PostgreSQL set_user Extension Module

Contents

PostgreSQL set_user Extension Module

Syntax

```
set_user(text rolename) returns text
set_user(text rolename, text token) returns text
set_user_u(text rolename) returns text
reset_user() returns text
reset_user(text token) returns text
```

Inputs

`rolename` is the role to be transitioned to. `token` if provided during set_user is saved, and then required to be provided again for reset.

Configuration Options

- Add `set_user` to `shared_preload_libraries` in postgresql.conf.
- Optionally, the following custom parameters may be set to control their respective commands:
- set_user.block_alter_system = off (defaults to “on”)
- set_user.block_copy_program = off (defaults to “on”)
- set_user.block_log_statement = off (defaults to “on”)
- set_user.superuser_whitelist = `'<role list>'`
  - `<role list>` can contain any of the following:
  - list of user roles (i.e. `<role1>, <role2>,...,<roleN>`)
  - Group roles may be indicated by `+<roleN>`
  - The wildcard character `*`
- set_user.nosuperuser_target_whitelist = `'<role list>'`
  - `<role list>` can contain any of the following:
  - list of user roles (i.e. `<role1>, <role2>,...,<roleN>`)
  - Group roles may be indicated by `+<roleN>`
  - The wildcard character `*`
- To make use of the optional `set_user` and `reset_user` hooks, please refer to the hooks section.

Description

This PostgreSQL extension allows switching users and optional privilege escalation with enhanced logging and control. It provides an additional layer of logging and control when unprivileged users must escalate themselves to superuser or object owner roles in order to perform needed maintenance tasks. Specifically, when an allowed user executes `set_user(text)` or `set_user_u(text)`, several actions occur:

- The current effective user becomes `rolename`.
- The role transition is logged, with a specific notation if `rolename` is a superuser.
- `log_statement` setting is set to “all”, meaning every SQL statement executed while in this state will also get logged.
- If `set_user.block_alter_system` is set to “on”, `ALTER SYSTEM` commands will be blocked.
- If `set_user.block_copy_program` is set to “on”, `COPY PROGRAM` commands will be blocked.
- If `set_user.block_log_statement` is set to “on”, `SET log_statement` and variations will be blocked.
- If `set_user.block_log_statement` is set to “on” and `rolename` is a database superuser, the current `log_statement` setting is changed to “all”, meaning every SQL statement executed
- If `set_user.superuser_audit_tag` is set, the string value will be appended to `log_line_prefix` upon superuser escalation. All logs after superuser escalation will be tagged with the value of `set_user.superuser_audit_tag`. This value defaults to `'AUDIT'`.
- Post-execution hook for `set_user` is called if it is set.

Only users with `EXECUTE` permission on `set_user_u(text)` may escalate to superuser. Additionally, all rules in Superuser Whitelist apply to `set_user.superuser_whitelist` and `set_user_u(text)`.

Postgres roles calling `set_user(text)` can only transition to roles listed or included in `set_user.nosuperuser_target_whitelist` (defaults to all roles). Additionally the logic in Nosuperuser Whitelist applies to `current_user` when `set_user()` is invoked.

Additionally, with `set_user('rolename','token')` the `token` is stored for the lifetime of the session.

When finished with required actions as `rolename`, the `reset_user()` function is 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.
- Post-execution hook for `reset_user` is called if it is set.

If `set_user`, was provided with a `token`, then `reset_user('token')` must be called instead of `reset_user()`:

- The provided `token` is compared with the stored token.
- If the tokens do not match, or if a `token` was provided to `set_user` but not `reset_user`, an ERROR occurs.

set_user Usage

Typical use of the set_user extension is as follows:

GRANT EXECUTE to Functions In order to make use of the set_user functions, some database roles must be able to execute the functions. Allow these privileges by GRANTing EXECUTE on the appropriate functions to their intended users.

```
GRANT EXECUTE ON FUNCTION set_user(text) TO dbclient,dbclient2;
GRANT EXECUTE ON FUNCTION set_user(text, text) to dbclient,dbclient2;
GRANT EXECUTE ON FUNCTION set_user_u(text) TO dbadmin;
```

This example assumes that there are three users of set_user:

1) dbclient is an unprivileged user that can run as dbclient2 through calls to set_user.
2) dbclient2 is an unprivileged user that can run as dbclient through calls to set_user.
3) dbadmin is the privileged (non-superuser) role, which is able to escalate privileges to superuser with Enhanced Logging.

Call set_user to Transition Transitioning to other roles through use of set_user provides the ability to change the session's current_user.

Transitions can be made to unprivileged users through use of set_user (with optional token, as described above).

```
SELECT set_user('dbclient2');
```

Alternatively, transitions can be made to superusers through use of set_user_u:

```
SELECT set_user_u('postgres');
```

Note: See rules in Superuser Whitelist for logic around calling set_user_u(text). See Nosuperuser Whitelist for reference logic around calling set_user(text).

Once one or more unprivileged users are able to run set_user_u() in order to escalate their privileges, the superuser account (typically postgres) can be altered to NOLOGIN, preventing any direct database connection by a superuser which would bypass the enhanced logging.

Naturally for this to work as expected, the PostgreSQL cluster must be audited to ensure there are no other PostgreSQL roles existing which are both superuser and can log in. Additionally there must be no unprivileged PostgreSQL roles which have been granted access to one of the existing superuser roles.

set_user.superuser_whitelist Rules and Logic The following rules govern escalation to superuser via the set_user_u(text) function:

- current_user must be GRANTed EXECUTE ON FUNCTION set_user_u(text) OR current_user must be the OWNER of the set_user_u(text) function OR current_user must be a superuser.
- current_user must be listed in set_user.superuser_whitelist OR current_user must belong to a group that is listed in set_user.superuser_whitelist (e.g. '+admin')
- If set_user.superuser_whitelist is the empty set , '', superuser escalation is blocked for all users.
- If set_user.superuser_whitelist is the wildcard character, '*', all users with EXECUTE permission on set_user_u(text) can escalate to superuser.
- If set_user.superuser_whitelist is not specified, the value defaults to the wildcard character, '*'.

set_user.nosuperuser_target_whitelist Rules and Logic The following rules govern non-superuser role transitions through use of set_user(text) or set_user(text, text) function (for simplicity, only set_user(text) is used):

- current_user must be GRANTed EXECUTE ON FUNCTION set_user(text) OR current_user must be the OWNER of the set_user(text) function OR current_user must be a superuser.
- The target rolename must be listed in set_user.nosuperuser_target_whitelist OR the target rolename must belong to a group that is listed in set_user.nosuperuser_target_whitelist (e.g. '+client')
- If set_user.nosuperuser_target_whitelist is the empty set , '', set_user(text) transitions to non-superusers are blocked for all users.
- If set_user.nosuperuser_target_whitelist is the wildcard character, '*', all users with EXECUTE permission on set_user(text) can transition to any other non-superuser role.
- If set_user.nosuperuser_target_whitelist is not specified, the value defaults to the wildcard character, '*'.

Perform Actions With Enhanced Logging Once a transition has been made, the current session behaves as if it has the privileges of the new `current_user`. The optional enhanced logging creates an audit trail upon transition to an alternate role, ensuring that any privilege escalation/alteration does not go unmonitored.

This audit trail is tagged with the value of `set_user.superuser_audit_tag`, such that actions after superuser escalation are easily identifiable.

```
SELECT reset_user();
```

If `set_user()` was initially called with a `token`, the same `token` must be provided in order to reset back to the previous user.

```
SELECT set_user('dbclient2', 'some_token_string');
SELECT reset_user('some_token_string');
```

Blocking `ALTER SYSTEM` and `COPY PROGRAM`

Note that for the blocking of `ALTER SYSTEM` and `COPY PROGRAM` to work properly, you must include `set_user` in `shared_preload_libraries` in `postgresql.conf` and restart PostgreSQL.

Notes:

If set_user.block_log_statement is set to "off", the `log_statement` setting is left unchanged.

For the blocking of `ALTER SYSTEM` and `COPY PROGRAM` to work properly, you must include `set_user` in shared_preload_libraries in postgresql.conf and restart PostgreSQL.

Neither `set_user(text)` nor `set_user_u(text)` may be executed from within an explicit transaction block.

Caveats

In its current state, this extension cannot prevent `rolename` from performing a variety of nefarious or otherwise undesireable actions. However, these actions will be logged providing an audit trail, which could also be used to trigger alerts.

Although this extension compiles and works with all supported versions of PostgreSQL starting with PostgreSQL 9.1, all features are not supported until PostgreSQL 9.4 or higher. The ALTER SYSTEM command does not exist prior to 9.4 and COPY PROGRAM does not exist prior to 9.3.

Post-Execution Hooks

`set_user` exposes two hooks that may be used to control post-execution behavior for `set_user` and `reset_user`.

Description

The following hooks are called (if set) directly before returning from successful calls to `set_user` and `reset_user`. These hooks are meant to give other extensions awareness of `set_user` actions. This is helpful, for instance, to keep track of dynamic user switching within a session.

To avoid order-dependency in `shared_preload_libraries`, these hooks are registered in the rendezvous hash table of core Postgres. The header defines a utility function for doing all of the necessary setup.

`post_set_user` hook

Allows another extension to take action after calls to `set_user`. This hook takes the username as an argument so that the hook implementation is aware of the username.

`post_reset_user` hook

Allows another extension to take action after calls to `reset_user`. This hook does not take any arguments, since the resulting username will always be the `session_user`.

Configuration

Follow the instructions below to implement `set_user` and `reset_user` post-execution hooks in another extension:

- Add '-I$(includedir)' to `CPPFLAGS` of the extension which implements the post-execution hooks.
- `#include set_user.h` in whichever file implements the hooks.
- Register hook implementations in `rendezvous_variable` hash using the `register_set_user_hooks` utility function.

Configuration is described in more detail in the post-execution hooks subsection of the Install documentation.

Caveats

If another extension implements the post-execution hooks, `post_set_user_hook` and `post_reset_user_hook`, `set_user` must be listed before that extension in `shared_preload_libraries`. This is due to the way `shared_preload_libraries` are opened and loaded into memory by Postgres: the hooks need to be loaded into memory before their implementations can access them.

Installation

Requirements

- PostgreSQL 9.1 or higher.

Compile and Install

Clone PostgreSQL repository:

```
$> git clone https://github.com/postgres/postgres.git
```

Checkout REL9_5_STABLE (for example) branch:

```
$> git checkout REL9_5_STABLE
```

Make PostgreSQL:

```
$> ./configure
$> make install -s
```

Change to the contrib directory:

```
$> cd contrib
```

Clone `set_user` extension:

```
$> git clone https://github.com/pgaudit/set_user
```

Change to `set_user` directory:

```
$> cd set_user
```

Build `set_user`:

```
$> make
```

Install `set_user`:

```
$> make install
```

Using PGXS If an instance of PostgreSQL is already installed, then PGXS can be utilized to build and install `set_user`. Ensure that PostgreSQL binaries are available via the `$PATH` environment variable then use the following commands.

```
$> make USE_PGXS=1
$> make USE_PGXS=1 install
```

Configure

The following bash commands should configure your system to utilize `set_user`. Replace all paths as appropriate. It may be prudent to visually inspect the files afterward to ensure the changes took place.

Initialize PostgreSQL (if needed):

```
$> initdb -D /path/to/data/directory
```

Create Target Database (if needed):

```
$> createdb <database>
```

Install `set_user` functions:

Edit postgresql.conf and add `set_user` to the `shared_preload_libraries` line, optionally also changing custom settings as mentioned above.

First edit postgresql.conf in your favorite editor:

```
$> vi $PGDATA/postgresql.conf
```

Then add these lines to the end of the file:

```
# Add set_user to any existing list
shared_preload_libraries = 'set_user'
# The following lines are only required to modify the
# blocking of each respective command if desired
set_user.block_alter_system = off       #defaults to "on"
set_user.block_copy_program = off       #defaults to "on"
set_user.block_log_statement = off      #defaults to "on"
set_user.superuser_whitelist = ''       #defaults to '*'
set_user.nosuperuser_target_whitelist = ''       #defaults to '*'
```

Finally, restart PostgreSQL (method may vary):

```
$> service postgresql restart
```

Install the extension into your database:

```
psql <database>
CREATE EXTENSION set_user;
```

Install `set_user` post-execution hooks:

Ensure that `set_user.h` is copied to `$(includedir)`.

This can be done automatically upon normal installation:

```
$> make USE_PGXS=1 install
```

There is also an explicit make target available to copy the header file to the appropriate directory:

```
$> make USE_PGXS=1 install-headers
```

Ensure that the implementing extension adds `-I$(includedir)` to `CPPFLAGS` in its Makefile:

```
# Add -I$(includedir) to CPPFLAGS so the set_user header is included
override CPPFLAGS += -I$(includedir)
```

Ensure that the implementing extension includes the `set_user` header file in the appropriate C file:

```
/* Include set_user hooks in whichever C file implements the hooks */
#include "set_user.h"
```

Create your `set_user` hooks and register them in the rendezvous_variable hash:

```
void _PG_Init(void)
{
    /*
     * Your _PG_Init code here
     */

     register_set_user_hooks(extension_post_set_user, extension_post_reset_user);

    /*
     * more _PG_Init code
     */
}

/*
 * extension_post_set_user
 *
 * Entrypoint of the set_user post-exec hook.
 */
static void
extension_post_set_user(void)
{
    /* Some magic */
}

/*
 * extension_post_reset_user
 *
 * Entrypoint of the reset_user post-exec hook.
 */
static void
extension_post_reset_user(void)
{
    /* Some magic */
}
```

GUC Parameters

- Block `ALTER SYSTEM` commands
- `set_user.block_alter_system = on`
- Block `COPY PROGRAM` commands
- `set_user.block_copy_program = on`
- Block `SET log_statement` commands
- `set_user.block_log_statement = on`
- Allow list of roles to escalate to superuser
- `set_user.superuser_whitelist = '<role1>,<role2>,...,<roleN>'`
- Allowed list of roles that can be switched to (not used in set_user_u)
- `set_user.nosuperuser_target_whitelist = '<role1>,<role2>,...,<roleN>'`

Examples

```
#################################
# OS command line, terminal 1
#################################
psql -U postgres <dbname>

---------------------------------
-- psql command line, terminal 1
---------------------------------
SELECT rolname FROM pg_authid WHERE rolsuper and rolcanlogin;
 rolname
----------
 postgres
```

```
(1 row)

CREATE EXTENSION set_user;
CREATE USER dba_user;
GRANT EXECUTE ON FUNCTION set_user(text) TO dba_user;
GRANT EXECUTE ON FUNCTION set_user_u(text) TO dba_user;

##################################
# OS command line, terminal 2
##################################
psql -U dba_user <dbname>

--------------------------------
-- psql command line, terminal 2
--------------------------------
SELECT set_user('postgres');
ERROR:  Switching to superuser only allowed for privileged procedure:
'set_user_u'
SELECT set_user_u('postgres');
SELECT CURRENT_USER, SESSION_USER;
 current_user | session_user
--------------+--------------
 postgres     | dba_user
(1 row)

SELECT reset_user();
SELECT CURRENT_USER, SESSION_USER;
 current_user | session_user
--------------+--------------
 dba_user     | dba_user
(1 row)

\q

--------------------------------
-- psql command line, terminal 1
--------------------------------
ALTER USER postgres NOLOGIN;
-- repeat terminal 2 test with dba_user before exiting
\q

##################################
# OS command line, terminal 1
##################################
tail -n 6 <postgres log>
LOG:  Role dba_user transitioning to Superuser Role postgres
STATEMENT:  SELECT set_user_u('postgres');
LOG:  statement: SELECT CURRENT_USER, SESSION_USER;
LOG:  statement: SELECT reset_user();
LOG:  Superuser Role postgres transitioning to Role dba_user
STATEMENT:  SELECT reset_user();

##################################
# OS command line, terminal 2
##################################
psql -U dba_user <dbname>

--------------------------------
-- psql command line, terminal 2
--------------------------------
-- Verify there are no superusers that can login directly
SELECT rolname FROM pg_authid WHERE rolsuper and rolcanlogin;
 rolname
---------
(0 rows)
```

```
-- Verify there are no unprivileged roles that can login directly
-- that are granted a superuser role even if it is multiple layers
-- removed
DROP VIEW IF EXISTS roletree;
CREATE OR REPLACE VIEW roletree AS
WITH RECURSIVE
roltree AS (
  SELECT u.rolname AS rolname,
         u.oid AS roloid,
         u.rolcanlogin,
         u.rolsuper,
         '{}'::name[] AS rolparents,
         NULL::oid AS parent_roloid,
         NULL::name AS parent_rolname
  FROM pg_catalog.pg_authid u
  LEFT JOIN pg_catalog.pg_auth_members m on u.oid = m.member
  LEFT JOIN pg_catalog.pg_authid g on m.roleid = g.oid
  WHERE g.oid IS NULL
  UNION ALL
  SELECT u.rolname AS rolname,
         u.oid AS roloid,
         u.rolcanlogin,
         u.rolsuper,
         t.rolparents || g.rolname AS rolparents,
         g.oid AS parent_roloid,
         g.rolname AS parent_rolname
  FROM pg_catalog.pg_authid u
  JOIN pg_catalog.pg_auth_members m on u.oid = m.member
  JOIN pg_catalog.pg_authid g on m.roleid = g.oid
  JOIN roltree t on t.roloid = g.oid
)
SELECT
  r.rolname,
  r.roloid,
  r.rolcanlogin,
  r.rolsuper,
  r.rolparents
FROM roltree r
ORDER BY 1;

-- For example purposes, given this set of roles
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
        FROM pg_catalog.pg_auth_members m
        JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
        WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolbypassrls
FROM pg_catalog.pg_roles r
ORDER BY 1;
                                  List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+------------
 bob       |                                                            | {}
 dba_user  |                                                            | {su}
 joe       |                                                            | {newbs}
 newbs     | Cannot login                                               | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 su        | No inheritance, Cannot login                               | {postgres}

-- This query shows current status is not acceptable
-- 1) postgres can login directly
```

```
-- 2) dba_user can login and is able to escalate without using set_user()
SELECT
  ro.rolname,
  ro.roloid,
  ro.rolcanlogin,
  ro.rolsuper,
  ro.rolparents
FROM roletree ro
WHERE (ro.rolcanlogin AND ro.rolsuper)
OR
(
    ro.rolcanlogin AND EXISTS
    (
      SELECT TRUE FROM roletree ri
      WHERE ri.rolname = ANY (ro.rolparents)
      AND ri.rolsuper
    )
);
 rolname  | roloid | rolcanlogin | rolsuper |  rolparents
----------+--------+-------------+----------+---------------
 dba_user |  16387 | t           | f        | {postgres,su}
 postgres |     10 | t           | t        | {}
(2 rows)

-- Fix it
REVOKE postgres FROM su;
ALTER USER postgres NOLOGIN;

-- Rerun the query - shows current status is acceptable
SELECT
  ro.rolname,
  ro.roloid,
  ro.rolcanlogin,
  ro.rolsuper,
  ro.rolparents
FROM roletree ro
WHERE (ro.rolcanlogin AND ro.rolsuper)
OR
(
    ro.rolcanlogin AND EXISTS
    (
      SELECT TRUE FROM roletree ri
      WHERE ri.rolname = ANY (ro.rolparents)
      AND ri.rolsuper
    )
);
 rolname | roloid | rolcanlogin | rolsuper | rolparents
---------+--------+-------------+----------+------------
(0 rows)
```

TODO

The following changes/enhancements are contemplated:

- Improve regression tests
- Add ability to create dependencies in `shared_preload_libraries` such that extension order does not matter.

Licensing

Please see the LICENSE file.