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
- optionally: a look at some implementation details

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
- Respecting people

Why 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
- uses Row-Level Security policies to implement MAC
- 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 access control policies

Use case

Assume the following setup:

- data owners: A, B, C, D, E, F
- data users: X, Y, Z
- tables: spending_habits, personal_details, containing data from all data owners

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 table spending_habits and only data from owners A, B, C, D
- data user Z should have access to all data - i.e. tables spending_habits, personal_details

Use case

A hypothetical sequence of events:

1. admin creates tables
2. data owners and data users register themselves, data is collected
3. admin creates groups, adds members, adds table grants
4. data is analysed
5. users manage their own data
6. admins get audit insights
7. developers create applications using composing these features

Table creation

```
set role admin_user;
SET
select table_create(
    '{"table_name": "spending_habits",
      "columns": [
        {"name": "spending", "type": "int",
         "description": "Amount spent in NOK"},
        {"name": "item_type", "type": "text",
         "description": "Type of item purchased"},
        {"name": "purchase_date", "type": "date",
         "description": "Year-Month-Day on which purchase occurred"} ],
      "description": "data about spending habits"}'::json,
    'mac');
 table_create
--------------
 Success
(1 row)
```

Figure 1:Creating a new table

User registration

- can require consent before user registration
- data collection not possible without registration

Group setup, table grants

- can link consent(s) to groups via group metadata
- group1
  - members: ((X, Y), (A, B, C, D))
  - select table access grant: (spending_habits)
- group2
  - members: ((Z), (A, B, C, D, E, F))
  - select table access grants: (spending_habits, personal_details)

Data analysis

```
set session "request.jwt.claim.user" = 'user_X';
SET
select current_setting('request.jwt.claim.user');
 current_setting
-----------------
 user_X
(1 row)

select * from spending_habits;
               row_id                | row_owner | row_originator | spending | item_type | purchase_date
-------------------------------------+-----------+----------------+----------+-----------+---------------
 4ad3b11e-32ff-42a1-850c-aff1f93f190e | owner_A   | owner_A        |      140 | food      | 2019-01-02
 975f2758-5749-4915-bac1-48530f703062 | owner_A   | owner_A        |      100 | drink     | 2019-01-03
 899efca4-a935-4e0d-ba25-29c4413d7c2a | owner_B   | owner_B        |       60 | drink     | 2019-01-02
 7ef73351-1e7d-4f26-989b-d55fa0f1bfa5 | owner_B   | owner_B        |       78 | drink     | 2019-01-04
 1be698b3-e1c0-4b98-a236-f273883f67dc | owner_C   | owner_C        |     1020 | travel    | 2019-01-04
 c225db92-2171-4d21-9b7a-67c4ef0ad942 | owner_C   | owner_C        |      101 | food      | 2019-01-04
 123f6322-130a-4b13-8f2a-2dbe0f0a9523 | owner_D   | owner_D        |      230 | travel    | 2019-01-05
 ca97c462-7fea-49ce-8bde-fcef08a910ab | owner_D   | owner_D        |      448 | travel    | 2019-01-06
(8 rows)

select * from personal_details;
psql:./src/11-user-X-data-access.sql:7: ERROR:  access denied to table
CONTEXT:  PL/pgSQL function ntk.data_user_group_membership_with_correct_privileges(uuid,text,text) line 18 at
RAISE
```

Figure 2:User X's data access

Data analysis

```
set session "request.jwt.claim.user" = 'user_Z';
SET
select * from spending_habits;
               row_id                | row_owner | row_originator | spending | item_type | purchase_date
-------------------------------------+-----------+----------------+----------+-----------+---------------
 4ad3b11e-32ff-42a1-850c-aff1f93f190e | owner_A   | owner_A        |      140 | food      | 2019-01-02
 975f2758-5749-4915-bac1-48530f703062 | owner_A   | owner_A        |      100 | drink     | 2019-01-03
 899efca4-a935-4e0d-ba25-29c4413d7c2a | owner_B   | owner_B        |       60 | drink     | 2019-01-02
 7ef73351-1e7d-4f26-989b-d55fa0f1bfa5 | owner_B   | owner_B        |       78 | drink     | 2019-01-04
 1be698b3-e1c0-4b98-a236-f273883f67dc | owner_C   | owner_C        |     1020 | travel    | 2019-01-04
 c225db92-2171-4d21-9b7a-67c4ef0ad942 | owner_C   | owner_C        |      101 | food      | 2019-01-04
 123f6322-130a-4b13-8f2a-2dbe0f0a9523 | owner_D   | owner_D        |      230 | travel    | 2019-01-05
 ca97c462-7fea-49ce-8bde-fcef08a910ab | owner_D   | owner_D        |      448 | travel    | 2019-01-06
 fc56af9e-b361-4f9d-814a-3cab834730fd | owner_E   | owner_E        |    10230 | housing   | 2019-01-01
 d2f1e45f-e3c0-4fae-8c3d-1470fc4fb75e | owner_F   | owner_F        |      209 | food      | 2019-01-06
(10 rows)

select * from personal_details;
               row_id                | row_owner | row_originator |       name       | age
-------------------------------------+-----------+----------------+------------------+-----
 5a2a949e-89e5-413d-b268-27516e4924b4 | owner_A   | owner_A        | James Martin     |  44
 336d4202-394c-4abc-9231-3127431df3e8 | owner_B   | owner_B        | Sandra Fourie    |  18
 ce67a250-dc92-48b1-882f-ef68c9ba9687 | owner_C   | owner_C        | Willem White     |  11
 e0ab7e50-3f81-4180-9c5e-b0423e8e17af | owner_D   | owner_D        | Lee Simpson      |  84
 0b8cb4f0-78a0-448d-8ad5-173e94e1c488 | owner_E   | owner_E        | Gerhard du Preez |  23
 0a7280e6-e19c-457f-82db-d2b40190ef7d | owner_F   | owner_F        | Hannah Furgeson  |  33
```

Figure 3:User Z's data access

Data ownership

- right to access
- data portability
- right to be forgotten

Right to access

```
set role data_owner;
SET
set session "request.jwt.claim.user" = 'owner_A';
SET
select * from spending_habits;
               row_id               | row_owner | row_originator | spending | item_type | purchase_date
------------------------------------+-----------+----------------+----------+-----------+---------------
 4ad3b11e-32ff-42a1-850c-aff1f93f190e | owner_A   | owner_A        |      140 | food      | 2019-01-02
 975f2758-5749-4915-bac1-48530f703062 | owner_A   | owner_A        |      100 | drink     | 2019-01-03
(2 rows)

select * from personal_details;
               row_id               | row_owner | row_originator |     name     | age
------------------------------------+-----------+----------------+--------------+-----
 5a2a949e-89e5-413d-b268-27516e4924b4 | owner_A   | owner_A        | James Martin |  44
(1 row)
```

Figure 4:Owner A's data access

Data portability

- owner A can simply download their data

Right to be forgotten

```
set session "request.jwt.claim.user" = 'owner_B';
SET
select * from spending_habits;
               row_id                | row_owner | row_originator | spending | item_type | purchase_date
-------------------------------------+-----------+----------------+----------+-----------+---------------
 899efca4-a935-4e0d-ba25-29c4413d7c2a | owner_B   | owner_B        |       60 | drink     | 2019-01-02
 7ef73351-1e7d-4f26-989b-d55fa0f1bfa5 | owner_B   | owner_B        |       78 | drink     | 2019-01-04
(2 rows)

select * from personal_details;
               row_id                | row_owner | row_originator |     name      | age
-------------------------------------+-----------+----------------+---------------+-----
 336d4202-394c-4abc-9231-3127431df3e8 | owner_B   | owner_B        | Sandra Fourie |  18
(1 row)

select user_delete_data();
psql:./src/15-owner-B-delete-data.sql:8: NOTICE:  cannot delete data from test1, permission denied
psql:./src/15-owner-B-delete-data.sql:8: NOTICE:  cannot delete data from testing, permission denied
 user_delete_data
------------------
 all data deleted
(1 row)

select * from spending_habits;
 row_id | row_owner | row_originator | spending | item_type | purchase_date
--------+-----------+----------------+----------+-----------+---------------
(0 rows)

select * from personal_details;
 row_id | row_owner | row_originator | name | age
--------+-----------+----------------+------+-----
(0 rows)
```

Figure 5:Owner B deletes their data

Audit insights

- data access
- access control changes
- user initiated group removals
- user initiated data deletions
- data updates

Audit: data access

```
select * from event_log_data_access;
         request_time          |    table_name    |               row_id                | data_user | data_owner
-------------------------------+------------------+-------------------------------------+-----------+------------
 2019-01-09 10:34:45.646518+01 | spending_habits  | 8719fc2e-5a59-4db0-906f-d844b496de2b | user_X    | owner_A
 2019-01-09 10:34:45.646518+01 | spending_habits  | 89b9bfe9-460f-47aa-bb25-b71ae78d7219 | user_X    | owner_A
 2019-01-09 10:34:45.646518+01 | spending_habits  | fbc12163-20e7-4889-b5db-a89e6c8dc210 | user_X    | owner_B
 2019-01-09 10:34:45.646518+01 | spending_habits  | 3307c91a-1d57-4725-b239-53c94f5bb568 | user_X    | owner_B
 2019-01-09 10:34:45.646518+01 | spending_habits  | 94989d0f-8537-4b0a-993c-59687a475f35 | user_X    | owner_C
 2019-01-09 10:34:45.646518+01 | spending_habits  | 8478942d-b37e-4f6d-9985-e12661b81234 | user_X    | owner_C
 2019-01-09 10:34:45.646518+01 | spending_habits  | 1a314e10-1768-45b0-bc0c-992b60c17b0c | user_X    | owner_D
 2019-01-09 10:34:45.646518+01 | spending_habits  | dbdf9389-38ff-4561-90bd-a496ed7cb5b6 | user_X    | owner_D
 2019-01-09 10:34:47.853098+01 | spending_habits  | 8719fc2e-5a59-4db0-906f-d844b496de2b | user_Z    | owner_A
 2019-01-09 10:34:47.853098+01 | spending_habits  | 89b9bfe9-460f-47aa-bb25-b71ae78d7219 | user_Z    | owner_A
 2019-01-09 10:34:47.853098+01 | spending_habits  | fbc12163-20e7-4889-b5db-a89e6c8dc210 | user_Z    | owner_B
 2019-01-09 10:34:47.853098+01 | spending_habits  | 3307c91a-1d57-4725-b239-53c94f5bb568 | user_Z    | owner_B
 2019-01-09 10:34:47.853098+01 | spending_habits  | 94989d0f-8537-4b0a-993c-59687a475f35 | user_Z    | owner_C
 2019-01-09 10:34:47.853098+01 | spending_habits  | 8478942d-b37e-4f6d-9985-e12661b81234 | user_Z    | owner_C
 2019-01-09 10:34:47.853098+01 | spending_habits  | 1a314e10-1768-45b0-bc0c-992b60c17b0c | user_Z    | owner_D
 2019-01-09 10:34:47.853098+01 | spending_habits  | dbdf9389-38ff-4561-90bd-a496ed7cb5b6 | user_Z    | owner_D
 2019-01-09 10:34:47.853098+01 | spending_habits  | f3d9bbd8-58fa-42a6-a150-d9bb44a2e2d4 | user_Z    | owner_E
 2019-01-09 10:34:47.853098+01 | spending_habits  | 276e1f83-8739-4c62-9eb3-e8724fd5ff04 | user_Z    | owner_F
 2019-01-09 10:34:47.878906+01 | personal_details | 31b62746-e497-42e7-848e-82438898785b | user_Z    | owner_A
 2019-01-09 10:34:47.878906+01 | personal_details | ff695fec-2f96-40d6-b3ed-42090892c88d | user_Z    | owner_B
 2019-01-09 10:34:47.878906+01 | personal_details | 378207c9-022a-48e3-ba82-d418b5b7c76f | user_Z    | owner_C
 2019-01-09 10:34:47.878906+01 | personal_details | bdcfdb9b-153d-44ed-857c-26f76064e6e7 | user_Z    | owner_D
 2019-01-09 10:34:47.878906+01 | personal_details | df105084-e8e3-4d2d-bb9e-5ee3e836feb8 | user_Z    | owner_E
 2019-01-09 10:34:47.878906+01 | personal_details | 41a8bf9c-a1e1-409d-a9f3-264c8c4e962b | user_Z    | owner_F
(24 rows)
```

Figure 6:Data access audit logs

Audit: access control changes

```
select * from event_log_access_control;
 id  |          event_time           |       event_type        | group_name |      target
-----+-------------------------------+-------------------------+------------+------------------
 383 | 2019-01-07 15:49:00.550321+01 | group_create            | group1     |
 384 | 2019-01-07 15:49:02.68408+01  | group_create            | group2     |
 385 | 2019-01-07 15:49:04.53359+01  | group_member_add        | group1     | owner_A
 386 | 2019-01-07 15:49:04.53359+01  | group_member_add        | group1     | owner_B
 387 | 2019-01-07 15:49:04.53359+01  | group_member_add        | group1     | owner_C
 388 | 2019-01-07 15:49:04.53359+01  | group_member_add        | group1     | owner_D
 389 | 2019-01-07 15:49:04.53359+01  | group_member_add        | group1     | user_X
 390 | 2019-01-07 15:49:04.53359+01  | group_member_add        | group1     | user_Y
 391 | 2019-01-07 15:49:07.356862+01 | group_member_add        | group2     | user_Z
 392 | 2019-01-07 15:49:07.365324+01 | group_member_add        | group2     | owner_A
 393 | 2019-01-07 15:49:07.365324+01 | group_member_add        | group2     | owner_B
 394 | 2019-01-07 15:49:07.365324+01 | group_member_add        | group2     | owner_C
 395 | 2019-01-07 15:49:07.365324+01 | group_member_add        | group2     | owner_D
 396 | 2019-01-07 15:49:07.365324+01 | group_member_add        | group2     | owner_E
 397 | 2019-01-07 15:49:07.365324+01 | group_member_add        | group2     | owner_F
 398 | 2019-01-07 15:49:10.540968+01 | table_grant_add_select  | group1     | spending_habits
 399 | 2019-01-07 15:49:11.772743+01 | table_grant_add_select  | group2     | spending_habits
 400 | 2019-01-07 15:49:11.785069+01 | table_grant_add_select  | group2     | personal_details
(18 rows)
```

Figure 7:Access control audit logs

Audit: user initiated group removals

```
select * from event_log_user_group_removals;
          removal_date           | user_name | group_name
---------------------------------+-----------+------------
 2019-01-07 15:49:30.547254+01 | owner_A   | group1
(1 row)
```

Figure 8:User group removals audit logs

Audit: user initiated data deletions

```
set role admin_user;
SET
set session "request.jwt.claim.user" = '';
SET
select * from event_log_user_data_deletions;
 user_name |          request_date
-----------+-------------------------------
 owner_B   | 2019-01-07 15:49:32.350643+01
(1 row)
```

Figure 9:User data deletion audit logs

Audit: data updates

```
select * from event_log_data_updates;
         updated_time          | updated_by |    table_name    |               row_id                | column_name | old_data | new_data
-------------------------------+------------+------------------+-------------------------------------+-------------+----------+----------
 2019-01-07 15:49:41.236255+01 | owner_C    | personal_details | ce67a250-dc92-48b1-882f-ef68c9ba9687 | age         | 11       | 55
(1 row)
```

Figure 10:Data update audit logs

Application development

Architecture:

```
webapp -> REST -> (pg-need-to-know, PostgresQL)
```

- developers can focus on business logic
- authorization taken care of
- authentication is left to the webapp implementor

postgrest

- `pg-need-to-know` designed to be used with postgrest
- open source project written in Haskell
- provides a REST API for any PostgreSQL DB
- https://github.com/leondutoit/pg-need-to-know/blob/master/api/http-api.md
- `pg-need-to-know` requires a custom compilation of this server due to audit logging
- available here: https://github.com/leondutoit/postgrest-need-to-know

Authentication requirements

- webapp must provide an access token at request time
- a JWT with the following claims:
  - `exp: expiry time`
  - `role: <data_owner, data_user, admin_user>`
  - `user: user name`
- `pg-need-to-know` provides a `/token` endpoint for access token generation
- but developers can implement their own
- refernce client for HTTP API: https://github.com/leondutoit/py-need-to-know

Implementation details

```
                                Table "public.spending_habits"
     Column     |  Type   | Collation | Nullable |                    Default
----------------+---------+-----------+----------+-----------------------------------------------
 row_id         | uuid    |           | not null | gen_random_uuid()
 row_owner      | text    |           | not null | current_setting('request.jwt.claim.user'::text)
 row_originator | text    |           | not null | current_setting('request.jwt.claim.user'::text)
 spending       | integer |           |          |
 item_type      | text    |           |          |
 purchase_date  | date    |           |          |
Check constraints:
    "spending_habits_row_originator_check" CHECK (row_originator = current_setting('request.jwt.claim.user'::text))
Foreign-key constraints:
    "spending_habits_row_originator_fkey" FOREIGN KEY (row_originator) REFERENCES ntk.registered_users(_user_name)
    "spending_habits_row_owner_fkey" FOREIGN KEY (row_owner) REFERENCES ntk.registered_users(_user_name)
Policies (forced row security enabled):
    POLICY "data_user_select_policy" FOR SELECT
      TO data_user
      USING (ntk.data_user_group_membership_with_correct_privileges(row_id, row_owner, 'spending_habits'::text))
    POLICY "row_originator_update_policy" FOR UPDATE
      USING (ntk.is_row_originator(row_originator))
    POLICY "row_ownership_delete_policy" FOR DELETE
      USING (ntk.is_row_owner(row_owner))
    POLICY "row_ownership_insert_policy" FOR INSERT
      WITH CHECK (true)
    POLICY "row_ownership_select_policy" FOR SELECT
      USING (ntk.is_row_owner(row_owner))
    POLICY "row_ownership_update_policy" FOR UPDATE
      USING (ntk.is_row_owner(row_owner))
Triggers:
    immutable_trigger BEFORE UPDATE ON spending_habits FOR EACH ROW EXECUTE PROCEDURE ntk.ensure_internal_columns_are_immutable()
    update_trigger AFTER UPDATE ON spending_habits FOR EACH ROW EXECUTE PROCEDURE ntk.log_data_update()
```

Figure 11:Example table definition

More info

- watch a demo recording: https://asciinema.org/a/c3XIyrfnoLixofqiSbx8p0l21
- read the docs: https://github.com/leondutoit/pg-need-to-know/tree/master/docs
- this presentation, and materials: https://github.com/leondutoit/pg-ntk-demo