POSTGRESQL 9.X
SUPPLEMENTAL PROCEDURES

Version 1, Release 2

26 January 2018

Developed by DISA for the DoD

Trademark Information

Names, products, and services referenced within this document may be the trade names, trademarks, or service marks of their respective owners. References to commercial vendors and their products or services are provided strictly as a convenience to our users, and do not constitute or imply endorsement by DISA of any non-Federal entity, event, product, service, or enterprise.

Table of Contents

Page

1. INTRODUCTION

The instructions and code samples in this document are provided to assist with the implementation of the Fixes in the main STIG document. As with the STIG, they are based on the assumption that the operating system is Red Hat Enterprise Linux (RHEL). They are examples that will be useful in many PostgreSQL deployments. However, it is important that the reader (database administrator/system administrator) verify that each example is applicable to the installation in question and tailor it as necessary.

2. APPENDICES

2.1 Appendix A: Supplemental Content for PGS9-00-011600

```
#!/bin/bash
# Lock: Switch roles to NOLOGIN, keep track of the roles that are switched
# Restore: Switch roles to LOGIN, only the roles found in .restore_login

lock_login=false
restore_login=false
current_dir=$(pwd)
restore_file="${current_dir}/.restore_login"

#####################################################################
# Get Options - database_name, help, lock user logins, restore logins
#####################################################################
while getopts ":d:hlr" opt
do
    case ${opt?} in
        d)  database_name=${OPTARG?} ;;
        h)  echo "$0 -d <database_name> -l (lock)|-r (restore)"; exit 0;;
        l)  lock_login=true ;;
        r)  restore_login=true ;;
        *)  echo "Invalid argument: $0 -d <database_name> -l (lock)|-r
(restore)"; exit 1;;
    esac
done

#####################################################################
# Make sure an option is set
#####################################################################\
if ( ! ${lock_login?} && ! ${restore_login?} )
then
    echo "Must choose lock or restore login access"
    echo "$0 -d <database_name> -l|-r"
    exit 1
fi

#####################################################################
# No options or both options detected, error out
#####################################################################\
if [ $# -ne 3 ] || (${lock_login?} && ${restore_login?} )
then
    echo "Must choose lock or restore login access"
    echo "$0 -d <database_name> -l|-r"
    exit 1
```

```
fi

######################################################################
# Keep track of users that we disable/restore so we don't restore
# access to users that have been locked already by DBA
######################################################################
role_array=()

if [[ ! -f ${restore_file?} ]]
then
    touch ${restore_file?}
    chmod 660 ${restore_file?}
else
    readarray -t role_array < ${restore_file?}
fi

######################################################################
# Lock detected but restore file has users - need to restore access
# first so we don't lose track of users
######################################################################
if (( ${lock_login?} && ${#role_array[@]?} > 0 ))
then
    echo "Lock triggered but roles are already disabled.  Restore access then relock:"
    echo "$0 -d <database_name> -r"
    exit 1
fi

######################################################################
# Lock users from login and disconnect their current session
######################################################################
if ${lock_login?}
then
    roles=($(psql -d ${database_name?} -A -t -q -c "SELECT rolname, rolcanlogin FROM pg_roles"))
    if [[ $? != 0 ]]
    then
        exit 1
    fi

    for role in ${roles[@]?}
    do
        rolname=$(echo ${role?} | awk -F'|' '{print $1}')
        rolcanlogin=$(echo ${role?} | awk -F'|' '{print $2}')
        if [[ ${rolname?} != 'postgres' ]]
        then
```

```
            if [[ ${rolcanlogin?} == 't' ]]
            then
                psql -q -c "ALTER ROLE ${rolname?} NOLOGIN"
                if [[ $? != 0 ]]
                then
                    echo "Error altering role.  Exiting.."
                    exit 1
                fi
                echo ${rolname?} >> ${restore_file?}

                psql -q -c "SELECT pg_terminate_backend(pid) FROM
pg_stat_activity WHERE usename='${rolname?}'" >& /dev/null
                if [[ $? != 0 ]]
                then
                    echo "Error terminating role cyrrebt session.  Exiting.."
                    exit 1
                fi
            fi
        fi
    done
    echo "Lock success"
######################################################################
# Restore users login
######################################################################
else
    if (( ${#role_array[@]?} <= 0))
    then
        echo "Nothing to restore"
        exit 0
    fi

    for rolname in ${role_array[@]?}
    do
        psql -q -c "ALTER ROLE ${rolname?} LOGIN"
        if [[ $? != 0 ]]
        then
            echo "Error altering role.  Exiting.."
            exit 1
        fi
    done
    > ${restore_file?}
    echo "Restore success"
fi

exit 0
```

2.2 Appendix B: pgaudit

The following are the core instructions for installing a PostgreSQL cluster with **pgaudit**.

If PostgreSQL is already installed, it is still necessary to ensure that all the software mentioned on this page is present on the server to support **pgaudit**.

Note: The following instructions use the PGDATA environment variable. See Appendix F for instructions on configuring PGDATA.

2.2.1 Installing pgaudit

First we require PostgreSQL to be installed. In this example we are using Red Hat Enterprise Linux 7 RPMs. The appropriate RPM URL should be used from the following webpage: http://yum.postgresql.org/9.5/

```
$ sudo yum update

$ sudo yum install bison flex gcc \
    readline-devel zlib-devel perl-devel \
    perl-ExtUtils-Embed openssl-devel \
    pam-devel libxml2-devel libxslt-devel \
    libuuid-devel openldap-devel tcl-devel \
    python-devel
```

Next we install PostgreSQL 9.5:

```
$ sudo yum install http://yum.postgresql.org/9.5/redhat/rhel-7-x86_64/pgdg-
redhat95-9.5-2.noarch.rpm

$ sudo yum install postgresql95 postgresql95-contrib \
              postgresql95-devel postgresql95-docs \
              postgresql95-libs postgresql95-server
```

Now that PostgreSQL is installed, we need to set PATH to point to the new binaries:

```
$ export PATH=/usr/pgsql-9.5/bin:$PATH
$ pg_config --configure
```

With PostgreSQL installed, we can now install **pgaudit**.

In this example we are using Git. The source code can be uploaded to the webserver by different means, depending on your organization’s rules.

```
$ cd /usr/pgsql-9.5/share/contrib/

$ sudo git clone https://github.com/pgaudit/pgaudit.git

$ cd ./pgaudit

$ sudo PATH=/usr/pgsql-9.5/bin:$PATH make USE_PGXS=1 install
```

pgaudit is built and ready to be configured. First the database needs to be initialized:

```
$ sudo /usr/pgsql-9.5/bin/postgresql95-setup initdb
$ sudo systemctl enable postgresql-9.5
```

Now as the **postgres** user, add **pgaudit** to the **shared_preload_libraries** in **postgresql.conf**:

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

Change **shared_preload_libraries** to the following:

```
shared_preload_libraries = 'pgaudit'
```

As a sudo user, start the PostgreSQL server:

```
# SERVER USING SYSTEMCTL ONLY
$ sudo systemctl start postgresql-9.5

# SERVER USING INITD ONLY
$ sudo service postgresql-9.5 start
```

pgaudit is now installed and ready to be configured.

2.2.2 Configuration

pgaudit is configured using either the postgresql.conf or an included configuration file (see Optional Configuration Organization).

For a complete list of configuration parameters, see https://github.com/pgaudit/pgaudit#settings

2.2.2.1 Example Settings for postgresql.conf

```
# Enable catalog logging - default is 'on'
pgaudit.log_catalog='on'
# Specify the verbosity of log information (INFO, NOTICE, LOG, WARNING, DEBUG)
pgaudit.log_level='log'
# Log the parameters being passed
pgaudit.log_parameter='on'
# Log each relation (TABLE, VIEW, etc.) mentioned in a SELECT or DML statement
pgaudit.log_relation='off'
# For every statement and substatement, log the statement and parameters every
time
pgaudit.log_statement_once='off'
# Define the master role to use for object logging
# pgaudit.role=''
# Choose the statements to log:
# READ - SELECT, COPY
# WRITE - INSERT, UPDATE, DELETE, TRUNCATE, COPY
# FUNCTION - Function Calls and DO Blocks
# ROLE - GRANT, REVOKE, CREATE/ALTER/DROP ROLE
# DDL - All DDL not included in ROLE
# MISC - DISCARD, FETCH, CHECKPOINT, VACUUM
pgaudit.log='ddl, role, read'
```

2.2.2.2 Setting log_line_prefix

It is advisable to change log_line_prefix in postgresql.conf to match your auditing needs.

At a minimum, it is suggested to set the parameter to the following:

```
$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
log_line_prefix = '%m %u %d: '
```

This will prefix all logged events with:

```
< 2016-01-28 19:43:12.126 UTC bob postgres: >
```

2.3 Appendix C: Logging

The following are the core instructions for enabling logging in PostgreSQL.
Note: The following instructions use the PGDATA environment variable. See Appendix F for instructions on configuring PGDATA.

2.3.1 Where to Log

2.3.1.1 stderr

PostgreSQL can be configured to use stderr for logging. This allows the server to log events from the database to a directory specified in postgresql.conf.

- **Configuring stderr Logging**

As the database administrator (shown here as postgres), edit the postgresql.conf file:

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

The following parameters must be configured:

```
log_destination = 'stderr'
logging_collector = on
log_directory = 'pg_log'
log_filename = 'postgresql-%a.log'
log_file_mode = 0600
log_truncate_on_rotation = on
log_rotation_age = 1d
log_rotation_size = 0
```

As a sudo user, reload the PostgreSQL server:

```
# SERVER USING SYSTEMCTL ONLY
$ sudo systemctl reload postgresql-9.5

# SERVER USING INITD ONLY
$ sudo service postgresql-9.5 reload
```

2.3.1.2 syslog

PostgreSQL can be configured to use syslog for logging. This allows the server to log events from the database to a centralized location and give log ownership to root instead of the database administrator (shown here as postgres). It is advised to use syslog whenever possible.

2.3.1.3 Configuring syslog Logging

Note: Syslog must be configured in your organization. The following instructions are only to configure PostgreSQL to use syslog.

As the database administrator (shown here as postgres), edit the postgresql.conf file:

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

The following parameters must be configured:

```
log destination = 'syslog'
syslog facility = 'LOCAL0' # choose the facility that makes sense
syslog ident = 'postgres'
```

As a sudo user, reload the PostgreSQL server:

```
# SERVER USING SYSTEMCTL ONLY
$ sudo systemctl reload postgresql-9.5

# SERVER USING INITD ONLY
$ sudo service postgresql-9.5 reload
```

2.3.2 What to Log

PostgreSQL can log a variety of events out of the box. The following parameters can be set to log additional information.

As the database administrator (shown here as postgres), edit the postgresql.conf file:

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

The following parameters can be configured:

```
log_checkpoints = on
log_connections = on
log_disconnections = on
log_duration = off
```

```
log_error_verbosity = default
log_hostname = off
log_lock_waits = on
log_statement = 'none' # pgaudit will be configured to log specific events
log_timezone = 'UTC'
```

Additionally, **log_line_prefix** can be configured to include extra information:

```
log_line_prefix = '< %m %a %u %d %c %s %r >'
#   %a = application name
#   %u = user name
#   %d = database name
#   %r = remote host and port
#   %h = remote host
#   %p = process ID
#   %t = timestamp without milliseconds
#   %m = timestamp with milliseconds
#   %i = command tag
#   %e = SQL state
#   %c = session ID
#   %l = session line number
#   %s = session start timestamp
#   %v = virtual transaction ID
#   %x = transaction ID (0 if none)
#   %q = stop here in non-session
#        processes
```

2.3.3 When to Log

PostgreSQL allows administrators to control what levels of detail are exposed to logs, administrators, and clients.

As the database administrator (shown here as postgres), edit the postgresql.conf file:

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

The following parameters can be configured:

```
client_min_messages = notice
log_min_messages = warning
```

```
log_min_error_statement = error
log_min_duration_statement = -1
```

For more information on logging, see the official documentation:
http://www.postgresql.org/docs/current/static/runtime-config-logging.html

2.4 Appendix D: Row-Level Security Example

2.4.1 Row-Level Security Setup

This code creates a table “accounts” where rows may be labeled with a security label to filter results based on role membership.

```
-- Create table with security labels
CREATE TABLE accounts (id SERIAL PRIMARY KEY, name TEXT NOT NULL,
    phone_number TEXT NOT NULL, security_label TEXT);

ALTER TABLE accounts ENABLE ROW LEVEL SECURITY;

-- Create groups that map to security-labels
CREATE ROLE unclassified;
CREATE ROLE classified;

-- Add roles to groups
CREATE ROLE bob LOGIN IN GROUP unclassified;
CREATE ROLE alice LOGIN IN GROUP classified;

-- Dummy data
INSERT INTO accounts(name, phone_number, security_label)
VALUES ('bob', '123-456-7890', 'unclassified');
INSERT INTO accounts(name, phone_number, security_label)
VALUES ('alice', '098-765-4321', 'classified');

-- Function to check if user is in group for security label filtering
CREATE OR REPLACE FUNCTION user_in_group(group_name TEXT, user_name TEXT)
RETURNS boolean
    AS 'SELECT EXISTS(
            SELECT grosysid FROM pg_group WHERE groname = $1
            AND (SELECT usesysid FROM pg_user
                 WHERE usename = $2) = ANY(grolist));'
    LANGUAGE SQL;

-- Row level security policy for information
CREATE POLICY classification_filter ON accounts
    USING ((SELECT user_in_group(accounts.security_label, current_user)));
```

```
-- Allow access to table
GRANT SELECT ON accounts TO classified;
GRANT SELECT ON accounts TO unclassified;

-- Change to role bob and query table
SET ROLE bob;
SELECT current_user;
SELECT * FROM accounts;

RESET ROLE;

-- Change to role alice and query table
SET ROLE alice;
SELECT current_user;
SELECT * FROM accounts;

RESET ROLE;

-- Cleanup
DROP TABLE IF EXISTS accounts CASCADE;
DROP ROLE IF EXISTS bob, alice, classified, unclassified;
```

2.4.2 Results

```
 current_user
--------------
 bob
(1 row)

 id | name | phone_number | security_label
----+------+--------------+----------------
  1 | bob  | 123-456-7890 | unclassified
(1 row)

RESET
SET
 current_user
--------------
 alice
(1 row)
```

```
 id | name  | phone_number | security_label
----+-------+--------------+----------------
  2 | alice | 098-765-4321 | classified
(1 row)
```

2.5 Appendix E: Installing a PostgreSQL Cluster with pgcrypto

The following are the core instructions for installing a PostgreSQL cluster with pgcrypto.

2.5.1 Installing pgcrypto Extension

The pgcrypto extension is included with the PostgreSQL contrib package. Although included, it needs to be created in the database.

This installation assumes that:

- The database has been initialized
- The contrib package is installed

As the database administrator (shown here as postgres), run the following:

```
$ sudo su - postgres
$ psql -c "CREATE EXTENSION pgcrypto"
```

pgcrypto is now ready for use. For more information on using pgcrypto, see the official documentation: http://www.postgresql.org/docs/current/static/pgcrypto.html

2.6 Appendix F: Finding the PostgreSQL Configured Data Directory (PGDATA)

2.6.1 How to Find PGDATA

To see where the data directory (PGDATA) is, run the following commands:

```
$ sudo su - postgres
$ psql -c "SHOW data_directory";
```

2.6.2 Export PGDATA Variable in .bashrc

After finding the configured data directory, it is handy to export the PGDATA variable in the database owner's .bashrc file.

As the database administrator (shown here as “postgres”), edit the .bashrc file in home directory of the user (this is just an example; use the result from the query above):

```
$ sudo su - postgres
$ echo "export PGDATA='/var/lib/pgsql/9.5/data' >> ~/.bashrc"
```

Next, reload the .bashrc file:

```
$ sudo su - postgres
$ source ~/.bashrc
```

Last, verify the variable is set:

```
$ echo ${PGDATA?}
```

2.7 Appendix G: PostgreSQL SSL Configuration

The following instructions assume that OpenSSL is enabled on the server at the operating system level. To be sure that OpenSSL is implemented using FIPS-certified modules, the operating system must be Red Hat Enterprise Linux (RHEL).

These instructions will guide you through the process of configuring PostgreSQL to use SSL for secure connections.

We will be placing an intermediate certificate authority (CA) in the chain of trust. While this is not strictly necessary, it is a good idea as it allows you to keep your root CA safe (i.e., offline) once the intermediate certificates have been created. In the case of a security breach, only the intermediate certificate needs to be revoked.

We will show how to create a self-signed root CA for the purposes of demonstration. In practice, use a DoD-approved CA.

2.7.1 Creating Certificates

2.7.1.1 Configuring openssl.cnf

First, default configuration for openssl.cnf needs to be changed to support signed Certification Revocation lists (CRL) and certificates.

To find where openssl.cnf is located, execute the following command with a sudo user:

```
$ find / -name "openssl.cnf"
```

With the location of openssl.cnf found, edit the file, locate the [v3_ca] section, and uncomment the following default:

```
keyUsage = cRLSign, keyCertSign
```

- **Create Self-Signed CA (optional)**

  You will likely have a certificate signed by a trusted CA, but for some installations or to just try out these instructions, you may want to create a self-signed certificate.

- **Create Private Key**

```
$ openssl genrsa -aes256 -out ca.key 4096
```

  You will be required to enter a passphrase. This should be long and guarded well.

- **Create Self-Signed Certificate**

```
$ openssl req -new -x509 -sha256 -days 1825 -key ca.key -out ca.crt \
    -subj "/C=US/ST=<STATE>/L=<LOCATION>/O=<ORGANIZATION NAME>/CN=root-ca"
```

2.7.1.2 Create Server/Client Intermediate CA

With the root CA created, next create the intermediate CAs that will be used to sign server and client certificates.

- **Create Server Intermediate Certificate**

  The following will create the server-intermediate key, signing request, and certificate. You will be required to enter a passphrase. It is best not to reuse the passphrase from your root key.

```
$ openssl genrsa -aes256 -out server-intermediate.key 4096

$ openssl req -new -sha256 -days 1825 -key server-intermediate.key -out server-
intermediate.csr \
    -subj "/C=US/ST=<STATE>/L=<LOCATION>/O=<ORGANIZATION NAME>/CN=server-im-ca"
```

```
$ openssl x509 -extfile <OPENSSL.CNF_LOCATION_HERE> -extensions v3_ca -req -
days 1825 \
    -CA ca.crt -CAkey ca.key -CAcreateserial \
    -in server-intermediate.csr -out server-intermediate.crt
```

- **Create Client Intermediate Certificate**

```
$ openssl genrsa -aes256 -out client-intermediate.key 4096

$ openssl req -new -sha256 -days 1825 -key client-intermediate.key -out client-
intermediate.csr \
    -subj "/C=US/ST=<STATE>/L=<LOCATION>/O=<ORGANIZATION NAME>/CN=client-im-ca"

$ openssl x509 -extfile <OPENSSL.CNF_LOCATION_HERE> -extensions v3_ca -req -
days 1825 \
    -CA ca.crt -CAkey ca.key -CAcreateserial \
    -in client-intermediate.csr -out client-intermediate.crt
```

2.7.1.3 Create Server/Client Certificate

Server and client certificates are signed by their respective intermediate CAs rather than the root CA. Additionally, the common name on server certificates MUST match the hostname of the server, and the common name of the client certificates MUST match the client's PostgreSQL user logon (or be mapped in pg_ident.conf). The private keys will be created without passphrases to allow automatic startup of the PostgreSQL server and client.

```
Create Server Certificate
```

Note: hostname should be entered for the common name. To find the hostname of your system, run the following:

```
$ hostname
```

With your hostname configured, create the server certificate. (**Note:** Fill in CN with your system’s hostname):

```
$ openssl req -nodes -new -newkey rsa:4096 -sha256 -keyout server.key -out
server.csr \
    -subj "/C=US/ST=<STATE>/L=<LOCATION>/O=<ORGANIZATION
NAME>/CN=<HOSTNAME_HERE>"
```

```
$ openssl x509 -extfile <OPENSSL.CNF_LOCATION_HERE> -extensions usr_cert -req -
days 1825 \
    -CA server-intermediate.crt -CAkey server-intermediate.key \
    -CAcreateserial -in server.csr -out server.crt
```

- **Create Client Certificate**

Note: The common name for the client cert must be mapped to a postgres role. This can be done by using a specific username or adding an ident_map.

```
$ openssl req -nodes -new -newkey rsa:4096 -sha256 -keyout client.key -out
client.csr \
    -subj "/C=US/ST=<STATE>/L=<LOCATION>/O=<ORGANIZATION NAME>/CN=bob@ssl-
test.com"

$ openssl x509 -extfile <OPENSSL.CNF_LOCATION_HERE> -extensions usr_cert -req -
days 1825 \
    -CA client-intermediate.crt -CAkey client-intermediate.key \
    -CAcreateserial -in client.csr -out client.crt
```

2.7.2 Configuring PostgreSQL

The previous instructions built all the certificates required to enable SSL on PostgreSQL. The following instructions will demonstrate how PostgreSQL is configured to use the certificates and authenticate users via SSL.

2.7.2.1 Create Test Role

First, create a role that can log in and be used for testing the certificates:

```
$ sudo su - postgres
$ psql -c "CREATE ROLE bob LOGIN"
```

2.7.2.2 Server Configuration

The examples below will use ${PGDATA?} as the data_directory setting in postgresql.conf. To find the location of the data_directory, run the following SQL:

```
$ sudo su - postgres
$ psql -c "SHOW data_directory"
```

- **Copy Root CA**

  Copy the root CA certificate to the PostgreSQL Data Directory:

```
$ sudo cp ca.crt ${PGDATA?}/ca.crt
$ sudo cp server.key ${PGDATA?}/server.key
$ sudo cat server.crt server-intermediate.crt ca.crt > ./server.crt.new
$ sudo cp server.crt.new ${PGDATA?}/server.crt
```

- **Set Permissions**

```
$ sudo chown postgres:postgres \
      ${PGDATA?}/ca.crt \
      ${PGDATA?}/server.crt \
      ${PGDATA?}/server.key

$ sudo chmod 600 \
      ${PGDATA?}/ca.crt \
      ${PGDATA?}/server.crt \
      ${PGDATA?}/server.key
```

- **Configure postgresql.conf**

  Now configure postgresql.conf with the SSL settings:

```
$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
ssl = true
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'ca.crt'
```

- **Configure pg_ident.conf**

  In order to map system usernames to postgres roles, pg_ident must be configured.

  The example common name for the client certificate is bob@ssl-test.com. Map that CN to a postgres role to allow bob login:

```
$ sudo su - postgres
$ vi ${PGDATA?}/pg_ident.conf
```

Add the following parameters:

```
ssl-test    bob@ssl-test.com   bob
```

- **Configure pg_hba.conf**

  Be sure that pg_hba.conf requires certs for the clients if you do not want them to be optional. The following example will demonstrate making all users require SSL when connecting locally (this is just a test). **Note:** Comment out any other entries that may be in the authentication file already.

```
hostssl     all     all     127.0.0.1/32    cert clientcert=1 map=ssl-test
hostssl     all     all     ::1/128         cert clientcert=1 map=ssl-test
```

As a system administrator, restart the PostgreSQL server:

```
# SERVER USING SYSTEMCTL ONLY
$ sudo systemctl restart postgresql-9.5

# SERVER USING INITD ONLY
$ sudo service postgresql-9.5 restart
```

2.7.3 Client Configuration

The examples below assume you are logged on to the OS as the user you want to configure.

```
$ mkdir ~/.postgresql
$ cp ca.crt ~/.postgresql/root.crt
$ cp client.key ~/.postgresql/postgresql.key
$ cat client.crt client-intermediate.crt ca.crt > ~/.postgresql/postgresql.crt
```

2.7.3.1 Set Permissions

```
$ chmod 600 \
      ~/.postgresql/root.crt \
      ~/.postgresql/postgresql.key \
      ~/.postgresql/postgresql.crt
```

Note: These files can also be configured with environment variables. See http://www.postgresql.org/docs/current/static/libpq-ssl.html for more information.

2.7.4 Running the client

When running client software, it is best to use the verify-full ssl mode. See the link in Client Configuration for a description of what the ssl modes mean and what level of protection they provide.

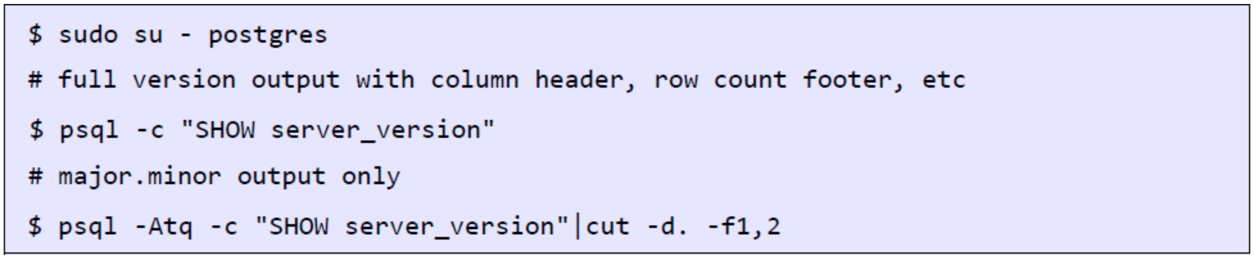
Following is an example using psql:

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

2.8 Appendix H: Finding the PostgreSQL Configured Version (PGVER)

2.8.1 How to Find PGVER

To determine PostgreSQL configured version (PGVER) run the following commands:

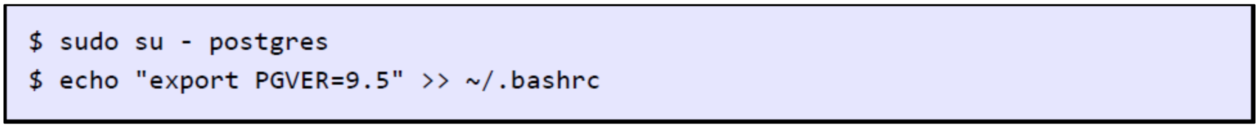


```
$ sudo su - postgres
# full version output with column header, row count footer, etc
$ psql -c "SHOW server_version"
# major.minor output only
$ psql -Atq -c "SHOW server_version"|cut -d. -f1,2
```

2.8.2 Export PGVER Variable in .bashrc

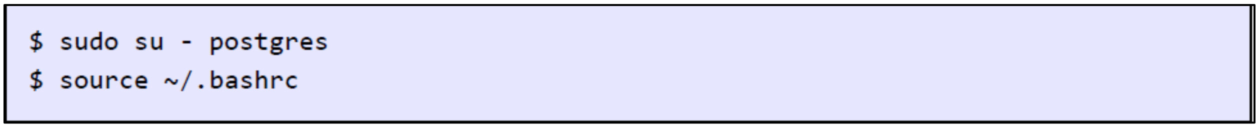
After finding the configured PostgreSQL version, it is handy to export the PGVER variable in the database owner's .bashrc file. For this example we are using 9.5.

As the database administrator (shown here as “postgres”), edit the .bashrc file in home directory of the user (this is just an example; use the result from the query above):



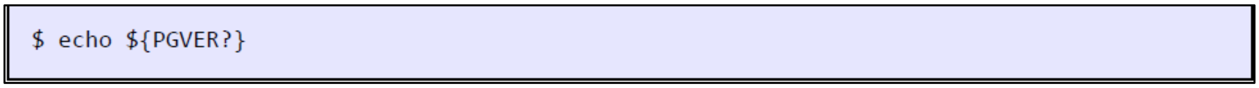
```
$ sudo su - postgres
$ echo "export PGVER=9.5" >> ~/.bashrc
```

Next, reload the .bashrc file:



```
$ sudo su - postgres
$ source ~/.bashrc
```

Last, verify the variable is set:



```
$ echo ${PGVER?}
```