Securing PostgreSQL – Exploring PostgreSQL Features, Extensions, and Guides

Joe Conway
joe.conway@crunchydata.com
mail@joeconway.com

Crunchy Data

2018-06-01

Securing PostgreSQL

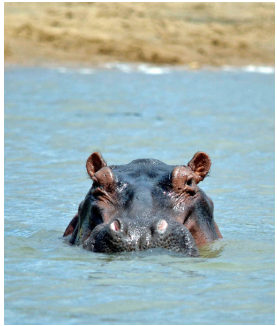
- PostgreSQL and Ecosystem: Security Features
- CIS Benchmark and Security Technical Implementation Guide (STIG)
- Related postgresql.conf settings and pg_hba.conf rules
- Appendix: set_user, pgaudit, RLS Timetravel

http://www.postgresql.org

https://postgresql.us/

Security

- International Recognition
  - Common Criteria, ISO/IEC 15408 (CC)
  - Security Technical Implementation Guide (STIG)
  - Center for Internet Security (CIS) Benchmark (Currently DRAFT - open for comments)
- Features
  - Perimeter
  - Internal
  - Chronological



Operating System

- OS Configuration
  - FIPS 140-2 compliance
  - STIG or CIS Benchmark
- Discretionary Access Control (DAC)
  - Not privileged account
  - Runtime perm checks
- Mandatory Access Control (MAC)
  - SELinux: Confined (RHEL - MCS Policy)
- Encryption at rest
  - Filesystem encryption, many options

Client-server

- Authentication
  - Host based authentication
  - Internal: md5*, SCRAM-SHA-256, cert (SSL)
  - OS: PAM, peer, ident
  - External: GSSAPI, SSPI, LDAP, RADIUS
- Encryption in transit
  - SSL

DAC

- ROLE
  - vs. USER and GROUP
  - Hierarchical
- GRANT and REVOKE
  - Follows SQL Standard reasonably closely
  - Covers virtually all DB Objects
- Encryption
  - pg_crypto: PGP, OpenSSL; hashing and encryption
  - Application encryption always possible

MAC

- sepgsql: SELinux bindings
  - RBAC Type Enforcement covers most DB Objects
  - Can combine with custom SELinux policy for powerful control

Row Level Security

- Tables can have row security policies
- Restrict, on a per-user basis
  - Which rows visible to normal queries
  - What can be inserted, updated, or deleted

set_user extension

- set_user extension
  - Allows switching users and privilege escalation with enhanced logging and control
  - Enhanced logging ensures an audit trail
  - More detail in Appendix

set_user - Why?

- PostgreSQL superuser capabilities
  - Bypass all DAC
  - Bypass RLS
  - Load any library
  - `COPY ... PROGRAM` (execute arbitrary shell command)
  - `ALTER SYSTEM` (change conf setting with SQL)
  - Create/execute any function
  - . . . others . . .
- Multiplex unprivileged users with pooled connection

CRUNCHY Enterprise PostgreSQL

Superuser Abuse - Bypass All DAC

```
SET SESSION AUTHORIZATION bob;
CREATE TABLE foo(id int);
INSERT INTO foo VALUES(42);
REVOKE ALL ON foo FROM public, postgres;

RESET SESSION AUTHORIZATION;
SELECT CURRENT_USER, * FROM foo;
 current_user | id
--------------+----
 postgres     | 42
```

Superuser Abuse - Bypass RLS

```
SET SESSION AUTHORIZATION bob;
ALTER TABLE foo ENABLE ROW LEVEL SECURITY;
ALTER TABLE foo FORCE ROW LEVEL SECURITY;
CREATE POLICY p1 ON foo USING (id != 42);
SELECT CURRENT_USER, * FROM foo;
 current_user | id
--------------+----
(0 rows)

RESET SESSION AUTHORIZATION;
SELECT CURRENT_USER, * FROM foo;
 current_user | id
--------------+----
 postgres     | 42
```

Superuser Abuse - Load Any library

```
CREATE FUNCTION timed_sys_exec (text)
RETURNS text
AS '$libdir/pgiftest','timed_sys_exec'
LANGUAGE C STRICT;

SELECT timed_sys_exec('echo "hello world" > tse.txt');
      timed_sys_exec
---------------------------
 duration=0.029718:stdout=

SELECT timed_sys_exec('cat tse.txt');
            timed_sys_exec
---------------------------------------
 duration=0.001130:stdout=hello world
```

Superuser Abuse - COPY PROGRAM

```
COPY foo TO PROGRAM 'echo "local all all trust" > /tmp/pg_hba.conf';
\q

# cat /tmp/pg_hba.conf
local all all trust
```

Superuser Abuse - ALTER SYSTEM

```
ALTER SYSTEM SET hba_file = '/tmp/pg_hba.conf';

-- wait for or initiate restart
SHOW hba_file;
     hba_file
------------------
 /tmp/pg_hba.conf
```

Superuser Abuse - Create/Execute Any Function

```
-- PostgreSQL 9.6 and earlier only
CREATE FUNCTION pgsystem(cstring) RETURNS integer
    LANGUAGE c IMMUTABLE STRICT LEAKPROOF
    AS '$libdir/plpgsql', 'system';

select pgsystem('echo "hello world" > suabuse.txt'::cstring);
 system
--------
      0
\q

# cd $PGDATA
# cat suabuse.txt
hello world
```

Superuser Abuse - Create/execute Any Function

```
-- PostgreSQL 9.6 and earlier only
CREATE FUNCTION malloc(integer) RETURNS bigint
    LANGUAGE c IMMUTABLE STRICT LEAKPROOF
    AS '$libdir/plpgsql', 'malloc';

CREATE FUNCTION strdup(bigint) RETURNS cstring
    LANGUAGE c IMMUTABLE STRICT LEAKPROOF
    AS '$libdir/plpgsql', 'strdup';

CREATE FUNCTION open(cstring, integer) RETURNS integer
    LANGUAGE c IMMUTABLE STRICT LEAKPROOF
    AS '$libdir/plpgsql', 'open';

CREATE FUNCTION read(integer, bigint, integer) RETURNS integer
    LANGUAGE c IMMUTABLE STRICT LEAKPROOF
    AS '$libdir/plpgsql', 'read';
```

Superuser Abuse - Create/execute Any Function

```
-- PostgreSQL 9.6 and earlier only
WITH
  buf(d) AS (select malloc(3312)),
  rd(l)  AS (select read(open('/etc/passwd'::cstring, 0),
                         buf.d,
                         3311) from buf)
SELECT rd.l AS size, left(strdup(buf.d)::text,188) AS first_few_lines
FROM rd, buf;
 size |                first_few_lines
------+-------------------------------------------------
 3311 | root:x:0:0:root:/root:/bin/bash                 +
      | daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin+
      | bin:x:2:2:bin:/bin:/usr/sbin/nologin            +
      | sys:x:3:3:sys:/dev:/usr/sbin/nologin            +
      | sync:x:4:65534:sync:/bin:/bin/sync
(1 row)
```

Logging

- Error logging and reporting
  - stderr, csvlog, eventlog (Windows only), and syslog
  - Many options: where, when, what
  - Remote via syslog, ship with logstash/beats
- pgaudit extension
  - More granular, resists obfuscation
  - Session (coarse grained) or object (fine grained)
  - More detail in Appendix
- sepgsql extension
  - Generally denials are logged to audit.log
  - Tunable with custom policy

CRUNCHY Enterprise PostgreSQL

History

- Trigger based history
  - Event triggers - DDL
  - DML triggers - JSON, hstore, timestamp range datatype
  - Audit trigger example
- RLS Policy
  - Use timestamp range datatype
  - Policy makes only current version visible by default
  - Old versions saved via trigger
  - Partitioning keeps current and old row separated
  - Example in Appendix

CRUNCHY Enterprise PostgreSQL

Terms

- CIS: Center for Internet Security
  - Non-profit entity that produces recognized best practices for securing IT systems and data
- CIS Benchmark
  - Configuration guidelines
- DISA: Defense Information Systems Agency
  - Agency assures globally accessible enterprise information infrastructure
- STIG: Security Technical Implementation Guide
  - Configuration standards for DOD systems
  - Contain technical guidance to "lock down" systems/software

CRUNCHY Enterprise PostgreSQL

PostgreSQL CIS Benchmark

- PostgreSQL 9.X on RHEL CIS Benchmark - DRAFT
  - Prescriptive guidance for establishing a secure configuration posture for open source PostgreSQL
  - Tested on CentOS 6
  - PostgreSQL 9.5+
  - All Open Source Components
  - Two profiles
    - Level 1: practical and prudent with clear security benefit but not inhibit utility of the technology
    - Level 2: extends level 1 with defense in depth, but may inhibit utility or performance

CRUNCHY Enterprise PostgreSQL

CIS Benchmark - Coverage Summary

- Contains the following sections:
  - Installation and Patches
  - Directory and File Permissions
  - Logging Monitoring And Auditing (Centos 6)
  - User Access and Authorization
  - Connection and Login
  - PostgreSQL Settings
  - Replication
  - Special Configuration Considerations

CRUNCHY Enterprise PostgreSQL

CIS Benchmark - Anatomy of a Rule

- Scoring Status
- Applicable Profiles
- Description
- Audit Procedure
- Remediation Procedure
- CIS Controls
- References
- Notes

PostgreSQL STIG

- PostgreSQL 9.X on RHEL Security Technical Implementation Guide (STIG)
  - Comprehensive guide for config and ops of open source PostgreSQL
  - Based on the Database SRG (Security Requirements Guide)
  - Derives controls from NIST SP 800-53
  - RHEL 7.X with FIPS 140-2 crypto enabled
  - PostgreSQL 9.5+
  - 111 Rules
  - All Open Source Components

CRUNCHY Enterprise PostgreSQL

STIG - Coverage Summary

- Provides guidance to address requirements associated with:
  - Auditing
  - Logging
  - Data Encryption at Rest
  - Data Encryption Over the Wire
  - Access Controls
  - Administration
  - Authentication
  - Protecting against SQL Injection

CRUNCHY Enterprise PostgreSQL

STIG - Appendixes

- Instructions and code samples, assists with implementation of Fixes
- Verify applicability and tailor it as necessary
- Covered:
  - Sample account lockout script
  - pgaudit installation and configuration
  - General logging and remote logging configuration
  - RLS use example SQL script
  - pgcrypto installation
  - Finding and checking PGDATA
  - SSL configuration guide (detailed)

STIG - Anatomy of a Rule

- General Info: title, identifiers, severity category
- Discussion: describes security issue under consideration
- Check: how to check compliance with the rule
- Fix: how to remediate for compliance with the rule
- References: related CCI and NIST standards

STIG - Viewer

- STIG Viewer:
  https://iase.disa.mil/stigs/Pages/stig-viewing-guidance.aspx
- `java -jar STIGViewer-2.5.4.jar`

STIG - Checker

- STIG Checker
  https://github.com/CrunchyData/pgstigcheck-inspec
- Open source
- Work in progress
- Uses chef/inspec
- Others planned, e.g. CIS Benchmark, NIST
- Also WIP – continuous compliance checker

CRUNCHY Enterprise PostgreSQL

Example CIS Benchmark Rule - General Information

Ensure excessive administrative privileges are revoked

Scoring Status
Not Scored

Applicable Profiles
Level 1 - PostgreSQL on Linux

Example CIS Benchmark Rule - Description

With respect to PostgreSQL administrative SQL commands, only superusers should have elevated privileges. PostgreSQL regular or application users should not possess the ability to create roles, create new databases, manage replication, or perform any other action deemed privileged for a superuser account. Typically, regular users should only be granted the minimal set of privileges commensurate with managing the application:

- DDL (create table, create view, create index, etc.)
- DML (select, insert, update, delete)

Rationale Statement

By not restricting global administrative commands to superusers only, regular users granted excessive privileges may execute administrative commands with unintended and undesirable results.

Example CIS Benchmark Rule - Audit Procedure

First, inspect the privileges granted to the database superuser (identified here as postgres) using the display command `psql -c "\du postgres"` to establish a baseline for granted administrative privileges. Based on the output below, the postgres superuser can create roles, create databases, manage replication, and bypass row level security:

```
$ psql -c "\du postgres"
                                 List of roles
Role name |                         Attributes
----------+------------------------------------------------------------
postgres  | Superuser, Create role, Create DB, Replication,
          | Bypass RLS
```

Example CIS Benchmark Rule - Audit Procedure

Now, let's inspect the same information for a mock regular user called appuser using the display command `psql -c "\du appuser"`. The output confirms that regular user appuser has the same elevated privileges as system administrator user postgres. This is a finding.

```
$ psql -c "\du appuser"
                             List of roles
Role name |                    Attributes
----------+-------------------------------------------------
appuser   | Superuser, Create role, Create DB, Replication,
          | Bypass RLS
```

While this example demonstrated excessive administrative privileges granted to a single user, a comprehensive audit should be conducted to inspect all database users for excessive administrative privileges. This can be accomplished via either of the commands below.

```
$ psql -c "\du *"
$ psql -c "select * from pg_user order by usename"
```

Example CIS Benchmark Rule - Remediation Procedure

If any regular or application users have been granted excessive administrative rights, those privileges should be removed immediately via the PostgreSQL ALTER ROLE SQL command. Using the same example above, the following SQL statements revoke all unnecessary elevated administrative privileges from the regular user appuser:

```
$ psql -c "ALTER ROLE appuser NOSUPERUSER;"
ALTER ROLE
$ psql -c "ALTER ROLE appuser NOCREATEROLE;"
ALTER ROLE
$ psql -c "ALTER ROLE appuser NOCREATEDB;"
ALTER ROLE
$ psql -c "ALTER ROLE appuser NOREPLICATION;"
ALTER ROLE
$ psql -c "ALTER ROLE appuser NOBYPASSRLS;"
ALTER ROLE
$ psql -c "ALTER ROLE appuser NOINHERIT;"
ALTER ROLE
```

CRUNCHY Enterprise PostgreSQL

Example CIS Benchmark Rule - Remediation Procedure

Verify the appuser now passes your check by having no defined Attributes:

```
$ psql -c "\du appuser"
          List of roles
Role name | Attributes | Member of
----------+------------+-----------
appuser   |            |
```

Example CIS Benchmark Rule - Controls and Refs

CIS Controls

Version 6
5.1: Minimize And Sparingly Use Administrative Privileges
Version 7

References

https://www.postgresql.org/docs/current/static/sql-revoke.html
https://www.postgresql.org/docs/current/static/sql-createrole.html
https://www.postgresql.org/docs/current/static/sql-alterrole.html

CRUNCHY Enterprise PostgreSQL

Example STIG Rule - General Information

Rule Title: PostgreSQL must be configured to prohibit or restrict the use of organization-defined functions, ports, protocols, and/or services, as defined in the PPSM CAL and vulnerability assessments.
STIG ID: PGS9-00-000100
Rule ID: SV-87493r1_rule
Vuln ID: V-72841
Severity: CAT II
Class: Unclass

Example STIG Rule - Discussion

In order to prevent unauthorized connection of devices, unauthorized transfer of information, or unauthorized tunneling (i.e., embedding of data types within data types), organizations must disable or restrict unused or unnecessary physical and logical ports/protocols/services on information systems.

Applications are capable of providing a wide variety of functions and services. Some of the functions and services provided by default may not be necessary to support essential organizational operations. Additionally, it is sometimes convenient to provide multiple services from a single component (e.g., email and web services); however, doing so increases risk over limiting the services provided by any one component.

. . .

Example STIG Rule - Check

As the database administrator, run the following SQL:

```
$ psql -c "SHOW port"
```

If the currently defined port configuration is deemed prohibited, this is a finding.

Example STIG Rule - Fix

Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To change the listening port of the database, as the database administrator, change the following setting in postgresql.conf:

```
$ sudo su - postgres
$ vi $PGDATA/postgresql.conf
```

Change the port parameter to the desired port.

Next, restart the database:

```
# SYSTEMD SERVER ONLY
$ systemctl restart postgresql-9.5
# INITD SERVER ONLY
$ service postgresql-9.5 restart
```

Example STIG Rule - Fix

Note: psql uses the default port 5432 by default. This can be changed by specifying the port with psql or by setting the PGPORT environment variable:

```
$ psql -p 5432 -c "SHOW work_mem"
$ export PGPORT=5432
```

Example STIG Rule - References

CCI: CCI-000382
The organization configures the information system to prohibit or restrict the use of organization defined functions, ports, protocols, and/or services.
NIST SP 800-53 :: CM-7
NIST SP 800-53A :: CM-7.1 (iii)
NIST SP 800-53 Revision 4 :: CM-7 b

CCI: CCI-001762
The organization disables organization-defined functions, ports, protocols, and services within the information system deemed to be unnecessary and/or nonsecure.
NIST SP 800-53 Revision 4 :: CM-7 (1) (b)

STIG - Related Configs

```
shared_preload_libraries = 'pgaudit'

pgaudit.log = 'all, -misc'
pgaudit.log_catalog = on
pgaudit.log_level = 'log'
pgaudit.log_parameter = on
pgaudit.log_relation = on
pgaudit.log_statement_once = off
pgaudit.role = 'auditor'

log_connections = on
log_disconnections = on
log_error_verbosity = default
log_line_prefix = '%m %a %u %d %r %p %s %c %e: '
log_file_mode = 0600
```

STIG - Related Configs

```
log_destination = 'syslog'
syslog_facility = 'LOCAL0'
syslog_ident = 'postgres'
client_min_messages = error

log_timezone = 'UTC'
password_encryption = on

ssl = on
ssl_ca_file = '/some/protected/directory/root.crt'
ssl_crl_file = '/some/protected/directory/root.crl'
ssl_cert_file = '/some/protected/directory/server.crt'
ssl_key_file = '/some/protected/directory/server.key'
```

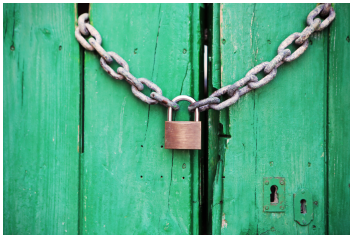
CRUNCHY Enterprise PostgreSQL

STIG - Related Configs

```
# Set these parameters to organizational requirements:
port = 5432
max_connections = N
statement_timeout = X
tcp_keepalives_idle = Y
tcp_keepalives_interval = Z
tcp_keepalives_count = Q
```

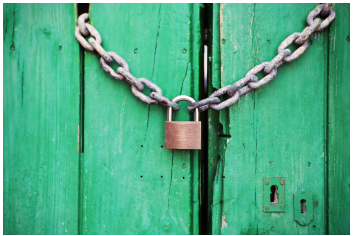
STIG - Related HBA settings

- An auth-method of "password" is explicitly forbidden
- Although auth-method "md5" not explicitly banned, FIPS 140-2 compliance blocks it
- Use auth-method of cert, gss, sspi, or ldap unless justified and approved
- PostgreSQL 10+ use SCRAM
- Every role must have unique authentication requirements
- LOGIN roles must not be shared



STIG - Related HBA settings

- With auth-method cert:
  - hostssl entries must contain clientcert=1
  - User mapping must be used as appropriate
  - CRL file must exist and be used



```
# pg_hba.conf example rule
hostssl all bob samenet cert clientcert=1 map=ssl-test

# example client command
psql "postgresql://<HOSTNAME>:<PORT>/postgres?sslmode=verify-full" -U bob
```

CRUNCHY Enterprise PostgreSQL

Questions?

Thank You!
mail@joeconway.com

set_user - Concept

- GRANT EXECUTE on `set_user()` and/or `set_user_u()` to otherwise unprivileged users
- Can switch the effective user when needed to perform specific actions
- Optional enhanced logging ensures an audit trail
- Once one or more unprivileged users able to run `set_user_u()`, ALTER superuser to NOLOGIN
- Multiplex unprivileged users, e.g. with connection pools

CRUNCHY Enterprise PostgreSQL

set_user - Overview

- When an allowed user executes `set_user('rolename')` or `set_user_u('rolename')`, several actions occur:
  - Current effective user becomes rolename
  - Role transition is logged, with specific notation if rolename is a superuser
  - Optionally `ALTER SYSTEM` commands will be blocked
  - Optionally `COPY PROGRAM` commands will be blocked
  - Optionally `SET log_statement` and variations will be blocked
  - If `set_user.block_log_statement = on` and rolename is a database superuser, current `log_statement` setting is changed to "all", meaning every SQL statement executed

set_user - Overview

- `reset_user()` function executed to restore the original user
- At that point, these actions occur:
  - Role transition is logged
  - `log_statement` setting is set to its original value
  - Blocked command behaviors return to normal

set_user - Superuser Escalation

- EXECUTE permission on `set_user_u('rolename')` required
- `set_user.superuser_whitelist` provides additional filter
- If `set_user.superuser_whitelist = ''`, escalation is blocked
- If `set_user.superuser_whitelist = '*'`, escalation is unfiltered
- Default is `set_user.superuser_whitelist = '*'`
- Combination of DAC (`GRANT EXECUTE ...`) and configuration (`set_user.superuser_whitelist`) allows two person control

set_user - Unprivileged Multiplexing

- EXECUTE permission on `set_user('rolename')` or `set_user('rolename','token')` required
- `set_user('rolename','token')`: token stored in session lifetime memory
  - `reset_user('token')` must be called instead of `reset_user()`
  - Provided token is compared with the stored token
  - If tokens do not match, or if a token was provided to `set_user` but not `reset_user`, ERROR occurs.

pgaudit

- Repository: https://github.com/pgaudit/pgaudit
- Provides detailed session and/or object audit logging
- Uses standard PostgreSQL logging facility
- Goal is to produce audit logs required for compliance
- Supports session and object level logging

Session Logging

- `read`: SELECT and COPY when the source is a relation or a query
- `write`: INSERT, UPDATE, DELETE, TRUNCATE, and COPY when the destination is a relation
- `function`: Function calls and DO blocks
- `role`: Statements related to roles and privileges: GRANT, REVOKE, CREATE/ALTER/DROP ROLE
- `ddl`: All DDL that is not included in the ROLE class
- `misc`: Miscellaneous commands, e.g. DISCARD, FETCH, CHECKPOINT, VACUUM
- `all`: All statements

CRUNCHY Enterprise PostgreSQL

Session Logging

```
-- Example:
-- Enable session logging for all DML and DDL
set pgaudit.log = 'write, ddl';

-- Enable session logging for all commands except MISC
set pgaudit.log = 'all, -misc';
```

Object Logging

- Logs statements affecting particular relation
- Only SELECT, INSERT, UPDATE and DELETE commands are supported
- Intended to be a finer-grained replacement for `pgaudit.log = 'read, write'`
- Implemented via the roles system (`pgaudit.role` setting)
- Relation (TABLE, VIEW, etc.) audit logged when audit role has (or inherits) permissions for the executed command

Object Logging

```
-- Example:
-- Set pgaudit.role to auditor and grant SELECT and DELETE privileges
-- on the account table. Any SELECT or DELETE statements on the account
-- table will now be logged:

set pgaudit.role = 'auditor';

grant select, delete
   on public.account
   to auditor;
```

Installation and Settings

- See U_PostgreSQL_9-x_V1R1_Supplemental.pdf
- Section 2.2. Appendix B for installation and configuration
- STIG configuration:

```
shared_preload_libraries = 'pgaudit'
pgaudit.log = 'all, -misc'
pgaudit.log_catalog = on
pgaudit.log_level = 'log'
pgaudit.log_parameter = on
pgaudit.log_relation = on
pgaudit.log_statement_once = off
pgaudit.role = 'auditor'
```

RLS Timetravel

- Possible alternative to "history" tables
- Use RLS to filter based on point in time
- Use PG10 partitioning to keep history separate from current

RLS Timetravel

```
CREATE TABLE timetravel
(
  id int8,
  f1 text not null,
  tr tstzrange not null default tstzrange(clock_timestamp(), 'infinity', '[]')
) PARTITION BY RANGE (upper(tr));

CREATE TABLE timetravel_current PARTITION OF timetravel
(
  primary key (id, tr) DEFERRABLE
) FOR VALUES FROM ('infinity') TO (MAXVALUE);
CREATE INDEX timetravel_current_tr_idx ON timetravel_current USING GIST (tr);

CREATE TABLE timetravel_history PARTITION OF timetravel
(
  primary key (id, tr) DEFERRABLE
) FOR VALUES FROM (MINVALUE) TO ('infinity');
CREATE INDEX timetravel_history_tr_idx ON timetravel_history USING GIST (tr);
```

RLS Timetravel

```
CREATE OR REPLACE FUNCTION get_pit() RETURNS timestamptz AS $$
  SELECT
  CASE WHEN current_setting('tt.cts', true) IS NULL OR
            current_setting('tt.cts', true) = '' THEN
    clock_timestamp()
  ELSE
    current_setting('tt.cts', true)::timestamptz
  END
$$ LANGUAGE sql;

-- only current rows are visible
-- unless tt.cts is defined, in which case get rows current as of that time
ALTER TABLE timetravel ENABLE ROW LEVEL SECURITY;
CREATE POLICY p1 ON timetravel
USING (tr @> get_pit())
WITH CHECK (/* NEW */tr @> clock_timestamp() OR
            /* OLD */ upper(tr) <= clock_timestamp());
```

RLS Timetravel

```
CREATE OR REPLACE FUNCTION modify_timetravel()
RETURNS TRIGGER AS $$
  DECLARE
    ctr timestamptz := clock_timestamp();
  BEGIN
    OLD.tr = tstzrange(lower(OLD.tr), ctr, '[)');
    INSERT INTO timetravel VALUES (OLD.*);
    IF (TG_OP = 'UPDATE') THEN
      NEW.tr = tstzrange(ctr, 'infinity', '[]');
      RETURN NEW;
    ELSIF (TG_OP = 'DELETE') THEN
      RETURN OLD;
    END IF;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER timetravel_audit BEFORE DELETE OR UPDATE
ON timetravel_current FOR EACH ROW EXECUTE PROCEDURE modify_timetravel();
```

RLS Timetravel

```
GRANT ALL ON timetravel TO dba;
SET SESSION AUTHORIZATION dba;

INSERT INTO timetravel(id, f1)
SELECT g.i, 'row-' || g.i::text
FROM generate_series(1,1000000) AS g(i);

RESET SESSION AUTHORIZATION;
VACUUM FREEZE ANALYZE timetravel;
SET SESSION AUTHORIZATION dba;
```

RLS Timetravel

```
-- NOTE: Point In Time (pit) is after update to id 42 and before delete of id 4242
UPDATE timetravel SET f1 = 'update number 1' WHERE id = 42 RETURNING lower(tr) AS pit \gset

DELETE FROM timetravel WHERE id = 4242;

SELECT * FROM timetravel WHERE id in (42, 4242);
 id |       f1        |                     tr
----+-----------------+-------------------------------------------
 42 | update number 1 | ["2017-09-03 16:44:22.160023-07",infinity]
```

RLS Timetravel

```
UPDATE timetravel SET f1 = 'update number 2' WHERE id = 42;
SELECT * FROM timetravel WHERE id = 42;
 id |       f1        |                      tr
----+-----------------+-----------------------------------------
 42 | update number 2 | ["2017-09-03 16:44:24.206489-07",infinity]

UPDATE timetravel SET f1 = 'update number 3' WHERE id = 42;
SELECT * FROM timetravel WHERE id = 42;
 id |       f1        |                      tr
----+-----------------+-----------------------------------------
 42 | update number 3 | ["2017-09-03 16:44:25.270849-07",infinity]
```

RLS Timetravel

```
SELECT set_config('tt.cts', :'pit', false);
         set_config
------------------------------
 2017-09-03 16:44:22.160023-07

SELECT * FROM timetravel WHERE id in (42, 4242) ORDER BY 1, 3;
  id  |       f1        |                                  tr
------+-----------------+-------------------------------------------------------------------
   42 | update number 1 | ["2017-09-03 16:44:22.160023-07","2017-09-03 16:44:24.206489-07")
 4242 | row-4242        | ["2017-09-03 16:44:04.406047-07","2017-09-03 16:44:23.173052-07")
```

RLS Timetravel

```
RESET SESSION AUTHORIZATION;

SELECT * FROM timetravel WHERE id in (42, 4242) ORDER BY 1, 3;
  id  |       f1        |                                  tr
------+-----------------+-----------------------------------------------------------------
   42 | row-42          | ["2017-09-03 16:44:04.34601-07","2017-09-03 16:44:22.160023-07")
   42 | update number 1 | ["2017-09-03 16:44:22.160023-07","2017-09-03 16:44:24.206489-07")
   42 | update number 2 | ["2017-09-03 16:44:24.206489-07","2017-09-03 16:44:25.270849-07")
   42 | update number 3 | ["2017-09-03 16:44:25.270849-07",infinity]
 4242 | row-4242        | ["2017-09-03 16:44:04.406047-07","2017-09-03 16:44:23.173052-07")
```