Intro to PostgreSQL Security

PGConf.NYC 2014
New York City, NY

Stephen Frost
sfrost@snowman.net

Stephen Frost

- PostgreSQL
  - Major Contributor, Committer
  - Implemented Roles in 8.3
  - Column-Level Privileges in 8.4
  - Contributions to PL/pgSQL, PostGIS
- Resonate, Inc.
  - Principal Database Engineer
  - Online Digital Media Company
  - We're Hiring! - techjobs@resonateinsights.com

resonate

Do you read...

- planet.postgresql.org

Security in PostgreSQL

- Role system
  - Role-level Privileges
  - Authentication

Security in PostgreSQL

- Authorization
  - Containers
  - GRANT / REVOKE
  - Defaults

Security in PostgreSQL

- Use-cases
  - Web-based
  - Enterprise DB / DW

Roles

- Identities inside PostgreSQL
- Each connection is assiged specific role
- Roles encompass both users and groups
- Nearly all objects are "owned" by a specific role
- Shared across entire cluster (not per-DB)

Roles

- Objects in PG with owners:

```
* Databases                          * Schemas
* Tables (Local and Foreign)         * Functions
* Aggregates                         * Collations
* Conversions                        * Domains
* Event Triggers                     * Foreign Data Wrappers
* Languages                          * Large Objects
* Sequences                          * Foreign Servers
* Tablespaces                        * Types
* Views (Normal and Materialized)
* Operators (and Classes and Families)
* Text Search Configuration and Dictionaries
```

Role Membership

- Roles can be members of other roles
- GRANT used to add a role to another role
- Loops are forbidden
- WITH ADMIN allows the role to grant the role

Role Membership

- inherit / noinherit
  - inherit - privileges (not attributes) automatic
  - noinherit - "SET ROLE ..." required
  - Great for sudo-like DB administration
  - Create "barrier" role- eg: "admin", with noinherit
  - Grant "admin" to, uh, admins, postgres to "admin"
- Supports traditional "User/Group", and then some

Changing Roles

- "SET ROLE" SQL command
  - Allows gaining "noinherit" privileges
  - Can be used to drop privileges too
  - DISCARD ALL; will reset role too
  - "$user" in search_path follows SET ROLE

Changing Roles

- Security Definer Functions run as owner
  - Need to be careful with search_path
  - Strongly recommend against superuser owned
- Views also run as owner
  - Need to mark view 'security_barrier'

Role Privileges

- SUPERUSER
  - Bypass *ALL* security (and some sanity..) checks
  - Use *very* sparingly
  - Never login to SUPERUSER role directly
  - Require "SET ROLE postgres;" to be superuser

Role Privileges

- What's wrong with SUPERUSER?

```
=# delete from pg_database;
DELETE 3
```

Role Privileges

- What's wrong with SUPERUSER?

```
=# delete from pg_class;
DELETE 295
```

Role Privileges

- What's wrong with SUPERUSER?

```
=# COPY pg_class TO '/home/sfrost/pg/src/clean/install/data/postmaster.conf' WITH CSV;
COPY 295
```

Role Privileges

- What's wrong with SUPERUSER?

```
=# COPY pg_class TO PROGRAM 'cat > postgresql.conf';
COPY 295
```

Role Privileges

- CREATEDB
  - Allows creating new databases
  - Give out sparingly- DBs are not free
  - User becomes database owner

Role Privileges

- CREATEROLE
  - Allows creating new roles
  - ALSO allows modifying EXISTING roles
  - Can add CREATEDB to roles, et al
  - Non-superuser can't modify superuser
  - Use with caution

Role Privileges

- REPLICATION
  - User can connect to "replication" database
  - Only grant to dedicated replication accounts
  - Can read every file in the cluster

Role Privileges

- LOGIN
  - Role is allowed to connect to PG
  - Roles with LOGIN will show up in "pg_user"
  - Roles with NOLOGIN will show up in "pg_group"

Role Privileges

- CONNECTION LIMIT
  - Concurrent connection limit
  - Changing this will impact existing connections

Role Privileges

- VALID UNTIL
  - Can't connect after this time
  - Does not impact existing connections

Authentication

- Connection parameters
  - Database
  - PostgreSQL Role
  - Client IP / Unix Socket
  - SSL vs. non-SSL

Authentication

- Based on parameters, auth method is chosen
- Auth method can provide "system" username
- System username can be mapped to PG role

pg_hba.conf

- Processed top-to-bottom, first match wins
- "User" can be "+role" to mean "member of role"
- Database can be "all", "replication", "sameuser"

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local   all             all                                     peer map=unixmap
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication     repl_user                               md5
#host    replication     repl_user       127.0.0.1/32            md5
#host    replication     repl_user       ::1/128                 md5
```

pg_ident.conf

- Also processed top-to-bottom, by map name
- Regexps can be used with "/" and "1"

```
# MAPNAME       SYSTEM-USERNAME         PG-USERNAME
unixmap         root                    postgres
unixmap         /^(.*)$                 \1
localrealm      /^([^@]*)@MYREALM\.COM$ \1
localrealm      jow@OTHERREALM.com      otherjoe
clientcert      "cn=Stephen P. Frost"   sfrost
clientcert      "cn=John Doe"           jdoe
```

Auth Methods

- peer
  - Unix socket based- uses the unix username
  - punts on the authentication issue to the unix layer
  - (ident covers this but also identd, do not use)

Auth Methods

- gss / sspi / krb5 (krb5 deprecated)
  - Kerberos / Active Directory based authentication
  - Perfect for Enterprise deployments
  - Supports cross-realm, princ-based identification
  - SSL required only for data encryption (not authN)
  - No option for Kerberos/GSS data encryption today

Auth Methods

- cert
  - Client-side SSL certificates
  - Useful with OpenSSL support, eg: Smart Cards
  - SSL required for SSL certificates, of course
  - Requires full PKI setup, CAs, etc

Auth Methods

- md5
  - Normal password-based authentication
  - ("password" exists, but PW is sent in the clear)
  - Should use SSL with this

Auth Methods

- radius
  - RADIUS servers- relatively rare / special case
  - Need to use SSL to PG, and RADIUS encryption
- reject
  - Special case- reject if matched

Auth Methods

- ldap
  - Allows for simple-bind, or LDAP lookup
  - Need to use SSL to PG, and TLS with LDAP
- trust
  - Allows any connection to connect as any user

Authorization

- Container objects
  - Databases
  - Schemas
- To access objects inside containers-
  - Must have CONNECT privs on the database
  - Must have USAGE privs on the schema

GRANT / REVOKE

- GRANT <privs> ON <object> TO <roles>;
- REVOKE <privs> ON <object> FROM <roles>;
- GRANT ... ON ALL <objtype> IN <schema> ...
- "PUBLIC" means "everyone"
- WITH GRANT OPTION allows role to re-grant priv

GRANT / REVOKE

- Owning the object grants all rights, and then some
- Only owner of object can DROP the object

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

...
```

Database Privileges

- CREATE (Able to create *schemas*)
- CONNECT (Granted to PUBLIC by default)
- TEMPORARY (Able to create temporary *objects*)
- Owners can use ALTER DATABASE to
  - RENAME
  - OWNER
  - SET TABLESPACE
  - SET other config options

Schema Privileges

- CREATE (Able to create *objects* in the schema)
- USAGE (Able to *see* objects- need rights on them)
- "public" schema defaults with CREATE to PUBLIC
- Owners can use ALTER SCHEMA to
  - RENAME
  - OWNER

Table Privileges

- SELECT (SELECT *any*/all columns)
- INSERT (INSERT *any*/all columns)
- UPDATE (UPDATE *any*/all columns)
- DELETE
- TRUNCATE (Not the same as DELETE FROM ...)
- REFERENCES (Can create a FK *to* the table)
- TRIGGER (Can create a trigger on the table)

Table Privileges

- Table owners can use ALTER TABLE to
  - ADD/DROP COLUMN
  - ADD/DROP Constraints
  - OWNER
  - CLUSTER
  - INHERIT / NOINHERIT
  - Lots of stuff...

Column Privileges

- SELECT (Only select out specified column)
- INSERT (Can only insert non-default values)
- UPDATE (Can only update these columns)
- REFERENCES (Can only reference specified column)
- Table owners can ALTER TABLE .. ALTER COLUMN to
  - SET/DROP DEFAULT expression
  - SET STATISTICS (target)
  - SET DATA TYPE
  - SET STORAGE

Sequence Privileges

- USAGE (currval && nextval)
- SELECT (Only currval)
- UPDATE (nextval && setval / reset sequence)

Function Privileges

- EXECUTE
- Granted to "PUBLIC" by default!
- Use caution with SECURITY DEFINER

Tablespace Privileges

- CREATE
- User allowed to create objects in tablespace
- Any kind of object allowed
- Can be temp or non-temp (even if temp tablespace)
- Database Default Tablespace
  - Skips tablespace priv checking
  - Only for connections to that DB

Usage Privileges

- Objects with just USAGE privs
- DOMAIN
- FOREIGN DATA WRAPPER
- FOREIGN SERVER
- LANGUAGE
- TYPE

Web-Scale

- Roles exist in a PG shared catalog
- Common across all DBs
- Unable to be partitioned
- Could be sharded..
  - Unable to set CHECK constraints
  - No triggers
  - etc..
- BUT- use roles also

Roles for Web-Scale

- Use tables for website users
- Use roles for permissions management
- Independent roles for ETL, daemon, etc

Roles for Web-Scale

- Read-only role
  - Only has read access
  - Useful for scaling out with read slaves
- Read/write role(s)
  - Possibly more than one (eg: per site)
  - Minimize access to what code "should" do

Enterprise Deployment

- Individual logins per user
- Roles for permissions management
- Roles to manage access to databases
- Kerberos / GSS / AD integration / Pass-thru

Enterprise Deployment

- Views
  - Limit rows individual users can see
  - Security Barrier
- PL/PgSQL Functions
  - Control writes- include auditing
  - Security Definer

Security Labels

- Defines labels for objects in PG
- Hooks for security providers (eg: sepgsql)

```
SECURITY LABEL [ FOR provider ] ON
{
  TABLE object_name |
  COLUMN table_name.column_name |
  AGGREGATE aggregate_name ( aggregate_signature ) |
  DATABASE object_name |
  DOMAIN object_name |
  EVENT TRIGGER object_name |
  FOREIGN TABLE object_name
  FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [, ...] ] ) |
  LARGE OBJECT large_object_oid |
  MATERIALIZED VIEW object_name |
  [ PROCEDURAL ] LANGUAGE object_name |
  ROLE object_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  TABLESPACE object_name |
  TYPE object_name |
  VIEW object_name
} IS 'label'
```

Additional Security

- SELinux Integration
  - sepgsql security provider
  - Works with SECURITY LABEL
- EVENT Triggers
  - Can prevent certain actions
- Row-Level Security being worked on
- Updatable security-barrier views

Questions?

resonate

Thank you!

Stephen Frost
sfrost@snowman.net
@net_snow

resonate