

Securing PostgreSQL – Exploring the PostgreSQL STIG and Beyond

Joe Conway

joe.conway@crunchydata.com

mail@joeconway.com

Crunchy Data

September 07, 2017



Securing PostgreSQL



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

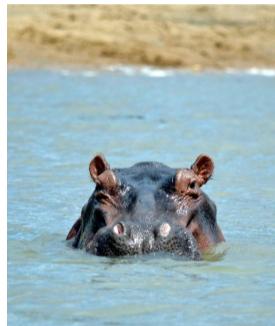
<http://www.postgresql.org>



<https://postgresql.us/>

Security

- International Recognition
 - Common Criteria, ISO/IEC 15408 (CC)
 - Security Technical Implementation Guide (STIG)
 - Center for Internet Security (CIS) Benchmark (Work in Progress)
- Features
 - Perimeter
 - Internal
 - Chronological



Security - Perimeter

- Operating System
 - OS Configuration
 - FIPS 140-2 compliance
 - STIG
 - 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



Security - Perimeter

- Client-server
 - Authentication
 - Host based authentication
 - Internal: md5*, SCRAM**, PAM, peer, SSL cert
 - External: GSSAPI, SSPI, LDAP, RADIUS
 - Encryption in transit
 - SSL



Security - Internal

- 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



Security - Internal

- 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



Security - Chronological

- 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
 - Enhanced logging for audit purposes
 - More granular
 - Resists obfuscation
 - Session-based (coarse grained)
 - Object-based (fine grained)



Security - Chronological

- 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 still exist
 - [Example in Appendix](#)



Terms

- Defense Information Systems Agency (DISA)
 - Combat Support Agency
 - Provides, operates, and assures:
 - command and control
 - information sharing capabilities
 - globally accessible enterprise information infrastructure



Terms

- Security Technical Implementation Guides (STIGs)
 - Configuration standards for DOD systems
 - Contain technical guidance to "lock down" systems/software



Terms

- Control Correlation Identifier (CCI)
 - Standard id and description for each of the singular, actionable statements that comprise an Information Assurance (IA) control or IA best practice
 - Allows traceability of security requirements from origin to low-level implementation



PostgreSQL STIG

- PostgreSQL 9.X on RHEL Security Technical Implementation Guide (STIG)
 - Offers security-conscious enterprises a comprehensive guide for configuration and operation of **open source** PostgreSQL
 - Based on the Database Security Requirements Guide (SRG) Version 2 Release 6
 - Derives controls from NIST Special Publication (SP) 800-53, Revision 4
 - Red Hat Enterprise Linux 7.X
 - PostgreSQL 9.5+
 - 111 Rules
 - All Open Source Components



PostgreSQL STIG

- 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



Appendixes

- Instructions and code samples
- Assist with the implementation of the Fixes in the main STIG document
- Useful in many PostgreSQL deployments
- Verify applicability and tailor it as necessary



Appendixes

- 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)



Notes

- Requires RHEL to enable FIPS 140-2 compliant cryptographic modules
- STIG is but one component of a robust defense-in-depth
 - Use along with the applicable operating system and network STIGs
 - Train users in importance of good security practices
- Rules, checks, fixes are generic
 - Verify each rule and examples applicable
 - Tailor as necessary for your environment



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



Severity Category Codes

- Measure of vulnerabilities used to assess security posture
- Rules assigned CAT I, II, or III
- With respect to Confidentiality, Availability, or Integrity (CAI):
 - CAT I: exploit directly and immediately results in loss of CAI
 - CAT II: exploit has potential to result in loss of CAI
 - CAT III: existence degrades measures to protect against loss of CAI



STIG Viewer

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



STIG Viewer - Demo

The screenshot shows the STIG Viewer application interface. On the left, there is a 'STIG Explorer' pane with a tree view showing 'STIGs' and a list of rules. Below it is a 'Filter Panel' with options for 'Include (+) Filter' and 'Exclude (-) Filter', and a search box for 'Keyword' and 'Filter'. The main pane displays a table of rules with columns 'Val ID' and 'Rule Name'. The selected rule, V-72841, is highlighted. On the right, the 'General Information' pane shows details for rule V-72841, including its title, rule ID, severity, and class. Below this, there are sections for 'Discussion', 'Check Content', and 'Fix Text'.

Val ID	Rule Name
V-72841	SWG-APP-000142-DB-000094
V-72843	SWG-APP-000099-DB-000043
V-72845	SWG-APP-000456-DB-000399
V-72847	SWG-APP-000119-DB-000092
V-72849	SWG-APP-000023-DB-000001
V-72851	SWG-APP-000266-DB-000162
V-72853	SWG-APP-000133-DB-000179
V-72855	SWG-APP-000133-DB-000179
V-72857	SWG-APP-000172-DB-000075
V-72859	SWG-APP-000073-DB-000084
V-72861	SWG-APP-000214-DB-000310
V-72863	SWG-APP-000001-DB-000031
V-72865	SWG-APP-000133-DB-000362
V-72867	SWG-APP-000189-DB-000115
V-72869	SWG-APP-000311-DB-000308
V-72871	SWG-APP-000251-DB-000149
V-72873	SWG-APP-000251-DB-000391
V-72875	SWG-APP-000251-DB-000392
V-72877	SWG-APP-000357-DB-000316
V-72883	SWG-APP-000328-DB-000301
V-72885	SWG-APP-000120-DB-000061
V-72887	SWG-APP-000374-DB-000322
V-72889	SWG-APP-000267-DB-000143
V-72891	SWG-APP-000290-DB-000065
V-72893	SWG-APP-000360-DB-000320
V-72895	SWG-APP-000442-DB-000379
V-72897	SWG-APP-000133-DB-000200
V-72899	SWG-APP-000133-DB-000198
V-72901	SWG-APP-000133-DB-000199
V-72903	SWG-APP-000102-DB-000044
V-72905	SWG-APP-000342-DB-000302
V-72907	SWG-APP-000447-DB-000293
V-72909	SWG-APP-000356-DB-000314
V-72911	SWG-APP-000233-DB-000124
V-72913	SWG-APP-000381-DB-000361
V-72915	SWG-APP-000119-DB-000079
V-72917	SWG-APP-000454-DB-000389
V-72919	SWG-APP-000494-DB-000344
V-72921	SWG-APP-000492-DB-000333

General Information

PostgreSQL 8.x Security Technical Implementation Guide - Release: 1 Benchmark Date: 20 Jun 2017

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 FPMN CAL and vulnerability assessments.

STIG ID: PGS9-00-000100 **Severity:** CAT II

Rule ID: SV-87493r1_rule **Class:** Unclass

Valn ID: V-72841

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.

Check Content

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.

Fix Text

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.

CCI

CCI: OCI400382
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 (U)
NIST SP 800-53 Revision 4 - CM-7 b

CCI: CCI401762
The organization disables organization defined functions, ports, protocols, and services within the information system deemed to be

STIG Checker

- STIG Checker
 - <https://github.com/CrunchyData/pgstigcheck-inspec>
- Open source
- Work in progress
- Uses [chef/inspec](#)
- Other versions planned, e.g. OpenSCAP



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



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.



Discussion

To support the requirements and principles of least functionality, the application must support the organizational requirements providing only essential capabilities and limiting the use of ports, protocols, and/or services to only those required, authorized, and approved to conduct official business or to address authorized quality of life issues.

Database Management Systems using ports, protocols, and services deemed unsafe are open to attack through those ports, protocols, and services. This can allow unauthorized access to the database and through the database to other components of the information system.



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.



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
```



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
```



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'
```



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
```



Other Security Related Configs

```
# related to Confidentiality, Availability,  
# or Integrity (CAI) -- not necessarily all inclusive
```

```
listen_addresses = 'localhost'
```

```
#data_directory = 'ConfigDir'
```

```
#hba_file = 'ConfigDir/pg_hba.conf'
```

```
#ident_file = 'ConfigDir/pg_ident.conf'
```

```
#superuser_reserved_connections = 3
```

```
#unix_socket_directories = '/tmp'
```

```
#unix_socket_group = ''
```

```
#unix_socket_permissions = 0777
```

```
#bonjour = off
```

```
#bonjour_name = ''
```



Other Security Related Configs

related to Confidentiality, Availability,
or Integrity (CAI) -- not necessarily all inclusive

```
#authentication_timeout = 1min
#db_user_namespace = off
#row_security = on
#krb_server_keyfile = ''
#krb_caseins_users = off
#max_prepared_transactions = 0
#work_mem = 4MB
#maintenance_work_mem = 64MB
#autovacuum_work_mem = -1
#autovacuum_max_workers = 3
#max_worker_processes = 8
#force_parallel_mode = off
```



Other Security Related Configs

```
# related to Confidentiality, Availability,  
# or Integrity (CAI) -- not necessarily all inclusive
```

```
#fsync = on  
#synchronous_commit = on  
#search_path = '"$user", public'  
#dynamic_library_path = '$libdir'  
#local_preload_libraries = ''  
#session_preload_libraries = ''  
#backslash_quote = safe_encoding  
#include_dir = 'conf.d'  
#include_if_exists = 'exists.conf'  
#include = 'special.conf'
```



Host Based Authentication File

- [pg_hba.conf](#)
- Which hosts are allowed to connect
- How clients are authenticated
- Which PostgreSQL user names they can use
- Which databases they can access



Host Based Authentication File

- Read on server startup
- Must reload postmaster for changes to take effect
- First line matching conn type, address, database, and user is used for authentication
- If line picked and authentication fails, access denied
- If no line matches, access denied



Host Based Authentication File

```
# CONN-TYPE  DATABASE  USER      ADDRESS    METHOD      OPTIONS
# local      <dbname>  <user>    <method>  [<opts>]
# host       <dbname>  <user>    <address> <method>  [<opts>]
# hostssl   <dbname>  <user>    <address> <method>  [<opts>]
# hostnossl <dbname>  <user>    <address> <method>  [<opts>]

# Default values on Debian-variants
local      all        postgres   peer
local      all        all        peer
host       all        all        127.0.0.1/32 md5
host       all        all        ::1/128     md5

# RHEL7
local      all        all        peer
host       all        all        127.0.0.1/32 ident
host       all        all        ::1/128     ident
```



Connection

- Specifies type of connection the rule matches
- local: Unix-domain socket
- host: Either plain or SSL-encrypted TCP/IP socket
- hostssl: SSL-encrypted TCP/IP socket
- hostnossl: Plain TCP/IP socket



Database

- Specifies set of databases the rule matches
- all: Wildcard
- sameuser: Database name matches user name
- samerole: User part of role/group matching database name
- replication: all keyword does not match replication
- <dbname>[,<dbname>]: One or more specific database names
- @<filename>: Separate file containing names to match



User

- Specifies set of users the rule matches
- all: Wildcard
- <username>[,<username>]: One or more user names
- +<groupname>: Any roles that are directly or indirectly members of this role
- @<filename>: Separate file containing names to match



Address

- Specifies set of client hosts the rule matches
- <IPAddr>/<CIDR-Mask>: Host or Network
- <IPAddr> <Mask>: Host or Network
- [.]<hostname>: [suffix] actual FQ hostname
- samehost: match any of server's own IP addresses
- samenet: match any address in any subnet that server directly connected to



Method

- Specifies **authentication method** to use when connection matches rule
- trust
- md5, password
- cert
- peer, pam, ident
- gss, sspi, ldap, radius
- reject



Options

- Set of options for the authentication in the format NAME=VALUE
- Options depend on authentication method
- Refer to "Client Authentication" section of docs



Brief Example

```
local all all trust
host all all 127.0.0.1/32 trust
host all all samenet md5
host all all 192.168.1.0/24 ldap ldapurl="ldap://ldap.ex.net/dc=ex,dc=net?uid?sub"
```



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
- 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 "postgres://<HOSTNAME>:<PORT>/postgres?sslmode=verify-full" -U bob
```

Questions?

Thank You!
mail@joeconway.com



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



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'
```



set_user

- `set_user` allows switching users and optionally privilege escalation with enhanced logging and control
- Useful when users must escalate to superuser or object owner roles to perform needed maintenance tasks
- Also useful to multiplex unprivileged users with common connection

Superuser abuse

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

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

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

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



Superuser abuse

```
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

```
with fname(d) as (select '/usr/local/pgsql-head/data/pg_hba.conf'::cstring),
    buf(d)      as (select malloc(261)),
    rd(l)       as (select read(open(strdup(memcpy(malloc(strlen(fname.d) + 1),
                                                fname.d,
                                                strlen(fname.d))), 0), buf.d, 260)

    from buf, fname
) select rd.l, strdup(buf.d) from rd, buf;
 1 |                                strdup
-----+-----
260 | # PostgreSQL Client Authentication Configuration File          +
    | # =====                                                    +
    | #                                                            +
    | # Refer to the "Client Authentication" section in the PostgreSQL +
    | # documentation for a complete description of this file.  A short+
    | # synopsis follows.
```

(1 row)



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

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



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

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



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.

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 timestampz AS $$
  SELECT
    CASE WHEN current_setting('tt.cts', true) IS NULL OR
          current_setting('tt.cts', true) = '' THEN
      clock_timestamp()
    ELSE
      current_setting('tt.cts', true)::timestampz
    END
$$ 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")

