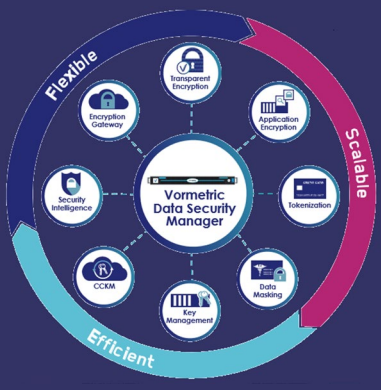
THALES

# VORMETRIC guide

## VTE implementation

## for postgreSQL



# Table of contents

# 1. VTE OVER POSTGRES

## 1.1 POSTGRES ARCHITECTURE

The data in a postgreSQL database is usually found under:

- ⇨ Linux: /var/lib/pgsql/data
- ⇨ Windows: C: \Program Files\PostgreSQL\some version\data

A $PGDATA environment variable is used to find the location of the data.

By default, the database content of a postgreSQL instance is in the $PGDATA/base subdirectory of the main data directory.

The $PGDATA/ base directory is the default tablespace for a postgreSQL instance.

The $PGDATA/ base directory contains a directory for each of the declared databases whose name is the Object Identifier (OID). A query is used to find the correspondence between the OID and the name given to the database. This request is indicated later in this document.

The directory of each database (thus under $PGDATA/base/<xxxxx> with <xxxxx> the OID of the database) contains a set of files but no directory.

Here is the detail:

- *pg_internal.init* file: this file exists by base. It is an index file of objects (tables, views, indexes, triggers etc.) and must necessarily be accessed by POSTGRES for performance reasons.
- *PG_VERSION* file: created when creating the database, it contains a major version number of the base instance
- *pg_filenode.map* file: contains the file name mapping - SQL object name for some relationships (mapped catalogs)
- *xxxxx* files (without extension or with extension including a number): contain the data of relations (tables, materialized views, indexes, sequences)
- *xxxxx_init* files (numbered with _init extension): match the log table or index empty
- *xxxxx_fsm* files (numbered with _fsm extension): contain the FSM structure of each table
- *xxxxx_vm* files (numbered with _vm extension): contain the VM structure of each table

## 1.2 CONTEXT

This implementation guide is based on tests performed on a CentOS7 system.
The base directory used in this guide is: /var/lib/pgsql/data/base/16386

In this test are used:

- A *postgres* system user (default user)
- A *vormetric* system user, specific to the vormetric database
- A *root* user, privileged user
- A *user1* user, privileged user
- A postgres role called "vormetric"
- POSTGRES configuration with a system user by role

## 1.3 PRE-REQUISITES

The POSTGRES environment must be set up.

⇨ Install (on CENTOS7):

```
sudo yum install postgresql-server postgresql-contrib
```

⇨ Create new database cluster:

```
sudo postgresql-setup initdb
```

⇨ Adapt the POSTGRES main configuration file *pg_hba.conf* to define access, per system user, in our case:

```
sudo vi /var/lib/pgsql/data/pg_hba.conf
```

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only

local    postgres        postgres                                  peer

local    vormetric       postgres                                  peer

local    vormetric       vormetric                                 peer
```

⇨ Enable and start database :

```
sudo systemctl start postgresql

sudo systemctl enable postgresql
```

⇨ Create a *vormetric* user:

```
[root@CentOS7-VTE ~]# sudo adduser vormetric
```

⇨ And connect to the database as *POSTGRES* user:

```
[root@CentOS7-VTE ~]# su - postgres
```

⇨ Then create a new role, which will be "matched" with the *vormetric* system user :

```
-bash-4.2$ createuser --interactive

-bash-4.2$ Enter name of role to add: vormetric

-bash-4.2$ Shall the new role be a superuser? (y/n) n

-bash-4.2$ Shall the new role be allowed to create databases? (y/n) y

-bash-4.2$ Shall the new role be allowed to create more new roles? (y/n) n
```

⇨ Exit, then log in as a *vormetric* user :

```
-bash-4.2$ \q

[root@CentOS7-VTE ~]# sudo -i -u vormetric
```

⇨ Then create the vormetric test database:

```
-bash-4.2$ createdb vormetric
```

⇨ Exit, reconnect and verify that by default the *vormetric* user is well connected on the same name basis:

```
vormetric-> \conninfo

You are connected to database "vormetric" as user "vormetric" via socket in
"/var/run/postgresql" at port "5432".
```

⇨ Exit and import the *vormetric.sql* dump into the vormetric database:

```
psql vormetric < /home/vormetric/DEMO_POSTGRES/vormetric.sql
```

⇨ Check tables list:

```
vormetric-> \dt

                        List of relations

 Schema |                  Name                   | Type  |   Owner

--------+-----------------------------------------+-------+-----------

 public | addresses                               | table | vormetric

 public | affiliation_phases                      | table | vormetric

 public | affiliations                            | table | vormetric

 public | affiliations_documents                  | table | vormetric

 public | affiliations_events                     | table | vormetric

 public | affiliations_media                      | table | vormetric

 public | american_football_action_participants | table | vormetric

...
```

⇨ Check one of them, for example, *injury_phases* table :

```
vormetric=> SELECT * FROM injury_phases;

 id  | person_id | injury_status |   injury_type    | injury_comment | disabled_list |
start_date_time   | end_date_time | season_id |    phase_type     | injury_side

-----+-----------+---------------+------------------+----------------+---------------
+---------------------+---------------+-----------+-------------------+-------------

  10 |       491 |               | other-excused    |                |               |
2007-08-03 00:00:00 |               |           | out               |

  14 |       553 |               | leg              |                |               |
2007-08-03 00:00:00 |               |           | out               |

…
```

⇨ Now, target the directory that corresponds to the vormetric database (in order to encrypt it) :

```
vormetric=> SELECT oid from pg_database where datname = 'vormetric';

  oid

-------

 16386

(1 row)
```

The OID identifier for *vormetric* database is *16386*.

The directory to protect is: /var/lib/pgsql/data/base/16386/

Note that it is possible to target a specific table :

```
vormetric=> SELECT relname, relfilenode FROM pg_class WHERE relname like
'%injury_phases%';

       relname        | relfilenode

----------------------+-------------

 injury_phases_id_seq |       16788

 injury_phases        |       16790

 injury_phases_id_key |       17243

 idx_injury_phases_2  |       17403

 idx_injury_phases_3  |       17404

 idx_injury_phases_4  |       17405

(6 rows)
```

The table *injury_phases* is therefore under the directory: /var/lib/pgsql/data/base/16386/16790

Useful: here is the command to check the version of the POSTGRES database:

```
postgres=# SELECT version();
```

# 1.4 POSTGRES DATABASE PROTECTION

## 1.4.1 Assumptions

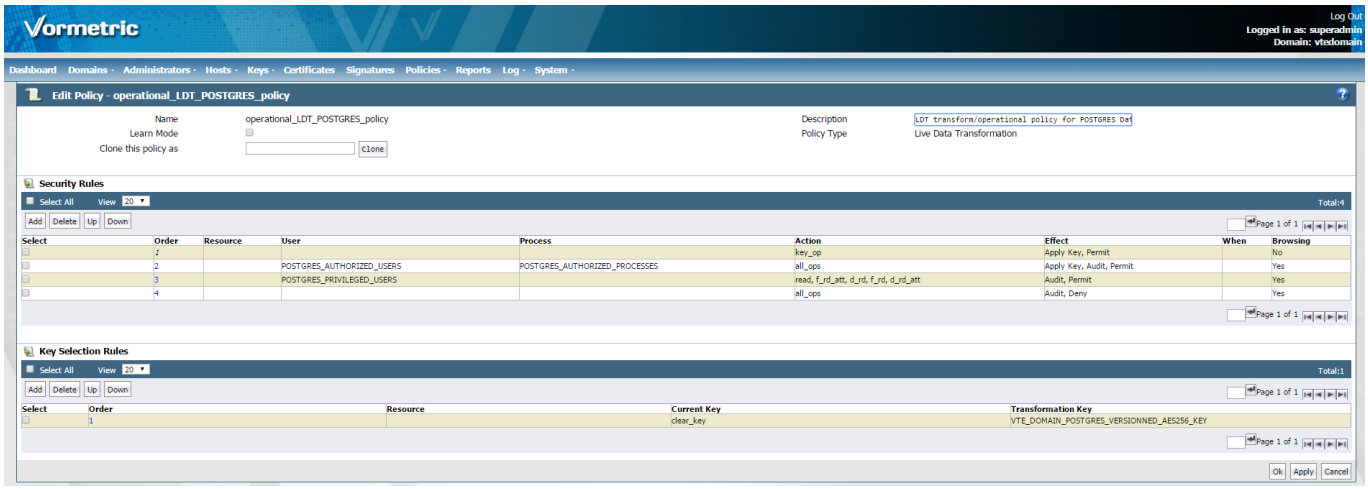
We assume here a DSM environment already in place with:

- A domain administrator
- A host administrator
- A log administrator
- The POSTGRES host machine with a VTE agent registered with the DSM
- An encryption key dedicated to the POSTGRES database:
  - Name VTE_DOMAIN_POSTGRES_VERSIONNED_AES256_KEY
  - Versionned (required for LDT)
  - With AES256 algorithm (recommended)
- LDT option enabled
- A valid license (with LDT)

## 1.4.2 Security policies

### 1.4.2.1 Security rules

Here is an example of a possible security policy configuration for POSTGRES :




With :

| Rule | Description | Clé ? |
|---|---|---|
| Rule 1 | *LDT default rule* | *YES* |
| Rule 2 | Postgres and vormetric POSTGRES (roles) specific user rule, with the right to interact fully with the data but only under the postgres process. | YES |
| Rule 3 | Rule for privileged system administrators, here root, allowed access to read metadata, to ensure a continuity of Administration. | NO |
| Rule 4 | *Default rule of type DENY ALL.* | NO |

### 1.4.2.2 Users, Process & Resource sets

- System users :

| Nom | Description |
|---|---|
| POSTGRES_AUTHORIZED_USERS | • System users corresponding to POSTGRES roles<br>• Allowed to apply the key<br>• Typically *postgres* and *vormetric* users. |
| POSTGRES_PRIVILEGED_USERS | • Users tolerated to interact with protected resources<br>• Not allowed to apply the key (so to encrypt / decrypt)<br>• Here we choose *root* and *user1*. |

- Process :

| Nom | Description |
|---|---|
| POSTGRES_AUTHORIZED_PROCESSES | • Processus systèmes autorises à appliquer la clé<br>• Typiquement /usr/bin/postgres |

- Resources:

| Nom | Description |
|---|---|
| - | ⇨ *(Warning: the resourceSet paths are relative to the GuardPoint)* |

### 1.4.2.3 Host settings

#### 1.4.2.3.1 Postgres user Authentification

For the purposes of this test, here are examples of the necessary Host settings configuration to declare processes that authenticate the *postgres* user.

| Parameter | Description |
|---|---|
| \|authenticator\|/usr/sbin/sshd | Configuration already in place in the default Host Settings.<br>This makes it possible to authenticate the user postgres via ssh. |
| \|authenticator_euid\|/bin/bash | "Trust" is given to the bash process to authenticate the postgres user. *|
| \|authenticator_euid\|/usr/bin/postgres | "Trust" is given to the *postgres* process to authenticate the *postgres* user. |

* Without the host settings parameter for bash, the server can not start:

```
[root@CentOS7-VTE ~]# sudo -i -u postgres pg_ctl start

server starting

[root@CentOS7-VTE ~]# FATAL:  could not read directory "base/16386": Permission
denied
```

In case of authentication problem (or failure in the Host Settings), the logs of the DSM reveal it:

```
CGP2604E: [SecFS, 0] PID[3175] [ALARM] Policy[operational_LDT_POSTGRES_policy]
User[postgres,uid=26 (User Not Authenticated)] Process[/usr/bin/postgres]
Action[read_dir] Res[/var/lib/pgsql/data/base/16386/] Effect[DENIED Code
(1A,2U,3U,4U,5U,6M)]
```

Reminder: In order to know the processes (authenticating postgres), to declare in the hosts settings, just use the command "pstree -u" on the non authenticated user:

```
-bash-4.2$ pstree -u | grep postgres
```

Typically, if we execute this command through an ssh console, through which we connected via the user *postgres*, we get this:

```
-bash-4.2$ pstree -u | grep postgres

        |-sshd-+-sshd---sshd(postgres)---sftp-server

        |      `-sshd---sshd(postgres)---bash-+-grep
```

Postgres has been authenticated, here in our case, through the sshd process. Therefore, sshd must be declared as a "trust" authentication process.

We find it via:

```
-bash-4.2$ which sshd

/usr/sbin/sshd
```

#### 1.4.2.3.2 Configuration files & other sensitive files

It may be useful to apply a Host Settings type setting to protect some files from any changes or at least be notified if changes occur:

| Parameter | Description |
|---|---|
| \|protect\|/path/to/sensitive/file | Modification forbidden |
| \|audit\|/path/to/sensitive/file | Modification tolerated and notified. |

In our case, we decide to apply this protection on the POSTGRES global configuration / clustering file:

```
[root@CentOS7-VTE ~]# ll /var/lib/pgsql/data/pg_hba.conf

-rw------- 1 postgres postgres 4449 Mar 21 17:54 /var/lib/pgsql/data/pg_hba.conf
```

In case of modification, the logs, sent back to the DSM, reveal the modification of the file:

```
VMD3918E: [vmd, 2207] PID[2207] An unauthorized attempt to update the authentication
signature for file /var/lib/pgsql/data/pg_hba.conf has been prevented
```