Building OAuth2 Validator Extension for Postgres 18

Ajit Awekar,
Senior Staff SDE
Date: 13/3/2026

Agenda

1. OAuth 2.0 Core Concepts
2. Configuration & Prerequisites
3. Server Architecture & Validator Framework
4. Validator Implementation Strategy
5. Security Best Practices

From Password to Access Tokens

Password

MD5 (Legacy)
SCRAM-SHA-256
(Modern)

Enterprise

LDAP
Kerberos

OAuth2

Access Tokens

The Access Token Concept

Core Idea

Replace permanent passwords with temporary "Access Tokens" (digital ID badges) issued by a trusted Identity Provider (IdP) like Google, Okta, or Keycloak.

How it Works:

- **1. Get Badge:** Client asks IdP for token.
- **2. Show Badge:** Client sends token to DB.
- **3. Check Badge:** DB verifies token validity.
- **4. Login:** DB maps identity to role.

OAuth 2.0 Device Authorization Flow

Simple, User-Friendly Login for Devices Without Browsers

1. Client Requests Codes

App asks IdP for a device code.

2. User Authenticates

User opens URL on phone/laptop, enters device code, and signs in.

3. Client Polls

App periodically checks IdP status to see if authentication is finished.

4. IdP Issues Token

Once auth is verified, the IdP returns an access_token to the app.

5. Access Granted

Client presents the token to PostgreSQL (or resource server) for access.

```
edb@localhost:~$ psql "host=127.0.0.1 user=edb dbname=edb port=5444 oauth_issue
r=https://integrator-3869937.okta.com/oauth2/default oauth_client_id=0oaw8wpqeu
wi9EQOH697"

Visit https://integrator-3869937.okta.com/activate and enter the code: JNNNGDDD
psql (19.0.0)
Type "help" for help.

edb=# select current_user;
 current_user
--------------
 edb
(1 row)

edb=# select system_user;
                system_user
-----------------------------------
 oauth:ajit.awekar@enterprisedb.com
(1 row)
```

Key Terminology

Term	Category	Definition
Identity Provider (IdP)	Security Service	External system (e.g., Okta, Google) that issues the Access Token.
Resource Server	Database	The Postgre Server itself that holds the data.
oauth_issuer	Client Param	URL identifying the trusted Provider (required for connection).
JWKS	Security Key	JSON Web Key Set: Cryptographic keys used to verify token signatures.
Discovery URL	IdP Config	Metadata endpoint to automatically fetch JWKS and scopes.

Advantages of OAuth 2.0

Modernizing security architecture through delegated access and token-based authentication.

Cures Password Phobia

Eliminates password fatigue by vastly reducing the need for users to create, memorize, and constantly manage multiple complex passwords across diverse applications.

Seamless SSO & MFA

Leverages centralized Identity Providers to seamlessly enable Single Sign-On (SSO) layered with advanced Multi-Factor Authentication (MFA) for frictionless, secure access.

Zero Password Risk

Applications authenticate using secure, scoped, and time-bound access tokens instead of actual user credentials, structurally preventing password interception and theft.

Instant Access Control

Users can be seamlessly added or removed instantaneously. Permissions and access rights can be granted, modified, or completely revoked in real-time from a central location.

Configuration & Prerequisites

Server Requirements

- **PostgreSQL 18+:** Native SASL OAUTHBEARER support.
- **Validator Module:** Pluggable extension to verify tokens.
- **Configuration:** pg_hba.conf set to method=oauth.

Client Requirements (libpq)

- **Build Flags:** Must be built with --with-openssl and --with-libcurl.
- **Library:** libpq-oauth-18.so (or version matching PG) for handling HTTP requests.

Server Configuration

pg_hba.conf

Enables OAuth feature for matching connections.

```
host all all ::1/128 oauth \
scope="openid profile postgres" \
issuer="https://myissuer/oauth2 \
map="oauth_map"
```

postgresql.conf

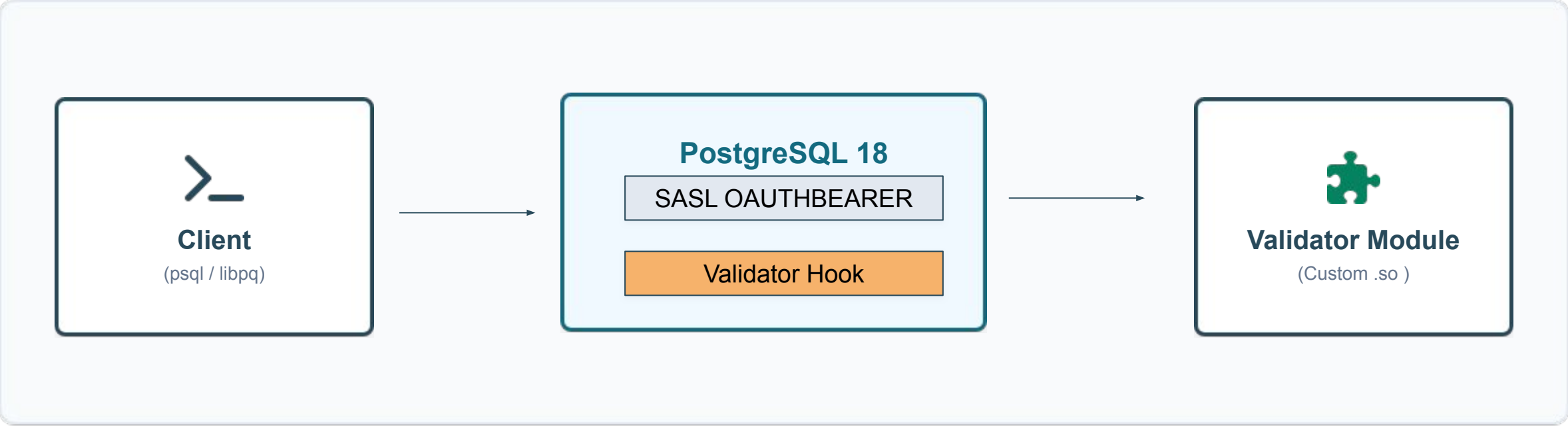
```
oauth_validator_libraries = '$libdir/okta_validator
```

Maps the Token Subject (sub) to a DB Role.

```
# MAPNAME     SYSTEM-USERNAME   PG-USERNAME
oauth_map     ^user@abc.com$    edb
```

ARCHITECTURE OVERVIEW

Server-Side SASL: PostgreSQL 18 implements the OAUTHBEARER SASL mechanism. It relies on an external Validator Framework.



Why PostgreSQL Needs an OAuth Validator extension

Inconsistent Formats

Access tokens differ significantly (e.g., opaque strings vs. signed JWTs). Native PostgreSQL lacks specialized parsing engines.

IdP-Specific Logic

Validation rules are rarely generic. They require handling unique endpoints and logic defined by each Identity Provider.

Cryptographic Complexity

Validators must dynamically switch between algorithms (HS256 vs. RS256) and perform complex PKI operations.

Key Rotation

Validators allow automatic fetching of new keys from JWKS endpoints to handle rotation without downtime.

Validator Module Responsibilities

1. Validate

Ensure the token is cryptographically sound, not expired, and issued by a trusted party (check signature & issuer).

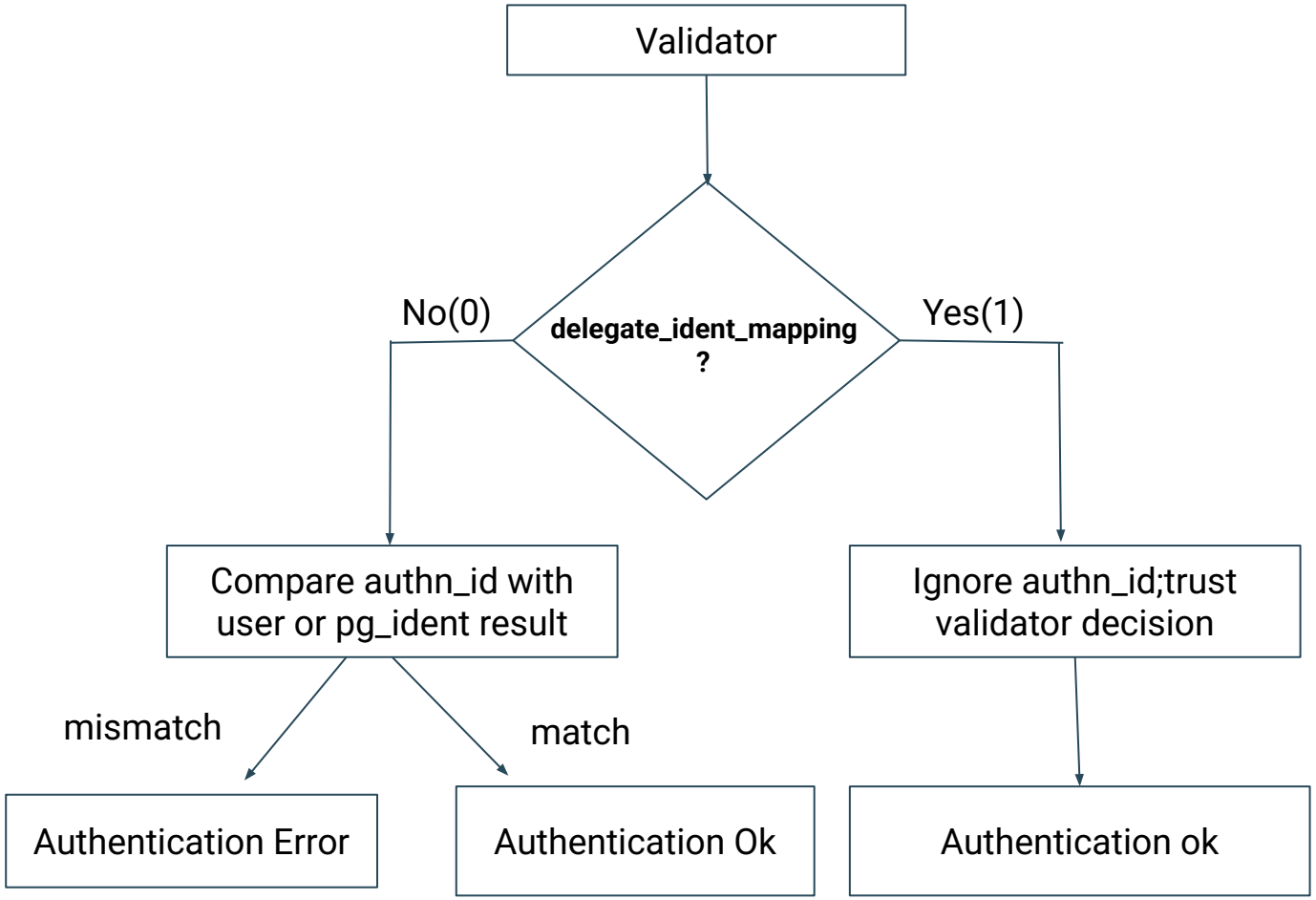
2. Authorize

Check the token's scopes to ensure the user has granted the specific permissions required to access the database.

3. Authenticate

Extract the unique user identifier (e.g., email or 'sub' claim) to map it to a PostgreSQL internal role.

Validator-User Mapping



Validation Strategies

Online Validation

Mechanism: Token Introspection.

Makes a network call to the IdP for every login attempt to verify the token.

- **Pros:** Immediate, central revocation. Simpler logic.
- **Cons:** Slower latency; adds runtime dependency on IdP.

Offline Validation

Mechanism: Cryptographic Signature.

Validator downloads public keys (JWKS) once and verifies token signatures locally.

- **Pros:** Fast (no per-login network call). Robust.
- **Cons:** Tokens cannot be revoked until they expire. Complex key caching.

Validator Interface: Entry Point

Custom C modules must implement specific hooks. The entry point typically sets up the environment and returns the callback structure.

Key Components:

- **_PG_oauth_validator_module_init:** The required entry function name.
- **OAuthValidatorCallbacks:** Struct containing pointers to your logic.
- **Lifetime:** The returned pointer must have server lifetime (static).

```c
static const OAuthValidatorCallbacks validator_callbacks = {
    PG_OAUTH_VALIDATOR_MAGIC,

    .startup_cb = validator_startup,
    .shutdown_cb = validator_shutdown,
    .validate_cb = validate_token
};


/*
 * Validator module entry point.
 */
const OAuthValidatorCallbacks *
_PG_oauth_validator_module_init(void)
{
    return &validator_callbacks;
}
```

Core Logic: Validate Callback

Inputs (From Postgres)

- **token:** Raw Bearer Token string from client.
- **role:** Requested DB role.
- **state:** Module internal state.

Outputs (To Postgres)

- **result->authorized:** Set 'true' if valid.
- **result->authn_id:** The authenticated identity string (e.g., user@email.com).

```
/*
 * Validator implementation.
*/
static bool
validate_token(const ValidatorModuleState *state,
               const char *token, const char *role,
               ValidatorModuleResult *res)
{

    if (verify_signature(signature))
    {
        res->authorized = true;
        //Decode token
        res->authn_id = extract_claim(token,"sub");
        return true;
    }
    else
        return false;
}
```

EDB POSTGRES AI

JWT Decoding: Base64URL to JSON

> **Note:** Header and Payload are deliberately non-encrypted. Decoding is necessary to read claims, but is **NOT** validation.

The Decoding Steps

1. **Split the Token:** Separate parts using the dot (.).
2. **Apply Base64URL Conversion:** Decode the Header and Payload (handling +/- characters).
3. **Parse as JSON:** Parse the resulting decoded string to access claims.

Segment	Encoded Content	Decoded Content
Part 1	Base64URL-encoded JSON metadata	Header JSON
Part 2	Base64URL-encoded claims	Payload JSON
Part 3	Base64URL-encoded crypto hash	Binary Signature

Token Validation Lifecycle

Signature Validation The first line of defense. We verify the token's integrity using the Authorization Server's public key (JWKS) or a shared secret. This ensures the token has not been tampered with in transit.

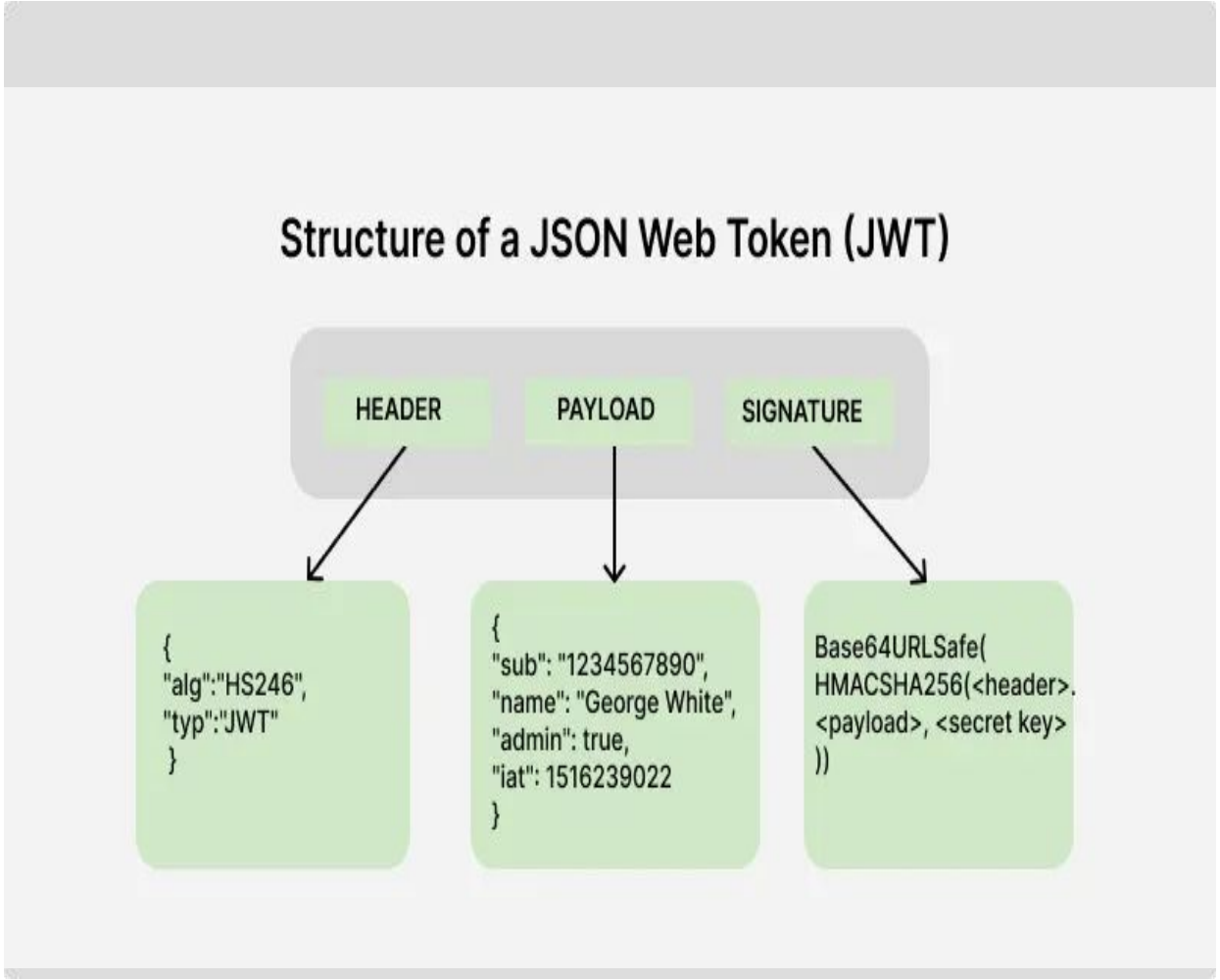
Decoding & Parsing The token structure (Header.Payload.Signature) is split and Base64Url decoded. This transforms the opaque string into a readable JSON object, exposing the claims required for authorization decisions.

Claim Comparison Validating business logic. We assert that the token is not expired (exp), is from a trusted issuer (iss), is intended for this audience (aud), and contains the required scopes.

JWT Structure

Structure: Three parts separated by dots (.):

- **Header:** Algo & Token Type.
- **Payload:** Data Claims (User ID, Exp, Scope).
- **Signature:** Cryptographic proof of integrity.



Prepare for Validation

Step 1: Header Decoding and Key Discovery

1 Decode Header

Perform a Base64URL decode on the Header string (Part 1). This reveals the algorithm and key identifier used by the IdP.

```
{
  "alg": "RS256",
  "kid": "key-id-01",
  "typ": "JWT"
}
```

2 Identify Params

Extract "alg" and "kid" from the JSON to determine the verification logic and locate the specific public key required.

3 Retrieve Key

Fetch the matching Public Key from the Authorization Server's JWKS endpoint based on the "kid" provided in the header.

Validate Signature

Step 2: Sequential Cryptographic Verification

1 Create Signing Input

Concatenate the encoded Header and encoded Payload strings, joined by a dot, to form the input for the hash.

```
Input = Header + "." +
Payload
```

2 Recalculate Signature Only applicable for symmetric validation

Apply the algorithm specified in the header to the input using the retrieved cryptographic key.

$$Sig_{calc} = AlgorithmInputK_{crypto}$$

3 Compare Signatures

Verify that the recalculated signature matches the original signature (Part 3) provided in the JWT token.

✓ MATCH: Token Authentic

⚠ MISMATCH: Reject Token

EDB POSTGRES AI

Optimizing Offline OAuth Validation

Enhancing Performance and Scalability with JWKS Caching

THE CHALLENGE

Latency & Performance

The Resource Server (RS) must validate JWTs using the IdP's public key. Fetching this from the JWKS endpoint for **every request** introduces significant network latency.

Inefficiency & Load

Repeated external calls for static data burden the Identity Provider's servers and saturate the network, creating a potential bottleneck for high-throughput APIs.

Key Rotation Complexity

Public keys change periodically. Without a strategy, static validation fails when keys rotate, but naive caching risks using stale keys.

THE SOLUTION

Fast Local Validation

Public keys are downloaded once and stored in a local cache. Validation occurs against this local copy, eliminating network calls and drastically reducing latency.

Reduced Traffic

Caching significantly lowers traffic to the JWKS endpoint. This decouples the RS from the IdP's immediate availability, improving overall resilience.

Graceful Refresh Strategy

Implement a TTL (e.g., 24h) combined with a "smart fallback": if a token's kid isn't in the cache, trigger an immediate one-time JWKS refresh.

EDB POSTGRES AI

Common JWT Claims

Claim	Full Name	Purpose in Postgres Context
iss	Issuer	Identifies the IdP (e.g., auth.google.com). Must match config.
sub	Subject	Unique User ID. Often mapped to the DB role.
aud	Audience	Intended recipient. Should match your DB Service ID.
exp	Expiration	Timestamp when token becomes invalid. Critical check.
iat	Issued At	Time at which the JWT was issued. Used to check token age.
nbf	Not Before	Time before which the JWT must not be accepted.
scope	Scope	Space-separated list of scopes (permissions). Checked by Validator.
email	Email	User email, often used as an alternative identifier for role mapping.

Okta Token Inspection & Decoding

</> RAW BEARER TOKEN (BASE64)

eyJraWQiOiI3RWNjUndNQU9FQ090emthUUw2a1hmYjctOVZNMERhWEtneEE3Mm
YtalBBIiwiaWFsZyI6IlJTMjU2In0.eyJ2ZXIi0jEsImp0aSI6IkFULnhMV2IzRlM0d3pqUDY
2WVR3MG5JUkM1Z0ZVLXFiYnE0a1NWYl9RUnhlN00iLCJpc3Mi0iJodHRwczovL2lud
GVncmF0b3ItMzg2OTkzNy5va3RhLmNvbS9vYXV0aDIvZGVmYXVsdCIsImF1ZCI6Im
FwaTovL2VwYXMiLCJpYXQi0jE3NjQ2NTM2MzgsImV4cCI6MTc2NDY1NzIzOCwiY2l
kIjoiMG9hdzh3cHFldXdPOUVRUUg20TciLCJ1aWQiOiIwMHV30HdvOG5iNEXYYjZF
RDY5NyIsInNjcCI6WyJvcGVuaWQiLCJwcm9maWxlIiwiZXBhcyJdLCJhdXRoX3RpbW
Ui0jE3NjQ2NTM1NzIsInN1YiI6ImFqaXQuYXdla2FyQGVudGVycHJpc2VkYi5jb20ifQ.P
H5XJQYKKW4lvo43EguZwww.rq0wvS6jnQQz3iSE6wkvZCw62lQIBMtGoKselpwjdHs
08aZq9CJEUifqQ7yP5U74h1vmEKeGtWYYOQewQe2as8tbTyGPkclydvmVuz_Lh_Yi
88HGHCcID3LC5qcukQxWL4J1AfBcmQmmhjtfumbJGu9HvS8v0rkY-dzAfQHT0BgM_
XXkxnXaLbeUOPpRI2T6rIBPGTiqdKMB5sMZLIbf3v9AatuxanRLpQRietCcXSvcaxoBS
qPWPPJCJMTZMnpEFJ6n-Ac0XnUMkybiJZMS7ctG5b18N3k5Sl-rud7DscypU8RaiYN
L5sw9HYZvGkg

DECODED IDENTITY & CLAIMS

aud: api://epas

iss: https://integrator-3869937.okta.com/...

sub: ajit.awekar@enterprisedb.com

uid: 00uw8wo8nb4LXb6ED697

scp: [openid, profile, epas]

iat: 1764653638 (2026-12-02 11:03:58)

exp: 1764657238 (2025-12-02 12:03:58)

jti: AT.xLWb3FS4wzjP66YTwOnIRC...

cid: 0oaw8wpqeuwi9EQQH697

EDB POSTGRES AI

Keycloak Token Inspection & Decoding

</>RAW BEARER TOKEN (BASE64)

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ik1mOGNKX2FkODdrRlZJR1R5VXVhTmY1ajdpWlpjR0pH
eVFqQ3FaMGhUdTQifQ.eyJleHAiOjE3NjQ5Mzk2NzAsImlhdCI6MTc2NDkzOTYxMCwiYXV0aF90aW1lIjoxNz
Y0OTM5NjA5LCJqdGkiOiJvbnJ0ZGc6NWIwZWMyNzAtMGFFMS1mZTM1LWEyNWYtNWQzOGZlM2M2YzFh
IiwiaXNzIjoiaHR0cHM6Ly95b2NhbGhvc3Q6ODQ0My9yZWFsbXMvbWFzdGVyIiwiYXVkIjpbIm15LXJlc291cm
NlLXNlcnZlciIsImFjY291bnQiXSwic3ViIjoiNWRiNGRjMmQtMjA0Ni00NDZjLWE4YTctNDhiNDhlMTQ4MTg2Iiwi
dHlwIjoiQmVhcmVyIiwiYXpwIjoidGVzdGNsaWVudCIsInNpZCI6IjgzMDA3YWI5LTg5ZjUtNDI1Mi1iMGUwLTVk
ZGI5ZTBiOGM3OCIsImFjciI6IjEiLCJhbGxvd2VkLW9yaWdpbnMiOlsiLyoiXSwicmVhbG1fYWNjZXNzIjp7InJvb
GVzIjpbImRlZmF1bHQtcm9sZXMtbWFzdGVyIiwib2ZmbGluZV9hY2Nlc3MiLCJ1bWFFYXV0aG9yaXphdGlvbiJ
dfSwicmVzb3VyY2VfYWNjZXNzIjp7Im15LXJlc291cmNlLWNlcnZlci16Ijp7InJvbGVzIjpbImVwYXMiXX0sInRlc3
RjbGllbnQiOnsicm9sZXMiOlsic2ltcGxlIl19LCJhY2NvdW50Ijp7InJvbGVzIjpbIm1hbmFnZS1hY2NvdW50IiwibW
FuYWdlLWFjY291bnQtbGlua3MiLCJ2aWV3LXByb2ZpbGUiXX19LCJzY29wZSI6InBvc3RncmVzIHByb2ZpbG
UgZW1haWwiLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwicG9lc3RncmVzIjoiaSd0cnVlJyIsInByZWZlcm5lZF91c2
VybmFtZSI6ImVkYiJ9.GmixYU-J-AVGQV2g5eRgm1gZToGkf6-qqzM-hBaZ3j7E0pARL2_122Asu9ErHe5PbO
wjf7RB0GPb27gRADZOUGEG--MLbViLtZ3ZT6WNJUIvjl9W9Ks86dkiQNp1LXXO3851BZDFZFWMN 3M
USDYjODSKkox4pQF8jrm8Frin9cUIT
uI3CrzQf4pHrVplgewoSG55xYF3nEMcWzXJ68R0_8SiyD72MhSFrJRCv3Q73YWB5tfEbEZUpRU5ppMS_qX7
svUVG5iyfdma7VG9BCSgXFHuqnx

DECODED IDENTITY & CLAIMS

aud: ['my-resource-server', 'account']

iss: https://localhost:8443/realms/master

sub: 5db4dc2d-2046-446c-a8a7...

preferred_username: edb

scope: postgres profile email

iat: 1764939610

exp: 1764939670

email_verified: True

azp: testclient

auth_time: 1764939609

Safety & Coding Guidelines

No Token Logging

Modules must not log raw tokens or parts of tokens to the server log. This prevents attackers from retrieving valid tokens from disk logs.

Error Reporting

Log verification problems at COMMERROR level and return normally. Do NOT use ERROR or FATAL during authentication to avoid leaking internal info to unauthenticated clients.

Interrupts

Modules must be interruptible. Use CHECK_FOR_INTERRUPTS() in loops and non-blocking socket calls to handle timeouts and shutdowns gracefully. Instead of a standard read() or connect(), developers must use Postgres-specific functions like WaitLatchOrSocket()

Key Internal functions

load_validator_library

Dynamic loader that initializes the external shared library (.so) defined in postgresql.conf.

validate

The primary enforcement hook. C-level callback that receives raw Bearer token and executes business logic.

oauth_exchange

SASL state machine handler. Orchestrates OAUTHBEARER handshake and parses initial client response.

check_oauth_validator

Integrity assurance. Verifies the loaded library correctly implements the PgOAuth Validator struct and essential callbacks.

Library Ecosystem Overview

Language	Primary Library	Package Manager	Verification Support
Python	PyJWT	pip	Native
Node.js	jsonwebtoken	npm / yarn	Native
Go	golang-jwt/jwt	go get	Native
Java	jjwt (Java JWT)	Maven/Gradle	Native

References & Further Reading

Development Journey

Insights into strategic decisions and security benefits behind implementing OAuth 2.0 natively in PostgreSQL 18.

Technical Deep Dive

A comprehensive guide to the SASL OAUTHBEARER mechanism, server config, and client-side behavior.

Validator Modules

Official documentation on creating and implementing custom server-side validator modules for token verification.

Thank you !

Have questions about the talk? Come meet me at the EDB booth.

LinkedIn: linkedin.com/in/ajit-awekar-6780648