**Adventures in Installing SEPostgres on a CentOS 7 Machine**

NCSU Skyward Federal Senior Design Team Spring 2020

Author: Spencer Yoder

**What is SEPostgres?**

[SEPostgres](https://wiki.postgresql.org/wiki/SEPostgreSQL_Documentation) is a module for [Postgres](https://www.postgresql.org/about/) which enforces MAC (Mandatory Access Control) through [SELinux](https://www.redhat.com/en/topics/linux/what-is-selinux). It allows database elements to be protected by SELinux.

**Cloning the Postgres Source Code**

As we discovered, the only way to get SEPostgres is to build it from the [Postgres source code](https://github.com/postgres/postgres). We cloned straight from the master branch; however, we found that Postgres maintains a branch for each version. I’d recommend cloning a particular version since Postgres is already sparsely documented as it is. After you get your code, you should build Postgres.

**Building Postgres**

After we cloned Postgres, we had to build it. Before it could be successfully built, we installed the following packages with yum:

* [readline\_devel](https://tiswww.case.edu/php/chet/readline/rltop.html)
* [zlib\_devel](https://www.zlib.net/)
* [libselinux\_devel](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/6.2_technical_notes/libselinux)
* [bison](https://www.gnu.org/software/bison/)
* [gcc](https://gcc.gnu.org/)
* [flex](https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator))

The build process will halt if you’re missing a package. We installed every package it said we were missing. However, some options can be ignored upon configuration. [Here](https://www.postgresql.org/docs/12/install-procedure.html) is a list of configuration options. Note that this list is incomplete, such is the way of Postgres documentation.

Speaking of configuration, that’s done by running the configure binary inside the Postgres directory with ./configure. To install SEPostgres, we used

**./configure --enable-debug --enable-cassert --with-selinux**

As described [here](https://wiki.postgresql.org/wiki/SEPostgreSQL_Documentation#Installation).

If configure runs successfully, the next step is to run

**sudo make**

and then

**sudo make install**

If both of these run successfully, Postgres should be installed in /usr/local/pgsql. If it’s not, you may be able to find the installation directory with

**sudo find / -n pgsql**

**Building SEPostgres**

SEPostgres is still not built. We found our SEPostgres directory at /home/maintuser/postgres/contrib/sepgsql. Again, this can be located with

**sudo find / -n sepgsql**

After changing to this directory, you can run

**sudo make**

**sudo make install**

If both of these succeed, SEPostgres should be built

**Configuring the System to Run Postgres**

Postgres creates [three databases](https://chartio.com/resources/tutorials/how-to-list-databases-and-tables-in-postgresql-using-psql/) by default, template0, template1, and postgres. A Linux user called ‘postgres’ needs to be created to use Postgres with the postgres database. After postgres is launched with this database, [other databases can be created and mapped to users](https://medium.com/coding-blocks/creating-user-database-and-adding-access-on-postgresql-8bfcd2f4a91e). Create the postgres user with:

**sudo adduser postgres**

Many Postgres commands start with **sudo -i -u postgres** which runs the succeeding command as the user postgres. In order to authorize this on your system, run

**sudo visudo**

to edit /etc/sudoers.tmp to allow your user or root to login as postgres. In our file, we uncommented an existing line:

**%wheel ALL=(ALL) NOPASSWD: ALL**

Which says, all users in the group wheel can login as any user without a password. If you’re missing such a line, you can:

Enter **i** to insert text

Type a line like the following:

**username ALL=(ALL) NOPASSWD: ALL**

For your default username. If that doesn’t work, you can use the username ‘root’.

Enter **esc** **:wq** to exit out of vi.

Next, run

**sudo chown postgres /usr/local/pgsql**

**sudo -i -u postgres /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data --no-locale**

To give postgres full access to the pgsql directory and initialize a database.

This may result in some issues with accessing files in this directory if you’re not logged in as postgres.

**sudo -i -u postgres**

will allow you to login as postgres.

This next part is optional: if you want Postgres to launch every time your system starts up, follow these next steps:

Add this file: [postgres.service](https://drive.google.com/file/d/1vDtNvee8rRlv5WoPiGwFUgmbAod6CXh7/view) to the directory **/usr/lib/systemd/system**. This file is a file which was modified by us from Postgres’s default service file. After adding this file, run the commands:

**sudo systemctl daemon-reload**

which tells systemctl about your new service file

**sudo systemctl enable postgres**

which tells systemctl to start the postgres server every time your

**systemctl is-enabled postgres**

which will let you know whether your postgres service was enabled

**Configuring the System to Run SEPostgres**

This step is relatively simple, all we did was edit /usr/local/pgsql/data/postgresql.conf, which contains a line saying

**shared\_preload\_libraries = ''**

Note that this file is very large. This line is towards the bottom. We used nano as a text editor, which allows you to search for text with ctrl+w.

Modify this line to say

**shared\_preload\_libraries = 'sepgsql'**

Finally, run this command:

**sudo -i -u postgres /usr/local/pgsql/bin/postgres**

**--single -F -O -c exit\_on\_error=true**

**-D /usr/local/pgsql/data postgres <**

**/usr/local/pgsql/share/contrib/sepgsql.sql > /dev/null**

This should all be on one line, it was broken up to maximize readability.

This command was extracted from [here](https://wiki.postgresql.org/wiki/SEPostgreSQL_Documentation#Installation). We don’t know what it does exactly.

A quick note about the full command paths: we originally added /usr/local/pgsql/bin to the path for the default user, but that doesn’t transfer when we run as user postgres. A way to fix that would be to run

**sudo -i -u postgres**

and then [modify the .bashrc file](https://docs.oracle.com/cd/E19062-01/sun.mgmt.ctr36/819-5418/gaznb/index.html) to add the postgres bin to the PATH environment variable. We ended up just using the full path names to the binaries to run everything.

There’s some information [here](https://wiki.postgresql.org/wiki/SEPostgreSQL_Documentation#Regression_Tests) about configuring SEPostgres for regression tests, but at the time of writing, we have not had to do that. Granted, we have not connected our app to SEPostgres yet. There should be a separate document detailing how we did that.

**Launching SEPostgres**

To start the SEPostgres server (if it’s not already started), you can run

**sudo systemctl start postgres**

if you have the postgres.service file configured, or

**sudo -i -u postgres /usr/local/pgsql/bin/pg\_ctl -D /usr/local/pgsql/data start**

if you don’t. [pg\_ctl](https://www.postgresql.org/docs/10/app-pg-ctl.html) is a command which helps manage your server. The -D argument tells it where to find your database files.

To issue commands to the server, run

**sudo -i -u postgres /usr/local/pgsql/bin/psql**

to open an interface where you can type in SQL commands.

If you create some data elements, you can assign them SELinux labels with the [SECURITY LABEL](https://www.postgresql.org/docs/9.1/sql-security-label.html) command. Security labels are stored in the [pg\_seclabel](https://www.postgresql.org/docs/10/catalog-pg-seclabel.html) catalog.

We automated a lot of this process in this shell script: [sepostgres.sh](https://drive.google.com/file/d/1UDtKxdY7nGUHCW0B4zzM1qbAYTc--s-v/view)