(integer) FROM PUBLIC;
```

“Controlled execution paths are essential to secure privilege boundaries”

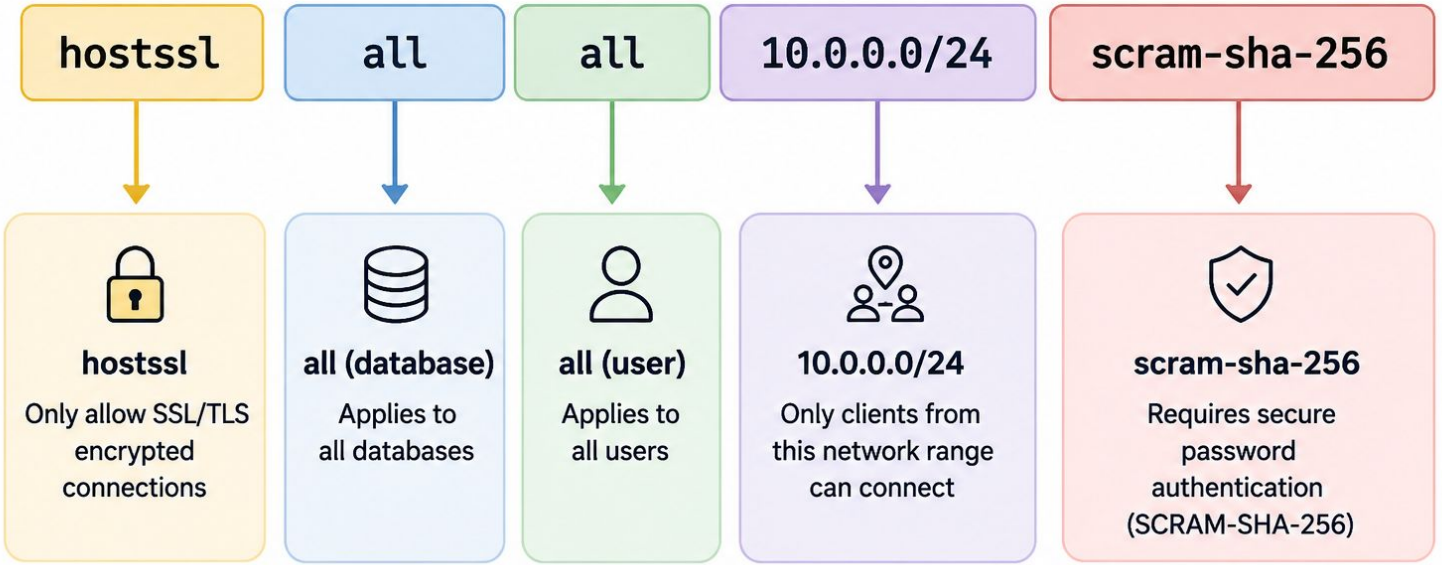
Secure data Beyond Permissions

4. Protecting Data at Rest & In Transit

- Data at Rest : Disk encryption, Encrypted backups, Short-lived credentials
- Data in Transit : Enforce TLS/SSL and Prevent credential exposure
- Avoid legacy authentication methods, use SCRAM-SHA-256
- Harden pg_hba.conf, and restrict trusted networks

```
hostssl all all 10.0.0.0/24 scram-sha-256
```

pg_hba.conf Rule Explained



In Simple Terms:

"Allow only encrypted connections from trusted internal network users using strong authentication."

Security Benefits:

- Encrypted traffic (TLS)
- Restricted source IPs
- Secure authentication method
- Reduced exposure surface

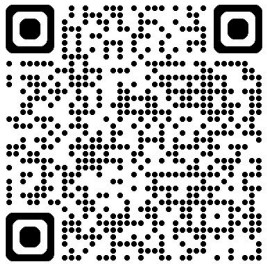
Recent PostgreSQL Security Enhancements (PG17 & PG18)

PostgreSQL 17

- Safer search_path handling in maintenance wokflows
- Direct TLS negotiation (sslnegotiation)
- pg_maintain role reduces superuser usage
- Native incremental backup support(pg_basebackup --incremental)

PostgreSQL 18

- Integration with external identity provider: OAuth Support
- Stronger enterprise authentication options

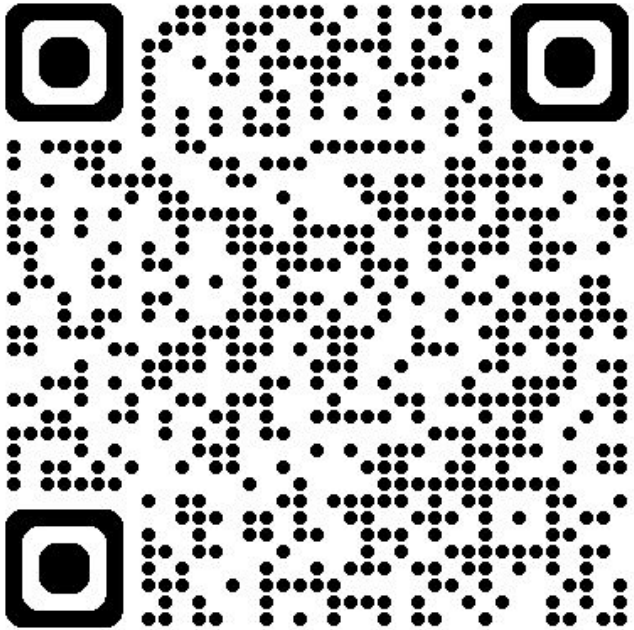


Production Security Checklist

1. **Identify the security blind spots** & revoke unsafe defaults
2. Define clear permission boundaries : **Enforce least privilege**
3. Namespace control via **Schema Isolation**
4. **Controlled Execution Path** :
   a. Harden search_path
   b. Secure SECURITY DEFINER usage
5. **Secure data beyond permissions** :
   @transit : Enforce TLS + SCRAM
   @rest : Encrypt storage and backups
   Harden pg_hba.conf
6. Stay current with PG releases

Microsoft

Connect with me 🙋🏻‍♀️



Microsoft

Thank you to the entire POSETTE Team

Microsoft