CYBERTEC

POSTGRESQL SERVICES & SUPPORT

PostgreSQL Security von A-Z

DATABASE SECURITY MATTERS

Wie können wir Sicherheit erreichen?

PostgreSQL: Ein geistiges Modell

- listen_addresses: postgresql.conf
- pg_hba.conf Host Based Access Control
- Instance level security
- Database level security
- Schema level security
- Table level security
- Column level security
- Row level security (RLS)

← VEREINFACHTES FEHLERHANDLING

Level 1: Bind addresses

- listen_addresses in postgresql.conf
  - Hört jemand zu?
  - Ist der Port offen?
  - Die erste Hürde beim Aufbau

```
[hs@hansmacbook ~]$ telnet 10.241.35.14 5432
Trying 10.241.35.14...
telnet: connect to address 10.241.35.14: Connection refused
telnet: Unable to connect to remote host
```

Level 2: pg_hba.conf

- Authentication je nach “Herkunft” der IP
  - Kann ohne Neustart geändert werden (SIGHUP ist genug)
  - "trust", "reject", "md5", "password", "scram-sha-256", "gss", "sspi", "ident", "peer", "pam", "ldap", "radius" oder "cert"
  - SSL passiert auf dieser Ebene

```
local   all             all                                     trust
host    all             all             127.0.0.1/32            trust
host    all             all             ::1/128                 trust
host    all             all             192.168.0.0/16
scram-sha-256
host    all             all             10.1.2.0/32             ldap ....
```

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

Level 3: Instance level

- Benutzer, Tablespaces, Datenbanken sind auf Instanz Ebene

```
Command:     CREATE USER
Description: define a new database role
Syntax:
CREATE USER name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
    | VALID UNTIL 'timestamp'
...
```

Level 4: Datenbank Level

- Permissions auf Database Ebene
  - CREATE: Schemata anlegen
  - CONNECT: Verbindungen aufbauen

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP }
        [, ...] | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
```

Level 5: Schema Level

- Permissions auf Schema Ebene
  - CREATE: Tabellen, Functions, Procedures, etc. anlegen
  - USAGE: Verbindungen aufbauen

```
GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
```

Level 6: Table Level

- Permissions auf Tabellen Ebene
  - DELETE vs. TRUNCATE: Row vs. Table locks
  - REFERENCES: Tabellen referenzieren aber nicht lesen
  - TRIGGER: Einen Trigger anlegen

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER
}
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
```

Level 7: Column Level

- Wird selten verwendet
- “SELECT *” ist nicht mehr möglich
- Spaltenrechte sind standardmäßig vorhanden

```
GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
```

Level 8: Row Level Security (RLS)

- Zeilen können “ausgeblendet” werden
- Wie eine “verpflichtende WHERE-Clause”

```
Command:     CREATE POLICY
Description: define a new row-level security policy for a table
Syntax:
CREATE POLICY name ON table_name
    [ AS { PERMISSIVE | RESTRICTIVE } ]
    [ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]
    [ TO { role_name | PUBLIC | CURRENT_ROLE |
        CURRENT_USER | SESSION_USER } [, ...] ]
    [ USING ( using_expression ) ]
    [ WITH CHECK ( check_expression ) ]
```

Level 8: RLS in Aktion

```
ALTER TABLE t_service ENABLE ROW LEVEL SECURITY;

CREATE POLICY bob_pol ON t_service
    FOR SELECT
    TO bob
    USING (service_type = 'open_source');

CREATE POLICY alice_pol ON t_service
    FOR SELECT
    TO alice
    USING (service_type = 'closed_source');
```

Level 8: RLS in Aktion

```
test=# SET ROLE bob;
SET
test=> SELECT * FROM t_service;
 service_type |          service
--------------+--------------------------
 open_source  | PostgreSQL consulting
 open_source  | PostgreSQL training
 open_source  | PostgreSQL 24x7 support
(3 rows)

test=> SET ROLE alice;
SET
test=> SELECT * FROM t_service;
 service_type   |           service
----------------+----------------------------
 closed_source | Oracle tuning
 closed_source | Oracle license management
 closed_source | IBM DB2 training
(3 rows)
```

Reicht das nicht?

Automatisierung hilft

Konsistenz, Vereinfachung

- Oft will man Rechte automatisch vergeben
- Reduziert das Risiko, Dinge zu vergessen

```
Command:     ALTER DEFAULT PRIVILEGES
Description: define default access privileges
Syntax:
ALTER DEFAULT PRIVILEGES
    [ FOR { ROLE | USER } target_role [, ...] ]
    [ IN SCHEMA schema_name [, ...] ]
    abbreviated_grant_or_revoke

where abbreviated_grant_or_revoke is one of:

GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON TABLES
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
…
```

PG_PERMISSIONS: AT A GLANCE

- pg_permissions liefert eine einfache Übersicht
- Vereinfacht die Übersicht und die Rechtevergabe

```
test=# CREATE EXTENSION pg_permissions;
CREATE EXTENSION
test=# SELECT * FROM all_permissions;
-[ RECORD 1 ]-----------------
object_type | TABLE
role_name   | pg_database_owner
schema_name | public
object_name | x1
column_name |
permission  | SELECT
granted     | f
…
```

PLAN / IST VERGLEICH

- pg_permissions liefert eine einfache Übersicht
- Vereinfacht die Übersicht und die Rechtevergabe

```
INSERT INTO public.permission_target
   (role_name, permissions,
    object_type, schema_name)
VALUES
   ('appuser', '{USAGE}',
    'SCHEMA', 'appschema');

SELECT * FROM public.permission_diffs();
```

```
 missing | role_name | object_type | schema_name | object_name | column_name | permission
---------+-----------+-------------+-------------+-------------+-------------+------------
 f       | laurenz   | VIEW        | appschema   | appview     |             | SELECT
 t       | appuser   | TABLE       | appschema   | apptable    |             | DELETE
(2 rows)
```

ES GIBT MEHR ...

- Views: Wer bin ich?
  - Security Invoker
  - Security definer
  - Security barrier
- Functions:
  - Security invoker
  - Security definer
- Transparent Data Encryption (TDE)
- pgcrypto

← EIN KLEINER AUSZUG

ANY QUESTIONS?

Feel free to ask

CEO

Hans–Jürgen SCHÖNIG

MAIL

hs@cybertec.at

PHONE

+43 2622 930 22 - 666

WEB

www.cybertec-postgresql.com

CYBERTEC
DATA SCIENCE & POSTGRESQL