Mandatory Access Control in PostgreSQL - giving users ownership of their data

Leon du Toit

2019-01-15

Outline

- why take data ownership seriously?
- why Mandatory Access Control?
- a brief introduction to the `pg-need-to-know` module
- a use case to demostrate features:
  - For users: ownership, insight and consent-based usage
  - For administrators: fine-grained access control, audit information
  - For developers: a rich REST API, with a built-in authorization model

Why take data ownership seriously?

- Regulations of the GDPR
  - increased focus on data privacy and protection
  - right to access
  - right to be forgotten
  - data portability
  - consent-based data usage
  - increased demand for audit information
- To counter surveilance capitalism
  - you (and your data) are the product
  - building applications to fight this trend

What Mandatory Access Control?

- *enforcible* policies, in constrast to Discretionary Access Control
- enables consent-based data access
- supports granular access needs

pg-need-to-know

- PostgresQL “module” - really just a set of tables, views, and functions
- implements Mandatory Access Control
- more limited approach than SEPostgreSQL
- source: https://github.com/leondutoit/pg-need-to-know
- written in PL/pgSQL
  - procedural language, extending SQL with control structures
  - used to create functions
  - ~1000 sloc, another ~1500 for tests
- designed to be used via a REST API

Use case

Key terms:

- data owner: provides data about themselves
- data user: analyses data about others
- admin: creates and implements access control policies

Use case

Assume the following setup:

```
data owners: A, B, C, D, E, F
tables: t1, t2, containing data from all data owners
data users: X, Y, Z
```

Use case

Now suppose we need to set up the following access control rules in our DB:

- data users X, and Y should only have access to data in tables t1 and only data from owners A, B, C, D
- data user Z should have access to all data - i.e. tables t1, t2

Use case

Using pg-need-to-know, we implement this with the following groups, and table grants:

```
group1
    - members: ((X, Y), (A, B, C, D))
    - select table access grant: (t1)
group2
    - members: ((Z), (A, B, C, D, E, F))
    - select table access grants: (t1, t2)
```

pg-need-to-know via REST

```
HTTP client -> webapp -> REST server -> (pg-need-to-know, P
```