CIS Benchmarks

CIS PostgreSQL 12 Benchmark

v1.0.0 - 11-19-2019

Terms of Use

Please see the below link for our current terms of use:

https://www.cisecurity.org/cis-securesuite/cis-securesuite-membership-terms-of-use/

Table of Contents

Overview

This document, CIS PostgreSQL 12 Benchmark, provides prescriptive guidance for establishing a secure configuration posture for PostgreSQL 12. This guide was tested against PostgreSQL 12 running on CentOS 8, but applies to other Linux distributions as well. To obtain the latest version of this guide, please visit http://benchmarks.cisecurity.org. If you have questions, comments, or have identified ways to improve this guide, please write us at feedback@cisecurity.org.

Intended Audience

This document is intended for system and application administrators, security specialists, auditors, help desk, and platform deployment personnel who plan to develop, deploy, assess, or secure solutions that incorporate PostgreSQL 12.

Consensus Guidance

This benchmark was created using a consensus review process comprised of subject matter experts. Consensus participants provide perspective from a diverse set of backgrounds including consulting, software development, audit and compliance, security research, operations, government, and legal.

Each CIS benchmark undergoes two phases of consensus review. The first phase occurs during initial benchmark development. During this phase, subject matter experts convene to discuss, create, and test working drafts of the benchmark. This discussion occurs until consensus has been reached on benchmark recommendations. The second phase begins after the benchmark has been published. During this phase, all feedback provided by the Internet community is reviewed by the consensus team for incorporation in the benchmark. If you are interested in participating in the consensus process, please visit https://workbench.cisecurity.org/.

Typographical Conventions

The following typographical conventions are used throughout this guide:

Convention	Meaning
`Stylized Monospace font`	Used for blocks of code, command, and script examples. Text should be interpreted exactly as presented.
`Monospace font`	Used for inline code, commands, or examples. Text should be interpreted exactly as presented.
<italic font in brackets>	Italic texts set in angle brackets denote a variable requiring substitution for a real value.
Italic font	Used to denote the title of a book, article, or other publication.
Note	Additional information or caveats

Scoring Information

A scoring status indicates whether compliance with the given recommendation impacts the assessed target's benchmark score. The following scoring statuses are used in this benchmark:

Scored

Failure to comply with "Scored" recommendations will decrease the final benchmark score. Compliance with "Scored" recommendations will increase the final benchmark score.

Not Scored

Failure to comply with "Not Scored" recommendations will not decrease the final benchmark score. Compliance with "Not Scored" recommendations will not increase the final benchmark score.

Profile Definitions

The following configuration profiles are defined by this Benchmark:

- **Level 1 - PostgreSQL**

  Items in this profile apply to PostgreSQL 10 and intend to:

  - be practical and prudent;
  - provide a clear security benefit; and
  - not inhibit the utility of the technology beyond acceptable means.

  Note: The intent of this profile is to include checks that can be assessed by remotely connecting to PostgreSQL. Therefore, file system-related checks are not contained in this profile.

- **Level 1 - PostgreSQL on Linux**

  Items in this profile apply to PostgreSQL 10 running on Linux and intend to:

  - be practical and prudent;
  - provide a clear security benefit; and
  - not inhibit the utility of the technology beyond acceptable means.

Acknowledgements

This benchmark exemplifies the great things a community of users, vendors, and subject matter experts can accomplish through consensus collaboration. The CIS community thanks the entire consensus team with special recognition to the following individuals who contributed greatly to the creation of this guide:

Author
Doug Hunley

Editor
Tim Harrison CISSP, ICP, Center for Internet Security

Recommendations

1 Installation and Patches

One of the best ways to ensure secure PostgreSQL security is to implement security updates as they come out, along with any applicable OS patches that will not interfere with system operations. It is additionally prudent to ensure the installed version has not reached end-of-life.

1.1 Ensure packages are obtained from authorized repositories (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

When obtaining and installing software packages (typically via `dnf`), it's imperative that packages are sourced only from valid and authorized repositories. For PostgreSQL, the canonical repositories are the official PostgreSQL YUM repository (yum.postgresql.org) and the official PostgreSQL APT repository (apt.postgresql.org).

Rationale:

Being open source, PostgreSQL packages are widely available across the internet through RPM aggregators and providers. However, using invalid or unauthorized sources for packages can lead to implementing untested, defective, or malicious software.

Many organizations choose to implement a local software repository within their organization. Care must be taken to ensure that only valid and authorized packages are downloaded and installed into such local repositories.

Audit:

Identify and inspect configured repositories to ensure they are all valid and authorized sources of packages. The following is an example of a simple CENTOS 8 install illustrating the use of the `dnf repolist all` command.

```
$ whoami
root
$ dnf repolist all | grep enabled:
```

```
AppStream                   CentOS-8 - AppStream                  enabled:
4,928
BaseOS                      CentOS-8 - Base                       enabled:
2,713
extras                      CentOS-8 - Extras                     enabled:
3
```

Ensure the list of configured repositories only includes organization-approved repositories. If any unapproved repositories are listed, this is a fail.

Remediation:

Alter the configured repositories so they only include valid and authorized sources of packages.

As an example of adding an authorized repository, we will install the PGDG repository RPM from 'yum.postgresql.org' (note that because of a change in the way packaging is handled in RHEL 8, we also need to disable the PostgreSQL module):

```
# whoami
root
# dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-
x86_64/pgdg-redhat-repo-latest.noarch.rpm
Last metadata expiration check: 0:01:35 ago on Fri 04 Oct 2019 01:19:37 PM
EDT.
[snip]
Installed:
  pgdg-redhat-repo-42.0-5.noarch

Complete!
# dnf -qy module disable postgresql
#
```

Verify the repository has been added and is enabled:

```
# whoami
root
# dnf repolist all | grep enabled:
AppStream                     CentOS-8 - AppStream                enabled:
4,928
BaseOS                        CentOS-8 - Base                     enabled:
2,713
extras                        CentOS-8 - Extras                   enabled:
3
pgdg10                        PostgreSQL 10 for RHEL/CentOS 8 - x enabled:
504
pgdg11                        PostgreSQL 11 for RHEL/CentOS 8 - x enabled:
526
pgdg12                        PostgreSQL 12 for RHEL/CentOS 8 - x enabled:
377
pgdg94                        PostgreSQL 9.4 for RHEL/CentOS 8 -  enabled:
184
```

```
pgdg95                        PostgreSQL 9.5 for RHEL/CentOS 8 -  enabled:
322
pgdg96                        PostgreSQL 9.6 for RHEL/CentOS 8 -  enabled:
482
```

References:

1. https://wiki.centos.org/PackageManagement/Yum/
2. https://www.centos.org/docs/5/html/5.2/Deployment_Guide/s1-yum-yumconf-repository.html
3. https://en.wikipedia.org/wiki/Yum_(software)
4. https://www.howtoforge.com/creating_a_local_yum_repository_centos
5. https://yum.postgresql.org
6. https://apt.postgresql.org

CIS Controls:

Version 6

2 Inventory of Authorized and Unauthorized Software
Inventory of Authorized and Unauthorized Software

Version 7

2.1 Maintain Inventory of Authorized Software
Maintain an up-to-date list of all authorized software that is required in the enterprise for any business purpose on any business system.

1.2 Ensure Installation of Binary Packages (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The PostgreSQL packages are installed on the Operating System from valid source.

Rationale:

Standard Linux distributions, although possessing the requisite packages, often do not have PostgreSQL pre-installed. The installation process includes installing the binaries and the means to generate a data cluster too. Package installation should include both the server and client packages. Contribution modules are optional depending upon one's architectural requirements (they are recommended though).

From a security perspective, it's imperative to verify the PostgreSQL binary packages are sourced from a valid software repository. For a complete listing of all PostgreSQL binaries available via configured repositories inspect the output from `dnf provides '*libpq.so'`.

Audit:

To inspect what versions of PostgreSQL packages are installed, **and** which repo they came from, we can query using the `dnf` and `rpm` commands. As illustrated below, PostgreSQL 11.3 packages are installed:

```
# whoami
root
# dnf info $(rpm -qa|grep postgres) | egrep '^Name|^Version|^From'
Name         : postgresql12
Version      : 12.0
From repo    : pgdg12
Name         : postgresql12-contrib
Version      : 12.0
From repo    : pgdg12
Name         : postgresql12-libs
Version      : 12.0
From repo    : pgdg12
Name         : postgresql12-server
Version      : 12.0
From repo    : pgdg12
```

If the expected binary packages are not installed, are not the expected versions, or did not come from an appropriate repo, this is a fail.

Remediation:

If the version of PostgreSQL installed is not 12.x, the packages may be uninstalled using this command:

```
$ whoami
root
$ dnf remove $(rpm -qa|grep postgres)
```

The next recommendation "1.3 Ensure Installation of Community Packages" describes how to explicitly choose which version of PostgreSQL to install, regardless of Linux distribution association.

Impact:

If the PostgreSQL version shipped as part of the default binary installation associated with your Linux distribution satisfies your requirements, this may be adequate *for development and testing purposes*. However, *for production instances* it's generally recommended to install the *latest stable release* of PostgreSQL.

CIS Controls:

Version 6

2 Inventory of Authorized and Unauthorized Software
Inventory of Authorized and Unauthorized Software

Version 7

2.1 Maintain Inventory of Authorized Software
Maintain an up-to-date list of all authorized software that is required in the enterprise for any business purpose on any business system.

1.3 Ensure Installation of Community Packages (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Adding, and installing, the PostgreSQL community packages to the host's package repository.

Rationale:

It's an unfortunate reality that Linux distributions do not always have the most up-to-date versions of PostgreSQL. Disadvantages of older releases include: missing bug patches, no access to highly desirable contribution modules, no access to 3rd party projects that are complimentary to PostgreSQL, and no upgrade path migrating from one version of PostgreSQL to the next. The worst set of circumstances is to be limited to a version of the RDBMS that has reached its end-of-life.

From a security perspective, it's imperative that Postgres Community Packages are only obtained from the official website https://yum.postgresql.org/. Being open source, the Postgres packages are widely available over the internet via myriad package aggregators and providers. Obtaining software from these unofficial sites risks installing defective, corrupt, or downright malicious versions of PostgreSQL.

Audit:

First determine whether or not the PostgreSQL Community Packages are installed. For this example, we are using a host that does not have any PostgreSQL packages installed and offer resolution in the Remediation Procedure below.

```
$ whoami
root
$ yum info $(rpm -qa|grep postgres) | egrep '^Name|^Version|^From'
$
```

If the expected community packages are not installed, are not the expected versions, or are not from the PGDG repo, this is a fail.

Remediation:

The following example adds the PGDG repository RPM for PostgreSQL, configures `dnf` to prefer the PGDG packages for version 12, and installs the client-server-contributions rpms to the host where you want to install the RDBMS.

Using a web browser, go to http://yum.postgresql.org and navigate to the repo download link for your OS and version. Copy the URL to the repo file, and then tell `dnf` to install it:

```
# whoami
root
# dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-
x86_64/pgdg-redhat-repo-latest.noarch.rpm
Last metadata expiration check: 0:01:35 ago on Fri 04 Oct 2019 01:19:37 PM
EDT.
[snip]
Installed:
  pgdg-redhat-repo-42.0-5.noarch

Complete!
# dnf -qy module disable postgresql
#
```

Now, configure `dnf` to prefer the PGDG packages for version 12:

```
# cd /etc/yum.repos.d
# for i in AppStream Base Extras
do
echo 'exclude=postgresql*' >> CentOS-$i.repo
done
```

Finally, install the PostgreSQL packages:

```
# whoami
root
# dnf -y groupinstall 'PostgreSQL Database Server 12 PGDG'
Dependencies resolved.
[snip]
Installed:
  postgresql12-12.0-1PGDG.rhel8.x86_64
  postgresql12-contrib-12.0-1PGDG.rhel8.x86_64
  postgresql12-libs-12.0-1PGDG.rhel8.x86_64
  postgresql12-server-12.0-1PGDG.rhel8.x86_64
  python2-2.7.15-22.module_el8.0.0+32+017b2cba.x86_64
  python2-libs-2.7.15-22.module_el8.0.0+32+017b2cba.x86_64
  python2-pip-9.0.3-13.module_el8.0.0+32+017b2cba.noarch
  python2-setuptools-39.0.1-11.module_el8.0.0+32+017b2cba.noarch
  libicu-60.2-7.el8.x86_64
  libxslt-1.1.32-3.el8.x86_64

Complete!
```

Note: The above-mentioned example is referenced as an illustration only. Package names and versions may differ.

References:

1. https://www.postgresql.org/
2. https://www.postgresql.org/support/versioning/
3. https://www.postgresql.org/developer/roadmap/
4. https://yum.postgresql.org/repopackages.php

CIS Controls:

Version 6

18.1 Use Only Vendor-supported Software
For all acquired application software, check that the version you are using is still supported by the vendor. If not, update to the most current version and install all relevant patches and vendor security recommendations.

Version 7

18.3 Verify That Acquired Software is Still Supported
Verify that the version of all software acquired from outside your organization is still supported by the developer or appropriately hardened based on developer security recommendations.

1.4 Ensure systemd Service Files Are Enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Confirm, and correct if necessary, the PostgreSQL `systemd` service is enabled.

Rationale:

Enabling the `systemd` service on the OS ensures the database service is active when a change of state occurs as in the case of a system startup or reboot.

Audit:

The default operating target on `systemd`-powered operating systems is typically "multi-user". One confirms the default target by executing the following:

```
$ whoami
root
$  systemctl get-default
multi-user.target
$ systemctl list-dependencies multi-user.target | grep -i postgres
$
```

If the intended PostgreSQL service is not registered as a dependency (or "want") of the default target (no output for the 3rd command above), this is a fail.

Remediation:

Irrespective of package source, PostgreSQL services can be identified because it typically includes the text string "postgresql". PGDG installs do not automatically register the service as a "want" of the default `systemd` target. Multiple instances of PostgreSQL services often distinguish themselves using a version number.

```
# whoami
root
# systemctl enable postgresql-12
Created symlink /etc/systemd/system/multi-user.target.wants/postgresql-
12.service → /usr/lib/systemd/system/postgresql-12.service.
# systemctl list-dependencies multi-user.target | grep -i postgres
● ├─postgresql-12.service
#
```

References:

1. https://linuxcommand.org/man_pages/runlevel8.html
2. https://linuxcommand.org/man_pages/chkconfig8.html
3. https://www.tldp.org/LDP/sag/html/run-levels-intro.html

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

1.5 Ensure Data Cluster Initialized Successfully (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

First time installs of PostgreSQL requires the instantiation of the database cluster. A database cluster is a collection of databases that are managed by a single server instance.

Rationale:

For the purposes of security, PostgreSQL enforces ownership and permissions of the data-cluster such that:

- An initialized data-cluster is owned by the UNIX account that created it.
- The data-cluster cannot be accessed by other UNIX user-accounts.
- The data-cluster cannot be created or owned by `root`
- The PostgreSQL process cannot be invoked by `root` nor any UNIX user account other than the owner of the data cluster.

Incorrectly instantiating the data-cluster will result in a failed installation.

Audit:

Assuming you are installing the PostgreSQL binary package from the PGDG repository, the standard method, as `root`, is to instantiate the cluster thusly:

```
# whoami
root
# PGSETUP_INITDB_OPTIONS="-k" /usr/pgsql-12/bin/postgresql-12-setup initdb
Initializing database ... OK

#
```

A correctly installed data-cluster possesses directory permissions similarly to the following example. Otherwise, the service will fail to start:

```
# whoami
root
# ls -la ~postgres/12
total 8
drwx------.  4 postgres postgres   51 Oct  4 14:01 .
drwx------.  3 postgres postgres   37 Oct  4 13:54 ..
drwx------.  2 postgres postgres    6 Oct  1 06:18 backups
drwx------. 20 postgres postgres 4096 Oct  4 14:01 data
```

```
-rw-------.  1 postgres postgres  923 Oct  4 14:01 initdb.log
#
```

You can verify the PGDATA has sane permissions and attributes by running:

```
$ whoami
postgres
$ /usr/pgsql-12/bin/postgresql-12-check-db-dir ~postgres/12/data
$ echo $?
0
```

As long as the return code is zero(`0`), as shown, everything is fine.

Remediation:

Attempting to instantiate a data cluster to an existing non-empty directory will fail:

```
# whoami
root
# PGSETUP_INITDB_OPTIONS="-k" /usr/pgsql-12/bin/postgresql-12-setup initdb
Data directory is not empty!
```

In the case of a cluster instantiation failure, one must delete/remove the entire data cluster directory and repeat the `initdb` command:

```
# whoami
root
# rm -rf ~postgres/12
# PGSETUP_INITDB_OPTIONS="-k" /usr/pgsql-12/bin/postgresql-12-setup initdb
Initializing database ... OK
```

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists
All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists
Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

2 Directory and File Permissions

This section provides guidance on securing all operating system specific objects for PostgreSQL.

2.1 Ensure the file permissions mask is correct (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Files are always created using a default set of permissions. File permissions can be restricted by applying a permissions mask called the **`umask`**. The `postgres` user account should use a umask of `077` to deny file access to all user accounts except the owner.

Rationale:

The Linux OS defaults the umask to `002`, which means the owner and primary group can read and write the file, and other accounts are permitted to read the file. Not explicitly setting the umask to a value as restrictive as `077` allows other users to read, write, or even execute files and scripts created by the `postgres` user account. The alternative to using a umask is explicitly updating file permissions after file creation using the command line utility `chmod` (a manual and error prone process that is not advised).

Audit:

To view the mask's current setting, execute the following commands:

```
$ whoami
root
$ su - postgres
$ whoami
postgres
$ umask
0022
```

The umask must be `077` or more restrictive for the `postgres` user, otherwise this is a fail.

Remediation:

Depending upon the `postgres` user's environment, the umask is typically set in the initialization file `.bash_profile`, but may also be set in `.profile` or `.bashrc`. To set the umask, add the following to the appropriate profile file:

```
$ whoami
postgres
$ cd ~
$ ls -ld .{bash_profile,profile,bashrc}
ls: cannot access .profile: No such file or directory
ls: cannot access .bashrc: No such file or directory
-rwx------. 1 postgres postgres 267 Aug 14 12:59 .bash_profile
$ echo "umask 077" >> .bash_profile
$ source .bash_profile
$ umask
0077
```

Default Value:

```
0022
```

CIS Controls:

Version 6

14.4 <u>Protect Information With Access Control Lists</u>
All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 <u>Protect Information through Access Control Lists</u>
Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

2.2 Ensure the PostgreSQL pg_wheel group membership is correct (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The group `pg_wheel` is explicitly created on a host where the PostgreSQL server is installed. Membership in this group enables an ordinary user account to gain 'superuser' access to a database cluster by using the `sudo` command (See 'Ensure sudo is configured correctly' later in this benchmark). Only user accounts authorized to have superuser access should be members of the `pg_wheel` group.

Rationale:

Users with unauthorized membership in the `pg_wheel` group can assume the privileges of the owner of the PostgreSQL RDBMS and administer the database, as well as accessing scripts, files, and other executables they should not be able to access.

Audit:

Execute the command `getent` to confirm that a `pg_wheel` group exists. If no such group exists, this is a fail:

```
$ whoami
root
$ # no output (below) means the group does not exist
$ getent group pg_wheel
$
```

If such a group does exist, view its membership and confirm that each user is authorized to act as an administrator;

```
$ whoami
root
$ # when the group exists, the command shows the 'group id' (GID)
$ getent group pg_wheel
pg_wheel:x:502:
$ # since the group exists, list its members thusly
$ awk -F':' '/pg_wheel/{print $4}' /etc/group
$ # empty output == no members
```

Remediation:

If the `pg_wheel` group does not exist, use the following command to create it:

```
$ whoami
root
$ groupadd pg_wheel && getent group pg_wheel
pg_wheel:x:502:
```

Note: that your system's group number may not be `502`. That's OK.

Adding the `postgres` user to the newly created group is done by issuing:

```
$ whoami
root
$ gpasswd -a postgres pg_wheel
Adding user postgres to group pg_wheel
$ # verify membership
$ awk -F':' '/pg_wheel/{print $4}' /etc/group
postgres
```

Removing a user account from the 'pg_wheel' group is achieved by executing the following command:

```
$ whoami
root
$ gpasswd -d pg_wheel postgres
Removing user postgres from group pg_wheel
$ # verify the user was removed
$ awk -F':' '/pg_wheel/{print $4}' /etc/group
$
```

References:

1. https://man7.org/linux/man-pages/man1/groups.1.html
2. https://man7.org/linux/man-pages/man8/getent.1.html
3. https://man7.org/linux/man-pages/man8/gpasswd.1.html
4. https://man7.org/linux/man-pages/man8/useradd.8.html
5. https://en.wikipedia.org/wiki/Wheel_%28Unix_term%29

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3 Logging Monitoring And Auditing

This section provides guidance with respect to PostgreSQL's auditing and logging behavior.

3.1 PostgreSQL Logging

This section provides guidance with respect to PostgreSQL's logging behavior *as it applies to security and auditing*. PostgreSQL contains significantly more logging options that are not audit and/or security related (and as such, are not covered herein).

3.1.1 Logging Rationale

Having an audit trail is an important feature of any relational database system. You want enough detail to describe when an event of interest has started and stopped, what the event is/was, the event's cause, and what the event did/is doing to the system.

Ideally, the logged information is in a format permitting further analysis giving us new perspectives and insight.

The PostgreSQL configuration file `postgresql.conf` is where all adjustable parameters can be set. A configuration file is created as part of the data cluster's creation i.e. `initdb`. The configuration file enumerates all tunable parameters and even though most of them are commented out it is understood that they are in fact active and at those very same documented values. The reason that they are commented out is to signify their default values. Uncommenting them will force the server to read these values instead of using the default values.

3.1.2 Ensure the log destinations are set correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

PostgreSQL supports several methods for logging server messages, including `stderr`, `csvlog` and `syslog`. On Windows, `eventlog` is also supported. One or more of these destinations should be set for server log output.

Rationale:

If `log_destination` is not set, then any log messages generated by the core PostgreSQL processes will be lost.

Audit:

Execute the following SQL statement to view the currently active log destinations:

```
postgres=# show log_destination;
 log_destination
-----------------
 stderr
(1 row)
```

The log destinations should comply with your organization's policies on logging. If all the expected log destinations are not set, this is a fail.

Remediation:

Execute the following SQL statements to remediate this setting (in this example, setting the log destination to `csvlog`):

```
postgres=# alter system set log_destination = 'csvlog';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Note: If more than one log destination is to be used, set this parameter to a list of desired log destinations separated by commas (e.g. `'csvlog, stderr'`).

Default Value:

```
stderr
```

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

Notes:

`logging_collector` (detailed in the next section) must be enabled to generate CSV-format log output.

CIS Controls:

Version 6

6.2 Ensure Audit Log Settings Support Appropriate Log Entry Formatting
Validate audit log settings for each hardware device and the software installed on it, ensuring that logs include a date, timestamp, source addresses, destination addresses, and various other useful elements of each packet and/or transaction. Systems should record logs in a standardized format such as syslog entries or those outlined by the Common Event Expression initiative. If systems cannot generate logs in a standardized format, log normalization tools can be deployed to convert logs into such a format.

Version 7

6.2 Activate audit logging
Ensure that local logging has been enabled on all systems and networking devices.

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.3 Ensure the logging collector is enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The logging collector is a background process that captures log messages sent to `stderr` and redirects them into log files. The `logging_collector` setting must be enabled in order for this process to run. It can only be set at server start.

Rationale:

The logging collector approach is often more useful than logging to `syslog`, since some types of messages might not appear in `syslog` output. One common example is dynamic-linker failure message; another may be error messages produced by scripts such as `archive_command`.

Note: This setting *must* be enabled when `log_destination` is either `stderr` or `csvlog` and for certain other logging parameters to take effect.

Audit:

Execute the following SQL statement and confirm that the `logging_collector` is enabled (`on`):

```
postgres=# show logging_collector;
 logging_collector
-------------------
 on
(1 row)
```

Remediation:

Execute the following SQL statement(s) to remediate this setting:

```
postgres=# alter system set logging_collector = 'on';
ALTER SYSTEM
```

Unfortunately, this setting can only be changed at server (re)start. As root, restart the PostgreSQL service for this change to take effect:

```
# whoami
root
# systemctl restart postgresql-12
# systemctl status postgresql-12|grep 'ago$'
   Active: active (running) since <date>; 1s ago
```

Default Value:

```
on
```

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6.2 Ensure Audit Log Settings Support Appropriate Log Entry Formatting
Validate audit log settings for each hardware device and the software installed on it, ensuring that logs include a date, timestamp, source addresses, destination addresses, and various other useful elements of each packet and/or transaction. Systems should record logs in a standardized format such as syslog entries or those outlined by the Common Event Expression initiative. If systems cannot generate logs in a standardized format, log normalization tools can be deployed to convert logs into such a format.

Version 7

6.2 Activate audit logging
Ensure that local logging has been enabled on all systems and networking devices.

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.4 Ensure the log file destination directory is set correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `log_directory` setting specifies the destination directory for log files when `log_destination` is `stderr` or `csvlog`. It can be specified as relative to the cluster data directory (`$PGDATA`) or as an absolute path. `log_directory` should be set according to your organization's logging policy.

Rationale:

If `log_directory` is not set, it is interpreted as the absolute path `'/'` and PostgreSQL will attempt to write its logs there (and typically fail due to a lack of permissions to that directory). This parameter should be set to direct the logs into the appropriate directory location as defined by your organization's logging policy.

Audit:

Execute the following SQL statement to confirm that the expected logging directory is specified:

```
postgres=# show log_directory;
 log_directory
---------------
 log
(1 row)
```

Note: This shows a path relative to cluster's data directory. An absolute path would start with a `/` like the following: `/var/log/pg_log`

Remediation:

Execute the following SQL statement(s) to remediate this setting:

```
postgres=# alter system set log_directory='/var/log/postgres';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

```
postgres=# show log_directory;
 log_directory
---------------
 /var/log/postgres
(1 row)
```

Note: The use of `/var/log/postgres`, above, is an example. This should be set to an appropriate path as defined by your organization's logging requirements. Having said that, it **is** a good idea to have the logs outside of your `PGDATA` directyory so that they are not included by things like `pg_basebackup` or `pgBackRest`.

Default Value:

`log` which is relative to the cluster's data directory (e.g. `/var/lib/pgsql/<majorversion>/data/log`)

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6.2 Ensure Audit Log Settings Support Appropriate Log Entry Formatting
Validate audit log settings for each hardware device and the software installed on it, ensuring that logs include a date, timestamp, source addresses, destination addresses, and various other useful elements of each packet and/or transaction. Systems should record logs in a standardized format such as syslog entries or those outlined by the Common Event Expression initiative. If systems cannot generate logs in a standardized format, log normalization tools can be deployed to convert logs into such a format.

Version 7

6.2 Activate audit logging
Ensure that local logging has been enabled on all systems and networking devices.

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.5 Ensure the filename pattern for log files is set correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `log_filename` setting specifies the filename pattern for log files. The value for `log_filename` should match your organization's logging policy.

The value is treated as a `strftime` pattern, so `%-escapes` can be used to specify time-varying filenames. The supported `%-escapes` are similar to those listed in the Open Group's `strftime` specification. If you specify a filename without escapes, you should plan to use a log rotation utility to avoid eventually filling the partition that contains `log_directory`. If there are any time-zone-dependent `%-escapes`, the computation is done in the zone specified by `log_timezone`. Also, the system's `strftime` is not used directly, so platform-specific (nonstandard) extensions do not work.

If CSV-format output is enabled in `log_destination`, `.csv` will be appended to the log filename. (If `log_filename` ends in `.log`, the suffix is replaced instead.)

Rationale:

If `log_filename` is not set, then the value of `log_directory` is appended to an empty string and PostgreSQL will fail to start as it will try to write to a directory instead of a file.

Audit:

Execute the following SQL statement to confirm that the desired pattern is set:

```
postgres=# show log_filename;
   log_filename
-------------------
 postgresql-%a.log
(1 row)
```

Note: This example shows the use of the `strftime %a` escape. This creates seven logfiles, one for each day of the week (e.g. `postgresql-Mon.log`, `postgresql-Tue.log`, et al)

Remediation:

Execute the following SQL statement(s) to remediate this setting:

```
postgres=# alter system set log_filename='postgresql-%Y%m%d.log';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
postgres=# show log_filename;
   log_filename
-------------------
 postgresql-%Y%m%d.log
(1 row)
```

Note: In this example, a new logfile will be created for each day (e.g. `postgresql-20180901.log`)

Default Value:

The default is `postgresql-%a.log`, which creates a new logfile for each day of the week (e.g. `postgresql-Mon.log`, `postgresql-Tue.log`).

References:

1. https://man7.org/linux/man-pages/man3/strftime.3.html
2. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6.2 Ensure Audit Log Settings Support Appropriate Log Entry Formatting
Validate audit log settings for each hardware device and the software installed on it, ensuring that logs include a date, timestamp, source addresses, destination addresses, and various other useful elements of each packet and/or transaction. Systems should record logs in a standardized format such as syslog entries or those outlined by the Common Event Expression initiative. If systems cannot generate logs in a standardized format, log normalization tools can be deployed to convert logs into such a format.

Version 7

6.2 Activate audit logging
Ensure that local logging has been enabled on all systems and networking devices.

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.6 Ensure the log file permissions are set correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `log_file_mode` setting determines the file permissions for log files when `logging_collector` is enabled. The parameter value is expected to be a numeric mode specification in the form accepted by the `chmod` and `umask` system calls. (To use the customary octal format, the number must start with a `0` (zero).)

The permissions should be set to allow only the necessary access to authorized personnel. In most cases the best setting is `0600`, so that only the server owner can read or write the log files. The other commonly useful setting is `0640`, allowing members of the owner's group to read the files, although to make use of that, you will need to alter the `log_directory` setting to store the log files outside the cluster data directory.

Rationale:

Log files often contain sensitive data. Allowing unnecessary access to log files may inadvertently expose sensitive data to unauthorized personnel.

Audit:

Execute the following SQL statement to verify that the setting is consistent with organizational logging policy:

```
postgres=# show log_file_mode;
 log_file_mode
---------------
 0600
(1 row)
```

Remediation:

Execute the following SQL statement(s) to remediate this setting (with the example assuming a desired value of `0600`):

```
postgres=# alter system set log_file_mode = '0600';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
```

```
-----------------
 t
(1 row)
postgres=# show log_file_mode;
 log_file_mode
----------------
 0600
(1 row)
```

Default Value:

`0600`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists
All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists
Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.1.7 Ensure 'log_truncate_on_rotation' is enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_truncate_on_rotation` setting when `logging_collector` is enabled causes PostgreSQL to truncate (overwrite) existing log files with the same name during log rotation instead of appending to them. For example, using this setting in combination with a `log_filename` setting value like `postgresql-%H.log` would result in generating 24 hourly log files and then cyclically overwriting them:

```
postgresql-00.log

postgresql-01.log

[...]

postgresql-23.log
```

Note: Truncation will occur ***only*** when a new file is being opened due to time-based rotation, not during server startup or size-based rotation (see later in this benchmark for size-based rotation details).

Rationale:

If this setting is disabled, pre-existing log files will be appended to if `log_filename` is configured in such a way that static names are generated.

Enabling or disabling the truncation should only be decided when **also** considering the value of `log_filename` and `log_rotation_age`/`log_rotation_size`. Some examples to illustrate the interaction between these settings:

```
# truncation is moot, as each rotation gets a unique filename (postgresql-
20180605.log)

log_truncate_on_rotation = on

log_filename = 'postgresql-%Y%m%d.log'

log_rotation_age = '1d'

log_rotation_size = 0
```

```
# truncation every hour, losing log data every hour until the date changes

log_truncate_on_rotation = on

log_filename = 'postgresql-%Y%m%d.log'

log_rotation_age = '1h'

log_rotation_size = 0

# no truncation if the date changed while generating 100M of log data,
truncation otherwise

log_truncate_on_rotation = on

log_filename = 'postgresql-%Y%m%d.log'

log_rotation_age = '0'

log_rotation_size = '100M'
```

Audit:

Execute the following SQL statement to verify how `log_truncate_on_rotation` is set:

```
postgres=# show log_truncate_on_rotation;
 log_truncate_on_rotation
--------------------------
 off
(1 row)
```

Remediation:

Execute the following SQL statement(s) to remediate this setting:

```
postgres=# alter system set log_truncate_on_rotation = 'on';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
postgres=# show log_truncate_on_rotation;
 log_truncate_on_rotation
--------------------------
 on
(1 row)
```

Default Value:

`on`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

Notes:

Be sure to consider your organization's logging retention policies and the use of any external log consumption tools before deciding if truncation should be enabled or disabled.

CIS Controls:

Version 6

6.3 Ensure Audit Logging Systems Are Not Subject To Loss (i.e. rotation/archive)
Ensure that all systems that store logs have adequate storage space for the logs generated on a regular basis, so that log files will not fill up between log rotation intervals. The logs must be archived and digitally signed on a periodic basis.

Version 7

6.4 Ensure adequate storage for logs
Ensure that all systems that store logs have adequate storage space for the logs generated.

3.1.8 Ensure the maximum log file lifetime is set correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

When `logging_collector` is enabled, the `log_rotation_age` parameter determines the maximum lifetime of an individual log file (depending on the value of `log_filename`). After this many minutes have elapsed, a new log file will be created via automatic log file rotation. Current best practices advise log rotation *at least* daily, but your organization's logging policy should dictate your rotation schedule.

Rationale:

Log rotation is a standard best practice for log management.

Audit:

Execute the following SQL statement to verify the log rotation age is set to an acceptable value:

```
postgres=# show log_rotation_age;
 log_rotation_age
------------------
 1d
```

Remediation:

Execute the following SQL statement(s) to remediate this setting (in this example, setting it to one hour):

```
postgres=# alter system set log_rotation_age='1h';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`1d` (one day)

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6.3 Ensure Audit Logging Systems Are Not Subject To Loss (i.e. rotation/archive)
Ensure that all systems that store logs have adequate storage space for the logs generated on a regular basis, so that log files will not fill up between log rotation intervals. The logs must be archived and digitally signed on a periodic basis.

Version 7

6.4 Ensure adequate storage for logs
Ensure that all systems that store logs have adequate storage space for the logs generated.

3.1.9 Ensure the maximum log file size is set correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `log_rotation_size` setting determines the maximum size of an individual log file. Once the maximum size is reached, automatic log file rotation will occur.

Rationale:

If this is set to zero, size-triggered creation of new log files is disabled. This will prevent automatic log file rotation when files become too large, which could put log data at increased risk of loss (unless age-based rotation is configured).

Audit:

Execute the following SQL statement to verify that `log_rotation_size` is set in compliance with the organization's logging policy:

```
postgres=# show log_rotation_size;
 log_rotation_size
-------------------
 1GB
(1 row)
```

Remediation:

Execute the following SQL statement(s) to remediate this setting (in this example, setting it to `1GB`):

```
postgres=# alter system set log_rotation_size = '1GB';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`0`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6.3 Ensure Audit Logging Systems Are Not Subject To Loss (i.e. rotation/archive)
Ensure that all systems that store logs have adequate storage space for the logs generated on a regular basis, so that log files will not fill up between log rotation intervals. The logs must be archived and digitally signed on a periodic basis.

Version 7

6.4 Ensure adequate storage for logs
Ensure that all systems that store logs have adequate storage space for the logs generated.

3.1.10 Ensure the correct syslog facility is selected (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `syslog_facility` setting specifies the syslog "facility" to be used when logging to `syslog` is enabled. You can choose from any of the 'local' facilities:

- `LOCAL0`
- `LOCAL1`
- `LOCAL2`
- `LOCAL3`
- `LOCAL4`
- `LOCAL5`
- `LOCAL6`
- `LOCAL7`

Your organization's logging policy should dictate which facility to use based on the `syslog` daemon in use.

Rationale:

If not set to the appropriate facility, the PostgreSQL log messages may be intermingled with other applications' log messages, incorrectly routed, or potentially dropped (depending on your `syslog` configuration).

Audit:

Execute the following SQL statement and verify that the correct facility is selected:

```
postgres=# show syslog_facility;
 syslog_facility
-----------------
 local0
(1 row)
```

Remediation:

Execute the following SQL statement(s) to remediate this setting (in this example, setting it to the `LOCAL1` facility):

```
postgres=# alter system set syslog_facility = 'LOCAL1';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`LOCAL0`

References:

1. https://tools.ietf.org/html/rfc3164#section-4.1.1
2. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.2 Activate audit logging
Ensure that local logging has been enabled on all systems and networking devices.

3.1.11 Ensure the program name for PostgreSQL syslog messages is correct (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `syslog_ident` setting specifies the program name used to identify PostgreSQL messages in syslog logs. An example of a possible program name is `postgres`.

Rationale:

If this is not set correctly, it may be difficult or impossible to distinguish PostgreSQL messages from other messages in syslog logs.

Audit:

Execute the following SQL statement to verify the program name is set correctly:

```
postgres=# show syslog_ident;
 syslog_ident
--------------
 postgres
(1 row)
```

Remediation:

Execute the following SQL statement(s) to remediate this setting (in this example, assuming a program name of `proddb`):

```
postgres=# alter system set syslog_ident = 'proddb';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
postgres=# show syslog_ident;
 syslog_ident
--------------
 proddb
(1 row)
```

Default Value:

```
postgres
```

References:

1. https://tools.ietf.org/html/rfc3164#section-4.1.3
2. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.12 Ensure the correct messages are written to the server log (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `log_min_messages` setting specifies the message levels that are written to the server log. Each level includes all the levels that follow it. The lower the level (vertically, below), the fewer messages are sent.

Valid values are:

- `DEBUG5` <-- exceedingly chatty
- `DEBUG4`
- `DEBUG3`
- `DEBUG2`
- `DEBUG1`
- `INFO`
- `NOTICE`
- `WARNING`
- `ERROR`
- `LOG`
- `FATAL`
- `PANIC` <-- practically mute

`WARNING` is considered the best practice unless indicated otherwise by your organization's logging policy.

Rationale:

If this is not set to the correct value, too many messages or too few messages may be written to the server log.

Audit:

Execute the following SQL statement to confirm the setting is correct:

```
postgres=# show log_min_messages;
 log_min_messages
------------------
```

```
 warning
(1 row)
```

Remediation:

Execute the following SQL statement(s) as superuser to remediate this setting (in this example, to set it to `warning`):

```
postgres=# alter system set log_min_messages = 'warning';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`WARNING`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.4 Ensure adequate storage for logs
Ensure that all systems that store logs have adequate storage space for the logs generated.

3.1.13 Ensure the correct SQL statements generating errors are recorded (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL

- Level 1 - PostgreSQL on Linux

Description:

The `log_min_error_statement` setting causes all SQL statements generating errors at or above the specified severity level to be recorded in the server log. Each level includes all the levels that follow it. The lower the level (vertically, below), the fewer messages are recorded. Valid values are:

- `DEBUG5` <-- exceedingly chatty
- `DEBUG4`
- `DEBUG3`
- `DEBUG2`
- `DEBUG1`
- `INFO`
- `NOTICE`
- `WARNING`
- `ERROR`
- `LOG`
- `FATAL`
- `PANIC` <-- practically mute

`ERROR` is considered the best practice setting. Changes should only be made in accordance with your organization's logging policy.

Note: To effectively turn off logging of failing statements, set this parameter to `PANIC`.

Rationale:

If this is not set to the correct value, too many erring SQL statements or too few erring SQL statements may be written to the server log.

Audit:

Execute the following SQL statement to verify the setting is correct:

```
postgres=# show log_min_error_statement;
 log_min_error_statement
```

```
-------------------------
 error
(1 row)
```

Remediation:

Execute the following SQL statement(s) as superuser to remediate this setting (in the example, to `error`):

```
postgres=# alter system set log_min_error_statement = 'error';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`ERROR`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.4 Ensure adequate storage for logs
Ensure that all systems that store logs have adequate storage space for the logs generated.

3.1.14 Ensure 'debug_print_parse' is disabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `debug_print_parse` setting enables printing the resulting parse tree for each executed query. These messages are emitted at the `LOG` message level. Unless directed otherwise by your organization's logging policy, it is recommended this setting be disabled by setting it to `off`.

Rationale:

Enabling any of the `DEBUG` printing variables may cause the logging of sensitive information that would otherwise be omitted based on the configuration of the other logging settings.

Audit:

Execute the following SQL statement to confirm the setting is correct:

```
postgres=# show debug_print_parse;
 debug_print_parse
-------------------
 off
(1 row)
```

Remediation:

Execute the following SQL statement(s) to remediate this setting:

```
postgres=# alter system set debug_print_parse='off';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`off`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

3.1.15 Ensure 'debug_print_rewritten' is disabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL

- Level 1 - PostgreSQL on Linux

Description:

The `debug_print_rewritten` setting enables printing the query rewriter output for each executed query. These messages are emitted at the `LOG` message level. Unless directed otherwise by your organization's logging policy, it is recommended this setting be disabled by setting it to `off`.

Rationale:

Enabling any of the `DEBUG` printing variables may cause the logging of sensitive information that would otherwise be omitted based on the configuration of the other logging settings.

Audit:

Execute the following SQL statement to confirm the setting is disabled:

```
postgres=# show debug_print_rewritten;
 debug_print_rewritten
-----------------------
 off
(1 row)
```

Remediation:

Execute the following SQL statement(s) to disable this setting:

```
postgres=# alter system set debug_print_rewritten = 'off';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`off`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

3.1.16 Ensure 'debug_print_plan' is disabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL

- Level 1 - PostgreSQL on Linux

Description:

The `debug_print_plan` setting enables printing the execution plan for each executed query. These messages are emitted at the `LOG` message level. Unless directed otherwise by your organization's logging policy, it is recommended this setting be disabled by setting it to `off`.

Rationale:

Enabling any of the `DEBUG` printing variables may cause the logging of sensitive information that would otherwise be omitted based on the configuration of the other logging settings.

Audit:

Execute the following SQL statement to verify the setting is disabled:

```
postgres=# show debug_print_plan ;
 debug_print_plan
------------------
 off
(1 row)
```

Remediation:

Execute the following SQL statement(s) to disable this setting:

```
postgres=# alter system set debug_print_plan = 'off';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`off`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

3.1.17 Ensure 'debug_pretty_print' is enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

Enabling `debug_pretty_print` indents the messages produced by `debug_print_parse`, `debug_print_rewritten`, or `debug_print_plan` making them significantly easier to read.

Rationale:

If this setting is disabled, the "compact" format is used instead, significantly reducing readability of the `DEBUG` statement log messages.

Audit:

Execute the following SQL statement to confirm the setting is enabled:

```
postgres=# show debug_pretty_print ;
 debug_pretty_print
--------------------
 on
(1 row)
```

Remediation:

Execute the following SQL statement(s) to enable this setting:

```
postgres=# alter system set debug_pretty_print = 'on';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Impact:

Be advised that the aforementioned `DEBUG` printing options are **disabled**, but if your organizational logging policy requires them to be `on` then this option comes into play.

Default Value:

`on`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.18 Ensure 'log_connections' is enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_connections` setting causes each attempted connection to the server to be logged, as well as successful completion of client authentication. This parameter cannot be changed after session start.

Rationale:

PostgreSQL does not maintain an internal record of attempted connections to the database for later auditing. It is only by enabling the logging of these attempts that one can determine if unexpected attempts are being made.

Note that enabling this without also enabling `log_disconnections` provides little value. Generally, you would enable/disable the pair together.

Audit:

Execute the following SQL statement to verify the setting is enabled:

```
postgres=# show log_connections;
 log_connections
-----------------
 on
(1 row)
```

Remediation:

Execute the following SQL statement(s) to enable this setting:

```
postgres=# alter system set log_connections = 'on';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`off`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.19 Ensure 'log_disconnections' is enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL

- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_disconnections` setting logs the end of each session, including session duration. This parameter cannot be changed after session start.

Rationale:

PostgreSQL does not maintain the beginning or ending of a connection internally for later review. It is only by enabling the logging of these that one can examine connections for failed attempts, 'over long' duration, or other anomalies.

Note that enabling this without also enabling `log_connections` provides little value. Generally, you would enable/disable the pair together.

Audit:

Execute the following SQL statement to verify the setting is enabled:

```
postgres=# show log_disconnections;
 log_disconnections
--------------------
 on
(1 row)
```

Remediation:

Execute the following SQL statement(s) to enable this setting:

```
postgres=# alter system set log_disconnections = 'on';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`off`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.20 Ensure 'log_error_verbosity' is set correctly (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL

- Level 1 - PostgreSQL on Linux

Description:

The `log_error_verbosity` setting specifies the verbosity (amount of detail) of logged messages. Valid values are:

- `TERSE`
- `DEFAULT`
- `VERBOSE`

with each containing the fields of the level above it as well as additional fields.

`TERSE` excludes the logging of `DETAIL`, `HINT`, `QUERY`, and `CONTEXT` error information.

`VERBOSE` output includes the `SQLSTATE`, error code, and the source code file name, function name, and line number that generated the error.

The appropriate value should be set based on your organization's logging policy.

Rationale:

If this is not set to the correct value, too many details or too few details may be logged.

Audit:

Execute the following SQL statement to verify the setting is correct:

```
postgres=# show log_error_verbosity ;
 log_error_verbosity
---------------------
 default
(1 row)
```

Remediation:

Execute the following SQL statement(s) as superuser to remediate this setting (in this example, to `verbose`):

```
postgres=# alter system set log_error_verbosity = 'verbose';
ALTER SYSTEM
```

```
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`DEFAULT`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.21 Ensure 'log_hostname' is set correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_hostname` setting causes the hostname of the connecting host to be logged **in addition** to the host's IP address for connection log messages. Disabling the setting causes only the connecting host's IP address to be logged, and not the hostname. Unless your organization's logging policy requires hostname logging, it is best to disable this setting so as not to incur the overhead of DNS resolution for each statement that is logged.

Rationale:

Depending on your hostname resolution setup, enabling this setting might impose a non-negligible performance penalty. Additionally, the IP addresses that are logged can be resolved to their DNS names when reviewing the logs (unless dynamic host names are being used as part of your DHCP setup).

Audit:

Execute the following SQL statement to verify the setting is correct:

```
postgres=# show log_hostname;
 log_hostname
--------------
 off
(1 row)
```

Remediation:

Execute the following SQL statement(s) to remediate this setting (in this example, to `off`):

```
postgres=# alter system set log_hostname='off';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

```
off
```

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

3.1.22 Ensure 'log_line_prefix' is set correctly (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `log_line_prefix` setting specifies a `printf`-style string that is prefixed to each log line. If blank, no prefix is used. You should configure this as recommended by the pgBadger development team unless directed otherwise by your organization's logging policy.

`%` characters begin "escape sequences" that are replaced with status information as outlined below. Unrecognized escapes are ignored. Other characters are copied straight to the log line. Some escapes are only recognized by session processes and will be treated as empty by background processes such as the main server process. Status information may be aligned either left or right by specifying a numeric literal after the `%` and before the option. A negative value will cause the status information to be padded on the right with spaces to give it a minimum width, whereas a positive value will pad on the left. Padding can be useful to aid human readability in log files.

Any of the following escape sequences can be used:

```
Escape  Effect                                              Session only
%a      Application name                                    yes
%u      User name                                           yes
%d      Database name                                       yes
%r      Remote host name or IP address, and remote port    yes
%h      Remote host name or IP address                      yes
%p      Process ID                                          no
%t      Time stamp without milliseconds                     no
%m      Time stamp with milliseconds                        no
%i      Command tag: type of session's current command     yes
%e      SQLSTATE error code                                 no
%c      Session ID: see below                               no
%l      Number of the log line for each session
        or process, starting at 1                           no
%s      Process start time stamp                            no
%v      Virtual transaction ID (backendID/localXID)         no
%x      Transaction ID (0 if none is assigned)              no
%q      Produces no output, but tells non-session
        processes to stop at this point in the string;
        ignored by session processes                        no
%%      Literal %
```

Rationale:

Properly setting `log_line_prefix` allows for adding additional information to each log entry (such as the user, or the database). Said information may then be of use in auditing or security reviews.

Audit:

Execute the following SQL statement to verify the setting is correct:

```
postgres=# show log_line_prefix;
 log_line_prefix
-----------------
 < %m >
(1 row)
```

Remediation:

Execute the following SQL statement(s) to remediate this setting:

```
postgres=# alter system set log_line_prefix = '%m [%p]: [%l-1]
db=%d,user=%u,app=%a,client=%h ';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`%m [%p]`

References:

1. https://pgbadger.darold.net/
2. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.3 Enable Detailed Logging

Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.23 Ensure 'log_statement' is set correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `log_statement` setting specifies the types of SQL statements that are logged. Valid values are:

- `none` (off)
- `ddl`
- `mod`
- `all` (all statements)

It is recommended this be set to `ddl` unless otherwise directed by your organization's logging policy.

`ddl` logs all data definition statements:

- `CREATE`
- `ALTER`
- `DROP`

`mod` logs all `ddl` statements, plus data-modifying statements:

- `INSERT`
- `UPDATE`
- `DELETE`
- `TRUNCATE`
- `COPY FROM`

(`PREPARE`, `EXECUTE`, and `EXPLAIN ANALYZE` statements are also logged if their contained command is of an appropriate type.)

For clients using extended query protocol, logging occurs when an Execute message is received, and values of the Bind parameters are included (with any embedded single-quote marks doubled).

Rationale:

Setting `log_statement` to align with your organization's security and logging policies facilitates later auditing and review of database activities.

Audit:

Execute the following SQL statement to verify the setting is correct:

```
postgres=# show log_statement;
 log_statement
---------------
 none
(1 row)
```

If `log_statement` is set to `none` then this is a fail.

Remediation:

Execute the following SQL statement(s) as superuser to remediate this setting:

```
postgres=# alter system set log_statement='ddl';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`none`

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.1.24 Ensure 'log_timezone' is set correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The `log_timezone` setting specifies the time zone to use in timestamps within log messages. This value is cluster-wide, so that all sessions will report timestamps consistently. Unless directed otherwise by your organization's logging policy, set this to either `GMT` or `UTC`.

Rationale:

Log entry timestamps should be configured for an appropriate time zone as defined by your organization's logging policy to ensure a lack of confusion around when a logged event occurred.

Note that this setting affects only the timestamps present in the logs. It does not affect the time zone in use by the database itself (for example, `select now()`), nor does it affect the host's time zone.

Audit:

Execute the following SQL statement:

```
postgres=# show log_timezone ;
 log_timezone
--------------
 US/Eastern
(1 row)
```

If `log_timezone` is not set to `GMT`, `UTC`, or as defined by your organization's logging policy this is a fail.

Remediation:

Execute the following SQL statement(s) to remediate this setting:

```
postgres=# alter system set log_timezone = 'GMT';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
```

```
 t
(1 row)
```

Default Value:

By default, the PGDG packages will set this to match the server's timezone in the Operating System.

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-logging.html
2. https://en.wikipedia.org/wiki/Time_zone

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.3 Enable Detailed Logging
Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

3.2 Ensure the PostgreSQL Audit Extension (pgAudit) is enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

The PostgreSQL Audit Extension (pgAudit) provides detailed session and/or object audit logging via the standard PostgreSQL logging facility. The goal of pgAudit is to provide PostgreSQL users with the capability to produce audit logs often required to comply with government, financial, or ISO certifications.

Rationale:

Basic statement logging can be provided by the standard logging facility with `log_statement = all`. This is acceptable for monitoring and other uses but does not provide the level of detail generally required for an audit. It is not enough to have a list of all the operations performed against the database, it must also be possible to find particular statements that are of interest to an auditor. The standard logging facility shows what the user requested, while pgAudit focuses on the details of what happened while the database was satisfying the request.

When logging `SELECT` and `DML` statements, pgAudit can be configured to log a separate entry for each relation referenced in a statement. No parsing is required to find all statements that touch a particular table. In fact, the goal is that the statement text is provided primarily for deep forensics and should not be required for an audit.

Audit:

First, as the database administrator (shown here as "postgres"), verify pgaudit is enabled by running the following commands:

```
postgres=# show shared_preload_libraries ;
 shared_preload_libraries
--------------------------
pgaudit
(1 row)
```

If the output does not contain "pgaudit", this is a fail.

Next, verify that desired auditing components are enabled:

```
postgres=# show pgaudit.log;
ERROR:  unrecognized configuration parameter "pgaudit.log"
```

If the output does not contain the desired auditing components, this is a fail.
The list below summarizes `pgAudit.log` components:

- 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`.

Remediation:

To install and enable pgAudit, simply install the appropriate rpm from the PGDG repo:

```
# whoami
root
[root@centos7 ~]# dnf -y install pgaudit14_12
Last metadata expiration check: 0:09:08 ago on Mon 28 Oct 2019 11:23:30 AM
EDT.
Dependencies resolved.
[snip]
Installed:
  pgaudit14_12-1.4.0-1.rhel8.x86_64

Complete!
```

pgAudit is now installed and ready to be configured. Next, we need to alter the `postgresql.conf` configuration file to:

- enable pgAudit as an extension in the `shared_preload_libraries` parameter
- indicate which classes of statements we want to log via the `pgaudit.log` parameter

and, finally, restart the PostgreSQL service:

```
$ vi ${PGDATA}/postgresql.conf
```

Find the `shared_preload_libraries` entry, and add 'pgaudit' to it (preserving any existing entries):

```
shared_preload_libraries = 'pgaudit'

OR
```

```
shared_preload_libraries = 'pgaudit,somethingelse'
```

Now, add a new `pgaudit`-specific entry:

```
# for this example we are logging the ddl and write operations
pgaudit.log='ddl,write'
```

Restart the PostgreSQL server for changes to take affect:

```
# whoami
root
# systemctl restart postgresql-12
# systemctl status postgresql-12|grep 'ago$'
   Active: active (running) since [date] 10s ago
#
```

Impact:

Depending on settings, it is possible for pgAudit to generate an *enormous volume of logging*. Be careful to determine exactly what needs to be audit logged in your environment to avoid logging too much.

References:

1. https://www.pgaudit.org/

Notes:

pgAudit versions relate to PostgreSQL major versions; ensure you install the pgAudit package that matches your PostgreSQL version.

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6.2 Activate audit logging
Ensure that local logging has been enabled on all systems and networking devices.

4 User Access and Authorization

The capability to use database resources at a given level, or user authorization rules, allows for user manipulation of the various parts of the PostgreSQL database. These authorizations must be structured to block unauthorized use and/or corruption of vital data and services by setting restrictions on user capabilities.

4.1 Ensure sudo is configured correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

It is common to have more than one authorized individual administering the PostgreSQL service at the Operating System level. It is also quite common to permit login privileges to individuals on a PostgreSQL host who otherwise are not authorized to access the server's data cluster and files. Administering the PostgreSQL data cluster, as opposed to its data, is to be accomplished via a localhost login of a regular UNIX user account. Access to the `postgres` superuser account is restricted in such a manner as to interdict unauthorized access. `sudo` satisfies the requirements by escalating ordinary user account privileges as the PostgreSQL RDBMS superuser.

Rationale:

Without `sudo`, there would not be capabilities to strictly control access to the superuser account and to securely and authoritatively audit its use.

Audit:

Log in as an Operating System user authorized to escalate privileges and test the `sudo` invocation by executing the following:

```
$ whoami
user1
$ groups
user1
$ sudo su - postgres
[sudo] password for user1:
user1 is not in the sudoers file. This incident will be reported.
```

As shown above, `user1` has not been added to the `/etc/sudoers` file or made a member of any group listed in the `/etc/sudoers` file. Whereas:

```
$ whoami
user2
$ groups
user2 pg_wheel
$ sudo su - postgres
[sudo] password for user2:
$ whoami
postgres
```

shows the `user2` user is configured properly for `sudo` access.

Remediation:

As superuser `root`, execute the following commands:

```
# echo '%pg_wheel ALL= /bin/su - postgres' > /etc/sudoers.d/postgres
# chmod 600 /etc/sudoers.d/postgres
```

This grants any Operating System user that is a member of the `pg_wheel` group to use `sudo su - postgres` to become the `postgres` user.

Ensure that all Operating System user's that need such access are members of the group as detailed earlier in this benchmark.

References:

1. https://www.sudo.ws/man/1.8.15/sudo.man.html
2. https://www.sudo.ws/man/1.8.17/visudo.man.html

CIS Controls:

Version 6

5.8 Administrators Should Not Directly Log In To A System (i.e. use RunAs/sudo)
Administrators should be required to access a system using a fully logged and non-administrative account. Then, once logged on to the machine without administrative privileges, the administrator should transition to administrative privileges using tools such as Sudo on Linux/UNIX, RunAs on Windows, and other similar facilities for other types of systems.

Version 7

4.3 Ensure the Use of Dedicated Administrative Accounts
Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.

4.2 Ensure excessive administrative privileges are revoked (Scored)

Profile Applicability:

- Level 1 - PostgreSQL

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. 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`)

Further, it has become best practice to create separate roles for DDL and DML. Given an application called 'payroll', one would create the following users:

- `payroll_owner`
- `payroll_user`

Any DDL privileges would be granted to the `payroll_owner` account only, while DML privileges would be given to the `payroll_user` account only. This prevents accidental creation/altering/dropping of database objects by application code that run as the `payroll_user` account.

Rationale:

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

Audit:

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 (RLS):

```
$ whoami
postgres
$ psql -c "\du postgres"
```

```
                              List of roles
Role name |                    Attributes                   | Member of
----------+-------------------------------------------------+-----------
postgres  | Superuser, Create role, Create DB, Replication, | {}
          | Bypass RLS                                      |
```

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 fail.

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

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

NOTE Using `\du *` will show all the default PostgreSQL roles (e.g. `pg_monitor`) as well as any 'normal' roles. This is expected, and should not be cause for alarm.

Remediation:

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`:

```
$ whoami
postgres
$ 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
```

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

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

References:

1. https://www.postgresql.org/docs/12/static/sql-revoke.html
2. https://www.postgresql.org/docs/12/static/sql-createrole.html
3. https://www.postgresql.org/docs/12/static/sql-alterrole.html

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges
Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

4.3 Ensure excessive function privileges are revoked (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

In certain situations, to provide required functionality, PostgreSQL needs to execute internal logic (stored procedures, functions, triggers, etc.) and/or external code modules with elevated privileges. However, if the privileges required for execution are at a higher level than the privileges assigned to organizational users invoking the functionality applications/programs, those users are indirectly provided with greater privileges than assigned by their organization. This is known as privilege elevation. Privilege elevation must be utilized only where necessary. Execute privileges for application functions should be restricted to authorized users only.

Rationale:

Ideally, all application source code should be vetted to validate interactions between the application and the logic in the database, but this is usually not possible or feasible with available resources even if the source code is available. The DBA should attempt to obtain assurances from the development organization that this issue has been addressed and should document what has been discovered. The DBA should also inspect all application logic stored in the database (in the form of functions, rules, and triggers) for excessive privileges.

Audit:

Functions in PostgreSQL can be created with the `SECURITY DEFINER` option. When `SECURITY DEFINER` functions are executed by a user, said function is run with the privileges of the user who **created** it, not the user who is *running* it.

To list all functions that have `SECURITY DEFINER`, run the following SQL:

```
$ whoami
root
$ sudo su - postgres
$ psql -c "SELECT nspname, proname, proargtypes, prosecdef, rolname,
proconfig FROM pg_proc p JOIN pg_namespace n ON p.pronamespace = n.oid JOIN
pg_authid a ON a.oid = p.proowner WHERE prosecdef OR NOT proconfig IS NULL;"
```

In the query results, a `prosecdef` value of '`t`' on a row indicates that that function uses privilege elevation.

If elevation of PostgreSQL privileges is utilized but not documented, this is a fail.

If elevation of PostgreSQL privileges is documented, but not implemented as described in the documentation, this is a fail.

If the privilege-elevation logic can be invoked in ways other than intended, or in contexts other than intended, or by subjects/principals other than intended, this is a fail.

Remediation:

Where possible, revoke `SECURITY DEFINER` on PostgreSQL functions. To change a `SECURITY DEFINER` function to `SECURITY INVOKER`, run the following SQL:

```
$ whoami
root
$ sudo su - postgres
$ psql -c "ALTER FUNCTION [functionname] SECURITY INVOKER;"
```

If it is not possible to revoke `SECURITY DEFINER`, ensure the function can be executed by only the accounts that absolutely need such functionality:

```
postgres=# SELECT proname, proacl FROM pg_proc WHERE proname =
'delete_customer';
     proname     |                         proacl
-----------------+--------------------------------------------------------
 delete_customer | {=X/postgres,postgres=X/postgres,appwriter=X/postgres}
(1 row)
postgres=# REVOKE EXECUTE ON FUNCTION delete_customer(integer,boolean) FROM
appreader;
REVOKE
postgres=# SELECT proname, proacl FROM pg_proc WHERE proname =
'delete_customer';
     proname     |                         proacl
-----------------+--------------------------------------------------------
 delete_customer | {=X/postgres,postgres=X/postgres}
(1 row)
```

Based on output above, `appreader=X/postgres` no longer exists in the `proacl` column results returned from query and confirms `appreader` is no longer granted execute privilege on the function.

References:

1. https://www.postgresql.org/docs/12/static/catalog-pg-proc.html
2. https://www.postgresql.org/docs/12/static/sql-grant.html
3. https://www.postgresql.org/docs/12/static/sql-revoke.html
4. https://www.postgresql.org/docs/12/static/sql-createfunction.html

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges
Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

4.4 Ensure excessive DML privileges are revoked (Scored)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

DML (insert, update, delete) operations at the table level should be restricted to only authorized users. PostgreSQL manages table level DML permissions via the GRANT statement.

Rationale:

Excessive DML grants can lead to unprivileged users changing or deleting information without proper authorization.

Audit:

To audit excessive DML privileges, take an inventory of all users defined in the cluster using the `\du+ *` SQL command, as well as all tables defined in the database using the `\dt *.*` SQL command. Furthermore, the intersection matrix of tables and user grants can be obtained by querying system catalogs `pg_tables` and `pg_user`. Note that in PostgreSQL, users are defined cluster-wide across all databases, while schemas and tables are specific to a particular database. Therefore, the commands below should be executed for each defined database in the cluster. With this information, inspect database table grants and determine if any are excessive for defined database users.

```
postgres=# -- display all users defined in the cluster
postgres=# \x
Expanded display is on.
postgres=# \du+ *

List of roles
-[ RECORD 1 ]----------------------------------------------------------
Role name   | pg_signal_backend
Attributes  | Cannot login
Member of   | {}
Description |
-[ RECORD 2 ]----------------------------------------------------------
Role name   | postgres
Attributes  | Superuser, Create role, Create DB, Replication, Bypass RLS
Member of   | {}
Description |

postgres=# -- display all schema.tables created in current database
postgres=# \x
Expanded display is off.
```

```
postgres=# \dt+ *.*
                                    List of relations
      Schema       |           Name           | Type  |  Owner   |    Size
|
-------------------+-------------------------+-------+----------+------------
+
information_schema | sql_features            | table | postgres | 96 kB
|
information_schema | sql_implementation_info | table | postgres | 48 kB
|
information_schema | sql_languages           | table | postgres | 48 kB
|
information_schema | sql_packages            | table | postgres | 48 kB
|
information_schema | sql_parts               | table | postgres | 48 kB
|
information_schema | sql_sizing              | table | postgres | 48 kB
|
information_schema | sql_sizing_profiles     | table | postgres | 8192 bytes
|
(snip)

postgres=# -- query all tables and user grants in current database
postgres=# -- the system catalogs 'information_schema' and 'pg_catalog' are
excluded
postgres=# select t.schemaname, t.tablename, u.usename,
     has_table_privilege(u.usename, t.tablename, 'select') as select,
     has_table_privilege(u.usename, t.tablename, 'insert') as insert,
     has_table_privilege(u.usename, t.tablename, 'update') as update,
     has_table_privilege(u.usename, t.tablename, 'delete') as delete
from  pg_tables t, pg_user u
where t.schemaname not in ('information_schema','pg_catalog');

 schemaname | tablename | usename | select | insert | update | delete
------------+-----------+---------+--------+--------+--------+--------
(0 rows)
```

For the example below, we illustrate using a single table `customer` and two application users `appwriter` and `appreader`. The intention is for `appwriter` to have full select, insert, update, and delete rights and for `appreader` to only have select rights. We can query these privileges with the example below using the `has_table_privilege` function and filtering for just the table and roles in question.

```
postgres=# select t.tablename, u.usename,
       has_table_privilege(u.usename, t.tablename, 'select') as select,
       has_table_privilege(u.usename, t.tablename, 'insert') as insert,
       has_table_privilege(u.usename, t.tablename, 'update') as update,
       has_table_privilege(u.usename, t.tablename, 'delete') as delete
from   pg_tables t, pg_user u
where  t.tablename = 'customer'
and    u.usename in ('appwriter','appreader');

tablename |  usename  | select | insert | update | delete
----------+-----------+--------+--------+--------+--------
```

```
customer   | appwriter | t        | t        | t        | t
customer   | appreader | t        | t        | t        | t
(2 rows)
```

As depicted, both users have full privileges for the customer table. This is a fail.

When inspecting database-wide results for all users and all table grants, employ a comprehensive approach. Collaboration with application developers is paramount to collectively determine only those database users that require specific DML privileges and on which tables.

Remediation:

If a given database user has been granted excessive DML privileges for a given database table, those privileges should be revoked immediately using the `REVOKE` SQL command.

Continuing with the example above, remove unauthorized grants for `appreader` user using the `REVOKE` statement and verify the Boolean values are now false.

```
postgres=# REVOKE INSERT, UPDATE, DELETE ON TABLE customer FROM appreader;
REVOKE

postgres=# select t.tablename, u.usename,
       has_table_privilege(u.usename, t.tablename, 'select') as select,
       has_table_privilege(u.usename, t.tablename, 'insert') as insert,
       has_table_privilege(u.usename, t.tablename, 'update') as update,
       has_table_privilege(u.usename, t.tablename, 'delete') as delete
from   pg_tables t, pg_user u
where  t.tablename = 'customer'
and    u.usename in ('appwriter','appreader');

tablename |  usename   | select | insert | update | delete
----------+-----------+--------+--------+--------+--------
customer   | appwriter | t        | t        | t        | t
customer   | appreader | t        | f        | f        | f
(2 rows)
```

With the publication of CVE-2018-1058, it is also recommended that all privileges be revoked from the `public` schema for all users on all databases:

```
postgres=# REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE
```

Default Value:

The table owner/creator has full privileges; all other users must be explicitly granted access.

References:

1. https://www.postgresql.org/docs/12/static/sql-grant.html
2. https://www.postgresql.org/docs/12/static/sql-revoke.html
3. https://www.postgresql.org/docs/12/static/functions-info.html#functions-info-access-table
4. https://wiki.postgresql.org/wiki/A_Guide_to_CVE-2018-1058:_Protect_Your_Search_Path
5. https://nvd.nist.gov/vuln/detail/CVE-2018-1058

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges
Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

4.5 Use pg_permission extension to audit object permissions (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

Using a PostgreSQL extension called `pg_permissions` it is possible to declare which DB users should have which permissions on a given object and generate a report showing compliance/deviation.

Rationale:

Auditing permissions in a PostgreSQL database can be intimidating given the default manner in which permissions are presented. The `pg_permissions` extension greatly simplifies this presentation and allows the user to declare what permissions should exist and then report on differences from that ideal.

Audit:

See if the `pg_permissions` extension is available for use:

```
postgres=# select * from pg_available_extensions where name =
'pg_permission';
 name | default_version | installed_version | comment
------+-----------------+-------------------+---------
(0 rows)
```

If the extension isn't found, this is a fail.

Remediation:

At this time, `pg_permission` is not packaged by the PGDG packaging team. As such, download the latest from the extension's site, compile it, and then install it:

```
# whoami
root
# dnf -y install postgresql11-devel
Last metadata expiration check: 1:28:07 ago on Mon 28 Oct 2019 11:23:30 AM
EDT.
Dependencies resolved.
[snip]
Installed:
  postgresql12-devel-12.0-1PGDG.rhel8.x86_64                      libicu-
devel-60.2-7.el8.x86_64
```

```
Complete!
# curl -L -o pg_permission_1.1.tgz https://github.com/cybertec-
postgresql/pg_permission/archive/REL_1_1.tar.gz
# tar xf pg_permission_1.1.tgz
# cd pg_permission-REL_1_1/
# which pg_config
/usr/bin/which: no pg_config in (various paths here)
# export PATH=/usr/pgsql-12/bin:$PATH
# which pg_config
/usr/pgsql-12/bin/pg_config
# make install
/usr/bin/mkdir -p '/usr/pgsql-12/share/extension'
/usr/bin/mkdir -p '/usr/pgsql-12/share/extension'
/usr/bin/mkdir -p '/usr/pgsql-12/doc/extension'
/usr/bin/install -c -m 644 .//pg_permissions.control '/usr/pgsql-
12/share/extension/'
/usr/bin/install -c -m 644 .//pg_permissions--*.sql  '/usr/pgsql-
12/share/extension/'
/usr/bin/install -c -m 644 .//README.pg_permissions '/usr/pgsql-
12/doc/extension/'
# su - postgres
bash-4.2$ whoami
postgres
bash-4.2$ psql -c "create extension pg_permissions;"
CREATE EXTENSION
```

Now you need to add entries to `permission_target` that correspond to your *desired permissions*.

Let's assume we have a schema `appschema`, and `appuser` should have `SELECT`, `UPDATE`, `DELETE`, and `INSERT` permissions on all tables and views in that schema:

```
postgres=# INSERT INTO public.permission_target
postgres=#   (id, role_name, permissions,
postgres=#    object_type, schema_name)
postgres=# VALUES
postgres=#   (1, 'appuser', '{SELECT,INSERT,UPDATE,DELETE}',
postgres=#    'TABLE', 'appschema');
INSERT 0 1

postgres=# INSERT INTO public.permission_target
postgres=#   (id, role_name, permissions,
postgres=#    object_type, schema_name)
postgres=# VALUES
postgres=#   (2, 'appuser', '{SELECT,INSERT,UPDATE,DELETE}',
postgres=#    'VIEW', 'appschema');
INSERT 0 1
```

Of course, the user will need the `USAGE` privilege on the schema:

```
postgres=# INSERT INTO public.permission_target
postgres=#   (id, role_name, permissions,i
postgres=#    object_type, schema_name)
```

```
postgres=# VALUES
postgres=#   (3, 'appuser', '{USAGE}',
postgres=#    'SCHEMA', 'appschema');
INSERT 0 1
```

The user also needs `USAGE` privileges on the `appseq` sequence in that schema:

```
postgres=# INSERT INTO public.permission_target
postgres=#   (id, role_name, permissions,
postgres=#    object_type, schema_name, object_name)
postgres=# VALUES
postgres=#   (4, 'appuser', '{USAGE}',
postgres=#    'SEQUENCE', 'appschema', 'appseq');
INSERT 0 1
```

Now we can review which permissions are missing and which additional permissions are granted:

```
postgres=# SELECT * FROM public.permission_diffs();

 missing | role_name | object_type | schema_name | object_name | column_name
| permission
---------+-----------+-------------+-------------+-------------+-------------
+------------
 f       | laurenz   | VIEW        | appschema   | appview     |
| SELECT
 t       | appuser   | TABLE       | appschema   | apptable    |
| DELETE
(2 rows)
```

That means that `appuser` is missing (missing is TRUE) the DELETE privilege on `appschema.apptable` which should be `GRANT`ed, while user `laurenz` has the **additional** `SELECT` privilege on `appschema.appview` (missing is FALSE).

To review the actual permissions on an object, we can use the `_permissions` views:

```
postgres=# SELECT * FROM schema_permissions
postgres=#   WHERE role_name = 'appuser' AND schema_name = 'appschema' AND
granted IS TRUE;

 object_type | role_name | schema_name | object_name | column_name |
permissions | granted
-------------+-----------+-------------+-------------+-------------+---------
----+---------
 SCHEMA      | appuser   | appschema   |             |             | USAGE
| t
 (1 row)
```

For more details and examples, visit the online documentation.

References:

1. https://github.com/cybertec-postgresql/pg_permission

CIS Controls:

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

4.6 Ensure Row Level Security (RLS) is configured correctly (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

In addition to the SQL-standard privilege system available through `GRANT`, tables can have row security policies that restrict, on a per-user basis, which individual rows can be returned by normal queries or inserted, updated, or deleted by data modification commands. This feature is also known as Row Level Security (RLS).

By default, tables do not have any policies, so if a user has access privileges to a table according to the SQL privilege system, all rows within it are equally available for querying or updating. Row security policies can be specific to commands, to roles, or to both. A policy can be specified to apply to `ALL` commands, or to any combination of `SELECT`, `INSERT`, `UPDATE`, or `DELETE`. Multiple roles can be assigned to a given policy, and normal role membership and inheritance rules apply.

If you use RLS and apply restrictive policies to certain users, it is important that the `Bypass RLS` privilege not be granted to any unauthorized users. This privilege overrides RLS-enabled tables and associated policies. Generally, only superusers and elevated users should possess this privilege.

Rationale:

If RLS policies and privileges are not configured correctly, users could perform actions on tables that they are not authorized to perform, such as inserting, updating, or deleting rows.

Audit:

The first step for an organization is to determine which, if any, database tables require RLS. This decision is a matter of business processes and is unique to each organization. To discover which, if any, database tables have RLS enabled, execute the following query. If any table(s) should have RLS policies applied, but do not appear in query results, then this is a finding.

```
postgres=# SELECT oid, relname, relrowsecurity FROM pg_class WHERE
relrowsecurity IS TRUE;
```

For the purpose of this illustration, we will demonstrate the standard example from the PostgreSQL documentation using the `passwd` table and policy example. As of PostgreSQL 9.5, the catalog table `pg_class` provides column `relrowsecurity` to query and determine whether a relation has RLS enabled. Based on results below we can see RLS is not enabled. Assuming this table should be RLS enabled but is not, this is a finding.

```
postgres=# SELECT oid, relname, relrowsecurity FROM pg_class WHERE relname =
'passwd';
  oid  | relname | relrowsecurity
-------+---------+----------------
 24679 | passwd  | f
(1 row)
```

Further inspection of RLS policies are provided via the system catalog `pg_policy`, which records policy details including table OID, policy name, applicable commands, the roles assigned a policy, and the `USING` and `WITH CHECK` clauses. Finally, RLS and associated policies (if implemented) may also be viewed using the standard `psql` display command `\d+ schema.table` which lists RLS information as part of the table description.

Should you implement Row Level Security and apply restrictive policies to certain users, it's imperative that you check each user's role definition via the `psql` display command `\du` and ensure unauthorized users have not been granted `Bypass RLS` privilege as this would override any RLS enabled tables and associated policies. If unauthorized users do have `Bypass RLS` granted then resolve this using the `ALTER ROLE <user> NOBYPASSRLS;` command.

Remediation:

Again, we are using the example from the PostgreSQL documentation using the example `passwd` table. We will create three database roles to illustrate the workings of RLS:

```
postgres=# CREATE ROLE admin;
CREATE ROLE
postgres=# CREATE ROLE bob;
CREATE ROLE
postgres=# CREATE ROLE alice;
CREATE ROLE
```

Now, we will insert known data into the `passwd` table:

```
postgres=# INSERT INTO passwd VALUES
  ('admin','xxx',0,0,'Admin','111-222-3333',null,'/root','/bin/dash');
INSERT 0 1
postgres=# INSERT INTO passwd VALUES
  ('bob','xxx',1,1,'Bob','123-456-7890',null,'/home/bob','/bin/zsh');
INSERT 0 1
postgres=# INSERT INTO passwd VALUES
```

```
  ('alice','xxx',2,1,'Alice','098-765-4321',null,'/home/alice','/bin/zsh');
INSERT 0 1
```

And we will enable RLS on the table:

```
postgres=# ALTER TABLE passwd ENABLE ROW LEVEL SECURITY;
ALTER TABLE
```

Now that RLS is enabled, we need to define one or more policies. Create the administrator policy and allow it access to all rows:

```
postgres=# CREATE POLICY admin_all ON passwd TO admin USING (true) WITH CHECK
(true);
CREATE POLICY
```

Create a policy for normal users to *view* all rows:

```
postgres=# CREATE POLICY all_view ON passwd FOR SELECT USING (true);
CREATE POLICY
```

Create a policy for normal users that allows them to update only their own rows and to limit what values can be set for their login shell:

```
postgres=# CREATE POLICY user_mod ON passwd FOR UPDATE
  USING (current_user = user_name)
  WITH CHECK (
    current_user = user_name AND
    shell IN ('/bin/bash','/bin/sh','/bin/dash','/bin/zsh','/bin/tcsh')
  );
CREATE POLICY
```

Grant all the normal rights on the table to the `admin` user:

```
postgres=# GRANT SELECT, INSERT, UPDATE, DELETE ON passwd TO admin;
GRANT
```

Grant only select access on non-sensitive columns to everyone:

```
postgres=# GRANT SELECT
  (user_name, uid, gid, real_name, home_phone, extra_info, home_dir, shell)
  ON passwd TO public;
GRANT
```

Grant update to only the sensitive columns:

```
postgres=# GRANT UPDATE
  (pwhash, real_name, home_phone, extra_info, shell)
  ON passwd TO public;
GRANT
```

Ensure that no one has been granted `Bypass RLS` inadvertently, by running the `psql` display command `\du+`. If unauthorized users do have `Bypass RLS` granted then resolve this using the `ALTER ROLE <user> NOBYPASSRLS;` command.

You can now verify that 'admin', 'bob', and 'alice' are properly restricted by querying the `passwd` table as each of these roles.

References:

1. https://www.postgresql.org/docs/12/static/ddl-rowsecurity.html
2. https://www.postgresql.org/docs/12/static/sql-alterrole.html

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists
All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists
Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

4.7 Ensure the set_user extension is installed (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

PostgreSQL access to the superuser database role must be controlled and audited to prevent unauthorized access.

Rationale:

Even when reducing and limiting the access to the superuser role as described earlier in this benchmark, it is still difficult to determine who accessed the superuser role and what actions were taken using that role. As such, it is ideal to prevent anyone from logging in as the superuser and forcing them to escalate their role. This model is used at the OS level by the use of `sudo` and should be emulated in the database. The `set_user` extension allows for this setup.

Audit:

Check if the extension is available by querying the `pg_available_extensions` table:

```
postgres=# select * from pg_available_extensions where name = 'set_user';
 name | default_version | installed_version | comment
------+-----------------+-------------------+---------
(0 rows)
```

If the extension is not listed this is a fail.

Remediation:

At the time this benchmark is being written, `set_user` is not available as a package in the PGDG repository. As such, we will build it from source:

```
# whoami
root
# dnf -y install postgresql12-devel
Last metadata expiration check: 1:28:07 ago on Mon 28 Oct 2019 11:23:30 AM
EDT.
Dependencies resolved.
[snip]
Installed:
  postgresql12-devel-12.0-1PGDG.rhel8.x86_64                     libicu-
devel-60.2-7.el8.x86_64
```

```

Complete!
```

```
# dnf -y install epel-release && dnf --enablerepo=* --disablerepo=base-
debuginfo --disablerepo=c8-media-* --disablerepo=pgdg13* install -y llvm-
devel clang-devel ccache
Last metadata expiration check: 0:00:32 ago on Tue 29 Oct 2019 09:29:18 AM
EDT.
Dependencies resolved.
[snip]
Installed:
  epel-release-8-5.el8.noarch

Complete!
Last metadata expiration check: 0:00:37 ago on Tue 29 Oct 2019 09:29:18 AM
EDT.
Dependencies resolved.
[snip]
Installed:
  clang-devel-7.0.1-1.module_el8.0.0+12+30b38a9a.x86_64 llvm-devel-7.0.1-
3.module_el8.0.0+176+9dc62ab1.x86_64
  ccache-3.7.4-1.epel8.playground.x86_64                compiler-rt-7.0.1-
1.module_el8.0.0+12+30b38a9a.x86_64
  libomp-7.0.1-1.module_el8.0.0+12+30b38a9a.x86_64      clang-7.0.1-
1.module_el8.0.0+12+30b38a9a.x86_64
  clang-libs-7.0.1-1.module_el8.0.0+12+30b38a9a.x86_64  clang-tools-extra-
7.0.1-1.module_el8.0.0+12+30b38a9a.x86_64
  cmake-filesystem-3.11.4-3.el8.x86_64                  gcc-c++-8.2.1-
3.5.el8.x86_64
  libstdc++-devel-8.2.1-3.5.el8.x86_64                  llvm-7.0.1-
3.module_el8.0.0+176+9dc62ab1.x86_64
  llvm-libs-7.0.1-3.module_el8.0.0+176+9dc62ab1.x86_64  libatomic-8.2.1-
3.5.el8.x86_64

Complete!

$ curl -L https://codeload.github.com/pgaudit/set_user/tar.gz/REL1_6_2 >
set_user-1.6.2.tgz
$ tar xf set_user-1.6.2.tgz
$ cd set_user-REL1_6_2
$ export PATH=/usr/pgsql-12/bin:$PATH
$ make USE_PGXS=1 install
/usr/lib64/ccache/clang -Wno-ignored-attributes -fno-strict-aliasing -fwrapv
-O2  -I. -I./ -I/usr/pgsql-12/include/server -I/usr/pgsql-12/include/internal
-D_GNU_SOURCE -I/usr/include/libxml2  -I/usr/include -flto=thin -emit-llvm -c
-o set_user.bc set_user.c
/usr/bin/mkdir -p '/usr/pgsql-12/share/extension'
/usr/bin/mkdir -p '/usr/pgsql-12/share/extension'
/usr/bin/mkdir -p '/usr/pgsql-12/lib'
/usr/bin/install -c -m 644 "set_user.h" /usr/pgsql-12/include
/usr/bin/install -c -m 644 .//set_user.control '/usr/pgsql-
12/share/extension/'
/usr/bin/install -c -m 644 .//set_user--1.6.sql .//set_user--1.5--1.6.sql
.//set_user--1.4--1.5.sql .//set_user--1.1--1.4.sql .//set_user--1.0--1.1.sql
'/usr/pgsql-12/share/extension/'
/usr/bin/install -c -m 755  set_user.so '/usr/pgsql-12/lib/'
```

```
/usr/bin/mkdir -p '/usr/pgsql-12/lib/bitcode/set_user'
/usr/bin/mkdir -p '/usr/pgsql-12/lib/bitcode'/set_user/
/usr/bin/install -c -m 644 set_user.bc '/usr/pgsql-
12/lib/bitcode'/set_user/./
cd '/usr/pgsql-12/lib/bitcode' && /usr/bin/llvm-lto -thinlto -thinlto-
action=thinlink -o set_user.index.bc set_user/set_user.bc
$
```

Now that `set_user` is installed, we need to tell PostgreSQL to load its library:

```
$ whoami
root
$ vi ~postgres/12/data/postgresql.conf
$ load set_user libs before anything else
shared_preload_libraries = 'set_user, other_libs'
$ systemctl restart postgresql-12
$ systemctl status postgresql-12|grep 'ago$'
   Active: active (running) since [timestamp]; 1s ago
```

And now, we can install the extension with SQL:

```
postgres=# select * from pg_available_extensions where name = 'set_user';
  name    | default_version | installed_version |                  comment
----------+-----------------+-------------------+-----------------------------
set_user  | 1.6             |                   | similar to SET ROLE but with
          |                 |                   | added logging
(1 row)

postgres=# create extension set_user;
CREATE EXTENSION
postgres=# select * from pg_available_extensions where name = 'set_user';
  name    | default_version | installed_version |                  comment
----------+-----------------+-------------------+-----------------------------
set_user  | 1.6             | 1.6               | similar to SET ROLE but with
          |                 |                   | added logging

(1 row)
```

Now, we use `GRANT` to configure each DBA role to allow it to use the `set_user` functions. In the example below, we will configure my db user `doug`. (You would do this for each DBA's normal user role.)

```
postgres=# grant execute on function set_user(text) to doug;
GRANT
postgres=# grant execute on function set_user_u(text) to doug;
GRANT
```

Connect to PostgreSQL as yourself and verify it works as expected:

```
$ whoami
psql
$ psql -U doug -d postgres
```

```
postgres=> select set_user('postgres');
ERROR:  switching to superuser not allowed
HINT:  Use 'set_user_u' to escalate.
postgres=> select set_user_u('postgres');
 set_user_u
------------
 OK
(1 row)

postgres=# select current_user, session_user;
 current_user | session_user
--------------+--------------
 postgres     | doug
(1 row)

postgres=# select reset_user();
 reset_user
------------
 OK
(1 row)

postgres=> select current_user, session_user;
 current_user | session_user
--------------+--------------
 doug         | doug
(1 row)
```

Once all DBA's normal user accounts have been `GRANT`ed permission, revoke the ability to login as the `postgres` (superuser) user:

```
postgres=# alter user postgres NOLOGIN;
ALTER ROLE
```

Which results in:

```
$ psql
psql: FATAL:  role "postgres" is not permitted to log in
$ psql -U doug -d postgres
psql (11.3)
```

Make sure there are no other roles that are superuser's and can still login:

```
postgres=# 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:

```
postgres=# DROP VIEW IF EXISTS roletree;
NOTICE:  view "roletree" does not exist, skipping
```

```
DROP VIEW
postgres=# CREATE OR REPLACE VIEW roletree AS
postgres-# WITH RECURSIVE
postgres-# roltree AS (
postgres(#   SELECT u.rolname AS rolname,
postgres(#          u.oid AS roloid,
postgres(#          u.rolcanlogin,
postgres(#          u.rolsuper,
postgres(#          '{}'::name[] AS rolparents,
postgres(#          NULL::oid AS parent_roloid,
postgres(#          NULL::name AS parent_rolname
postgres(#   FROM pg_catalog.pg_authid u
postgres(#   LEFT JOIN pg_catalog.pg_auth_members m on u.oid = m.member
postgres(#   LEFT JOIN pg_catalog.pg_authid g on m.roleid = g.oid
postgres(#   WHERE g.oid IS NULL
postgres(#   UNION ALL
postgres(#   SELECT u.rolname AS rolname,
postgres(#          u.oid AS roloid,
postgres(#          u.rolcanlogin,
postgres(#          u.rolsuper,
postgres(#          t.rolparents || g.rolname AS rolparents,
postgres(#          g.oid AS parent_roloid,
postgres(#          g.rolname AS parent_rolname
postgres(#   FROM pg_catalog.pg_authid u
postgres(#   JOIN pg_catalog.pg_auth_members m on u.oid = m.member
postgres(#   JOIN pg_catalog.pg_authid g on m.roleid = g.oid
postgres(#   JOIN roltree t on t.roloid = g.oid
postgres(# )
postgres-# SELECT
postgres-#   r.rolname,
postgres-#   r.roloid,
postgres-#   r.rolcanlogin,
postgres-#   r.rolsuper,
postgres-#   r.rolparents
postgres-# FROM roltree r
postgres-# ORDER BY 1;
CREATE VIEW
postgres=# SELECT
postgres-#   ro.rolname,
postgres-#   ro.roloid,
postgres-#   ro.rolcanlogin,
postgres-#   ro.rolsuper,
postgres-#   ro.rolparents
postgres-# FROM roletree ro
postgres-# WHERE (ro.rolcanlogin AND ro.rolsuper)
postgres-# OR
postgres-# (
postgres(#     ro.rolcanlogin AND EXISTS
postgres(#     (
postgres(#       SELECT TRUE FROM roletree ri
postgres(#       WHERE ri.rolname = ANY (ro.rolparents)
postgres(#       AND ri.rolsuper
postgres(#     )
postgres(# );
 rolname | roloid | rolcanlogin | rolsuper | rolparents
---------+--------+-------------+----------+------------
(0 rows)
```

If any roles are identified by this query, use `REVOKE` to correct.

Impact:

Much like the venerable `sudo` does for the OS, `set_user` manages superuser access for PostgreSQL. Complete configuration of `set_user` is documented at the extension's website and should be reviewed to ensure the logging entries that your organization cares about are properly configured.

Note that some external tools assume they can connect as the `postgres` user by default and this is no longer true. You may find some tools need different options, reconfigured, or even abandoned to compensate for this.

References:

1. https://github.com/pgaudit/set_user

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges
Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

5.8 Administrators Should Not Directly Log In To A System (i.e. use RunAs/sudo)
Administrators should be required to access a system using a fully logged and non-administrative account. Then, once logged on to the machine without administrative privileges, the administrator should transition to administrative privileges using tools such as Sudo on Linux/UNIX, RunAs on Windows, and other similar facilities for other types of systems.

Version 7

4.3 Ensure the Use of Dedicated Administrative Accounts
Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.

4.8 Make use of default roles (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

PostgreSQL provides a set of default roles which provide access to certain, commonly needed, privileged capabilities and information. Administrators can GRANT these roles to users and/or other roles in their environment, providing those users with access to the specified capabilities and information.

Rationale:

In keeping with the principle of least privilege, judicious use of the PostgreSQL default roles can greatly limit the access to privileged, or superuser, access.

Audit:

Review the list of all database roles that have `superuser` access and determine if one or more the default roles would suffice for the needs of that role:

```
$ whoami
postgres
$ psql
postgres=# select rolname from pg_roles where rolsuper is true;
 rolname
----------
 postgres
 doug
(2 rows)
```

Remediation:

If you've determined that one or more of the default roles can be used, simply `GRANT` it:

```
postgres=# GRANT pg_monitor TO doug;
GRANT ROLE
```

And then remove `superuser` from the account:

```
postgres=# ALTER ROLE doug NOSUPERUSER;
ALTER ROLE
postgres=# select rolname from pg_roles where rolsuper is true;
 rolname
----------
```

```
 postgres
(1 row)
```

Default Value:

The following default roles exist in PostgreSQL 12.x:

- `pg_read_all_settings` Read all configuration variables, even those normally visible only to superusers.
- `pg_read_all_stats` Read all `pg_stat_*` views and use various statistics related extensions, even those normally visible only to superusers.
- `pg_stat_scan_tables` Execute monitoring functions that may take `ACCESS SHARE` locks on tables, potentially for a long time.
- `pg_signal_backend` Send signals to other backends (eg: cancel query, terminate).
- `pg_read_server_files` Allow reading files from any location the database can access on the server with COPY and other file-access functions.
- `pg_write_server_files` Allow writing to files in any location the database can access on the server with COPY and other file-access functions.
- `pg_execute_server_program` Allow executing programs on the database server as the user the database runs as with COPY and other functions which allow executing a server-side program.
- `pg_monitor` Read/execute various monitoring views and functions. This role is a member of `pg_read_all_settings`, `pg_read_all_stats` and `pg_stat_scan_tables`.

Administrators can grant access to these roles to users using the `GRANT` command.

References:

1. https://www.postgresql.org/docs/12/default-roles.html

CIS Controls:

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

5 Connection and Login

The restrictions on client/user connections to the PostgreSQL database blocks unauthorized access to data and services by setting access rules. These security measures help to ensure that successful logins cannot be easily made through brute-force password attacks, pass the hash, or intuited by clever social engineering exploits.

Settings are generally recommended to be applied to all defined profiles. The following presents standalone examples of logins for particular use cases. The authentication rules are read from the PostgreSQL host-based authentication file, `pg_hba.conf`, from top to bottom. The first rule conforming to the condition of the request executes the METHOD *and stops further processing of the file*. Incorrectly applied rules, as defined by a single line instruction, can substantially alter the intended behavior resulting in either allowing or denying login attempts.

It is strongly recommended that authentication configurations be constructed incrementally with rigid testing for each newly applied rule. Because of the large number of different variations, this benchmark limits itself to a small number of authentication methods that can be successfully applied under most circumstances. Further analysis, using the other authentication methods available in PostgreSQL, is encouraged.

5.1 Ensure login via "local" UNIX Domain Socket is configured correctly (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

A remote host login, via ssh, is arguably the most secure means of remotely accessing and administering the PostgreSQL server. Connecting with the `psql` client, via UNIX DOMAIN SOCKETS, using the `peer` authentication method is the most secure mechanism available for local connections. Provided a database user account of the same name of the UNIX account has already been defined in the database, even ordinary user accounts can access the cluster in a similarly highly secure manner.

Rationale:

Audit:

Newly created data clusters are empty of data and have only one user account, the superuser (`postgres`). By default, the data cluster superuser is named after the UNIX account. Login authentication is tested via UNIX DOMAIN SOCKETS by the UNIX user account `postgres`, the default account, and `set_user` has not yet been configured:

```
$ whoami
postgres
$ psql postgres
postgres=#
```

Login attempts by another UNIX user account as the superuser should be denied:

```
$ su - user1
$ whoami
user1
$ psql -U postgres -d postgres
psql: FATAL:  Peer authentication failed for user "postgres"
$ exit
```

This test demonstrates that not only is logging in as the superuser blocked, but so is logging in as another user:

```
$ su - user2
$ whoami
user2
$ psql -U postgres -d postgres
```

```
psql: FATAL: Peer authentication failed for user "postgres"
$ psql -U user1 -d postgres
psql: FATAL: Peer authentication failed for user "user1"
$ psql -U user2 -d postgres
postgres=>
```

Remediation:

Creation of a database account that matches the local account allows PEER authentication:

```
$ psql -c "CREATE ROLE user1 WITH LOGIN;"
CREATE ROLE
```

Execute the following as the UNIX user account, the default authentication rules should now permit the login:

```
$ su - user1
$ whoami
user1
$ psql -d postgres
postgres=>
```

As per the host-based authentication rules in `$PGDATA/pg_hba.conf`, all login attempts via UNIX DOMAIN SOCKETS are processed on the line beginning with `local`.

This is the minimal rule that must be in place allowing PEER connections:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             postgres                                peer
```

More traditionally, a rule like the following would be used to allow any local PEER connection:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     peer
```

Once edited, the server process must reload the authentication file before it can take effect. Improperly configured rules cannot update i.e. the old rules remain in place. The PostgreSQL logs will report the outcome of the SIGHUP:

```
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

The following examples illustrate other possible configurations. The resultant "rule" of success/failure depends upon the first matching line:

```
# allow postgres user logins locally via UNIX socket
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             postgres                                peer

# allow all local users via UNIX socket
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     peer

# allow all local users, via UNIX socket, only if they are connecting to a db
named the same as their username
# e.g. if user 'bob' is connecting to a db named 'bob'
# TYPE  DATABASE        USER                                    METHOD
local   samerole        all                                     peer

# allow only local users, via UNIX socket, who are members of the 'rw' role
in the db
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             +rw                                     peer
```

References:

1. https://www.postgresql.org/docs/12/static/client-authentication.html
2. https://www.postgresql.org/docs/12/static/auth-pg-hba-conf.html

CIS Controls:

Version 6

3.4 Use Only Secure Channels For Remote System Administration
Perform all remote administration of servers, workstation, network devices, and similar equipment over secure channels. Protocols such as telnet, VNC, RDP, or others that do not actively support strong encryption should only be used if they are performed over a secondary encryption channel, such as SSL, TLS or IPSEC.

Version 7

4.5 Use Multifactor Authentication For All Administrative Access
Use multi-factor authentication and encrypted channels for all administrative account access.

5.2 Ensure login via "host" TCP/IP Socket is configured correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

A large number of authentication METHODs are available for hosts connecting using TCP/IP sockets, including:

- `trust`
- `reject`
- `md5`
- `scram-sha-256`
- `password`
- `gss`
- `sspi`
- `ident`
- `pam`
- `ldap`
- `radius`
- `cert`

METHODs `trust`, `password`, and `ident` are **not** to be used for remote logins. METHOD `md5` is the most popular and can be used in both encrypted and unencrypted sessions,however, *it is vulnerable to packet replay attacks*. **It is recommended that `scram-sha-256` be used instead of `md5`.**

Use of the `gss`, `sspi`, `pam`, `ldap`, `radius`, and `cert` METHODs, while more secure than `md5`, are dependent upon the availability of external authenticating processes/services and thus are not covered in this benchmark.

Rationale:

Audit:

Newly created data clusters are empty of data and have one only one user account, the superuser. By default, the data cluster superuser is named after the UNIX account `postgres`. Login authentication can be tested via TCP/IP SOCKETS by any UNIX user account from the localhost. A password must be assigned to each login ROLE:

```
postgres=# ALTER ROLE postgres WITH PASSWORD 'secret_password';
ALTER ROLE
```

Test an unencrypted session:

```
$ psql 'host=localhost user=postgres sslmode=disable'
Password:
```

Test an encrypted session:

```
$ psql 'host=localhost user=postgres sslmode=require'
Password:
```

Remote logins repeat the previous invocations but, of course, from the remote host:
Test unencrypted session:

```
$ psql 'host=server-name-or-IP user=postgres sslmode=disable'
Password:
```

Test encrypted sessions:

```
$ psql 'host=server-name-or-IP user=postgres sslmode=require'
Password:
```

Remediation:

Confirm a login attempt has been made by looking for a logged error message detailing the nature of the authenticating failure. In the case of failed login attempts, whether encrypted or unencrypted, check the following:

- The server should be sitting on a port exposed to the remote connecting host i.e. NOT ip address 127.0.0.1

  ```
  listen_addresses = '*'
  ```

- An authenticating rule must exist in the file `pg_hba.conf`

This example permits only encrypted sessions for the `postgres` role and denies all unencrypted session for the `postgres` role:

```
# TYPE    DATABASE           USER            ADDRESS           METHOD
hostssl    all             postgres         0.0.0.0/0          scram-sha-256
hostnossl  all             postgres         0.0.0.0/0          reject
```

The following examples illustrate other possible configurations. The resultant "rule" of success/failure depends upon the **first matching line**.

```
# allow 'postgres' user only from 'localhost/loopback' connections
# and only if you know the password
# TYPE    DATABASE        USER            ADDRESS                 METHOD
```

```
host      all             postgres        127.0.0.1/32            scram-sha-
256

# allow users to connect remotely only to the database named after them,
# with the correct user password:
# (accepts both SSL and non-SSL connections)
# TYPE    DATABASE        USER            ADDRESS                 METHOD
host      samerole        all             0.0.0.0/0               scram-sha-
256

# allow only those users who are a member of the 'rw' role to connect
# only to the database named after them, with the correct user password:
# (accepts both SSL and non-SSL connections)
# TYPE    DATABASE        USER            ADDRESS                 METHOD
host      samerole        +rw             0.0.0.0/0               scram-sha-
256
```

Default Value:

The availability of the different password-based authentication methods depends on how a user's password on the server is encrypted (or hashed, more accurately). This is controlled by the configuration parameter `password_encryption` at the time the password is set.

If a password was encrypted using the `scram-sha-256` setting, then it can be used for the authentication methods `scram-sha-256` and `password` (but password transmission will be in plain text in the latter case). The authentication method specification `md5` will automatically switch to using the `scram-sha-256` method in this case, as explained above, so it will also work.

If a password was encrypted using the `md5` setting, then it can be used only for the `md5` and `password` authentication method specifications (again, with the password transmitted in plain text in the latter case).

Previous PostgreSQL releases supported storing the password on the server in plain text. This is no longer possible.

To check the currently stored password hashes, see the system catalog `pg_authid`. To upgrade an existing installation from `md5` to `scram-sha-256`, after having ensured that all client libraries in use are new enough to support SCRAM, set `password_encryption = 'scram-sha-256'` in `postgresql.conf`, reload the `postmaster`, make all users set new passwords, and change the authentication method specifications in `pg_hba.conf` to `scram-sha-256`.

References:

1. https://www.postgresql.org/docs/12/static/client-authentication.html
2. https://www.postgresql.org/docs/12/static/auth-pg-hba-conf.html

3. https://tools.ietf.org/html/rfc7677

Notes:

1. Use TYPE `hostssl` when administrating the database cluster as a superuser.
2. Use TYPE `hostnossl` for performance purposes and when DML operations are deemed safe without SSL connections.
3. No examples have been given for ADDRESS, i.e., CIDR, hostname, domain names, etc.
4. Only three (3) types of METHOD have been documented; there are many more.

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

6 PostgreSQL Settings

As PostgreSQL evolves with each new iteration, configuration parameters are constantly being added, deprecated, or removed. These configuration parameters define not only server function but how well it performs.

Many routine activities, combined with a specific set of configuration parameter values, can sometimes result in degraded performance and, under a specific set of conditions, even comprise the security of the RDBMS. The fact of the matter is that any parameter has the potential to affect the accessibility and performance of a running server.

Rather than describing all the possible combination of events, this benchmark describes how a parameter can be compromised. Examples reflect the most common, and easiest to understand, exploits. Although by no means exhaustive, it is hoped that you will be able to understand the attack vectors in the context of your environment.

6.1 Ensure 'Attack Vectors' Runtime Parameters are Configured (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

Understanding the vulnerability of PostgreSQL runtime parameters by the particular delivery method, or attack vector.

Rationale:

There are as many ways of compromising a server as there are runtime parameters. A combination of any one or more of them executed at the right time under the right conditions has the potential to compromise the RDBMS. Mitigating risk is dependent upon one's understanding of the attack vectors and includes:

1. Via user session: includes those runtime parameters that can be set by a ROLE that persists for the life of a server-client session.
2. Via attribute: includes those runtime parameters that can be set by a ROLE during a server-client session that can be assigned as an attribute for an entity such as a table, index, database, or role.
3. Via server reload: includes those runtime parameters that can be set by the superuser using a SIGHUP or configuration file reload command and affects the entire cluster.
4. Via server restart: includes those runtime parameters that can be set and effected by restarting the server process and affects the entire cluster.

Audit:

Review all configuration settings. Configure PostgreSQL logging to record all modifications and changes to the RDBMS.

Remediation:

In the case of a changed parameter, the value is returned back to its default value. In the case of a successful exploit of an already set runtime parameter then an analysis must be carried out determining the best approach mitigating the risk.

Impact:

It can be difficult to totally eliminate risk. Once changed, detecting a miscreant parameter can become problematic.

References:

1. https://www.postgresql.org/docs/12/static/runtime-config.html

CIS Controls:

Version 6

18.7 Use Standard Database Hardening Templates
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

Version 7

18.11 Use Standard Hardening Configuration Templates for Databases
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

6.2 Ensure 'backend' runtime parameters are configured correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

In order to serve multiple clients efficiently, the PostgreSQL server launches a new "backend" process for each client. The runtime parameters in this benchmark section are controlled by the backend process. The server's performance, in the form of slow queries causing a denial of service, and the RDBM's auditing abilities for determining root cause analysis can be compromised via these parameters.

Rationale:

A denial of service is possible by denying the use of indexes and by slowing down client access to an unreasonable level. Unsanctioned behavior can be introduced by introducing rogue libraries which can then be called in a database session. Logging can be altered and obfuscated inhibiting root cause analysis.

Audit:

Issue the following command to verify the backend runtime parameters are configured correctly:

```
postgres=# SELECT name, setting FROM pg_settings WHERE context IN
('backend','superuser-backend') ORDER BY 1;
         name          | setting
-----------------------+---------
 ignore_system_indexes | off
 jit_debugging_support | off
 jit_profiling_support | off
 log_connections       | on
 log_disconnections    | on
 post_auth_delay       | 0
(6 rows)
```

Note: Effecting changes to these parameters can only be made at server start. Therefore, a successful exploit *may not be detected until after* a server restart, e.g., during a maintenance window.

Remediation:

Once detected, the unauthorized/undesired change can be corrected by altering the configuration file and executing a server restart. In the case where the parameter has been on the command line invocation of `pg_ctl` the `restart` invocation is insufficient and an explicit `stop` and `start` must instead be made.

1. Query the view `pg_settings` and compare with previous query outputs for any changes.
2. Review configuration files `postgresql.conf` and `postgresql.auto.conf` and compare them with previously archived file copies for any changes.
3. Examine the process output and look for parameters that were used at server startup:

```
ps aux | grep -E '[p]ost' | grep -- '-[D]'
```

Impact:

All changes made on this level will affect the overall behavior of the server. These changes can only be affected by a server restart after the parameters have been altered in the configuration files.

References:

1. https://www.postgresql.org/docs/12/static/view-pg-settings.html
2. https://www.postgresql.org/docs/12/static/runtime-config.html

CIS Controls:

Version 6

18.7 Use Standard Database Hardening Templates
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

Version 7

18.11 Use Standard Hardening Configuration Templates for Databases
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

6.3 Ensure 'Postmaster' Runtime Parameters are Configured (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

PostgreSQL runtime parameters that are executed by the postmaster process.

Rationale:

The `postmaster` process is the supervisory process that assigns a backend process to an incoming client connection. The `postmaster` manages key runtime parameters that are either shared by all backend connections or needed by the `postmaster` process itself to run.

Audit:

The following parameters can only be set at server start by the owner of the PostgreSQL server process and cluster, typically the UNIX user account `postgres`. Therefore, all exploits require the successful compromise of either that UNIX account or the `postgres` superuser account itself.

```
postgres=# SELECT name, setting FROM pg_settings WHERE context = 'postmaster'
ORDER BY 1;
                name                 |                setting
-------------------------------------+--------------------------------------
-
 allow_system_table_mods             | off
 archive_mode                        | off
 autovacuum_freeze_max_age           | 200000000
 autovacuum_max_workers              | 3
 autovacuum_multixact_freeze_max_age | 400000000
 bonjour                             | off
 bonjour_name                        |
 cluster_name                        |
 config_file                         | /var/lib/pgsql/12/data/postgresql.conf
 data_directory                      | /var/lib/pgsql/12/data
 data_sync_retry                     | off
 dynamic_shared_memory_type          | posix
 event_source                        | PostgreSQL
 external_pid_file                   |
 hba_file                            | /var/lib/pgsql/12/data/pg_hba.conf
 hot_standby                         | on
 huge_pages                          | try
 ident_file                          | /var/lib/pgsql/12/data/pg_ident.conf
 jit_provider                        | llvmjit
 listen_addresses                    | localhost
 logging_collector                   | on
 max_connections                     | 100
 max_files_per_process               | 1000
 max_locks_per_transaction           | 64
 max_logical_replication_workers     | 4
 max_pred_locks_per_transaction      | 64
 max_prepared_transactions           | 0
 max_replication_slots               | 10
 max_wal_senders                     | 10
 max_worker_processes                | 8
 old_snapshot_threshold              | -1
 port                                | 5432
 primary_conninfo                    |
 primary_slot_name                   |
 recovery_target                     |
 recovery_target_action              | pause
 recovery_target_inclusive           | on
 recovery_target_lsn                 |
 recovery_target_name                |
 recovery_target_time                |
 recovery_target_timeline            | latest
 recovery_target_xid                 |
 restore_command                     |
 shared_buffers                      | 16384
 shared_memory_type                  | mmap
 shared_preload_libraries            | pgaudit,set_user
 superuser_reserved_connections      | 3
 track_activity_query_size           | 1024
 track_commit_timestamp              | off
 unix_socket_directories             | /var/run/postgresql, /tmp
 unix_socket_group                   |
 unix_socket_permissions             | 0777
 wal_buffers                         | 512
 wal_level                           | replica
 wal_log_hints                       | off
(55 rows)
```

Remediation:

Once detected, the unauthorized/undesired change can be corrected by editing the altered configuration file and executing a server restart. In the case where the parameter has been on the command line invocation of `pg_ctl` the `restart` invocation is insufficient and an explicit `stop` and `start` must instead be made.

Detecting a change is possible by one of the following methods:

1. Query the view `pg_settings` and compare with previous query outputs for any changes
2. Review the configuration files `postgresql.conf` and `postgresql.auto.conf` and compare with previously archived file copies for any changes
3. Examine the process output and look for parameters that were used at server startup:

```
ps aux | grep -E 'postgres' | grep -- '-[D]'
```

Impact:

All changes made on this level will affect the overall behavior of the server. These changes can be effected by editing the PostgreSQL configuration files and by either executing a server SIGHUP from the command line or, as superuser `postgres`, executing the SQL command `select pg_reload_conf()`. A denial of service is possible by the over-allocating of limited resources, such as RAM. Data can be corrupted by allowing damaged pages to load or by changing parameters to reinterpret values in an unexpected fashion, e.g. changing the time zone. Client messages can be altered in such a way as to interfere with the application logic. Logging can be altered and obfuscated inhibiting root cause analysis.

References:

1. https://www.postgresql.org/docs/12/static/view-pg-settings.html
2. https://www.postgresql.org/docs/12/static/runtime-config.html

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

Version 7

18.11 Use Standard Hardening Configuration Templates for Databases
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

6.4 Ensure 'SIGHUP' Runtime Parameters are Configured (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

PostgreSQL runtime parameters that are executed by the SIGHUP signal.

Rationale:

In order to define server behavior and optimize server performance, the server's superuser has the privilege of setting these parameters which are found in the configuration files `postgresql.conf` and `pg_hba.conf`. Alternatively, those parameters found in `postgresql.conf` can also be changed using a server login session and executing the SQL command `ALTER SYSTEM` which writes its changes in the configuration file `postgresql.auto.conf`.

Audit:

The following parameters can be set at any time, without interrupting the server, by the owner of the `postmaster` server process and cluster (typically UNIX user account `postgres`).

```
postgres=# SELECT name, setting FROM pg_settings WHERE context = 'sighup'
ORDER BY 1;
                  name                  |                setting
----------------------------------------+------------------------------------
---
 archive_cleanup_command                |
 archive_command                        | (disabled)
 archive_timeout                        | 0
 authentication_timeout                 | 60
 autovacuum                             | on
 autovacuum_analyze_scale_factor        | 0.1
 autovacuum_analyze_threshold           | 50
 autovacuum_naptime                     | 60
 autovacuum_vacuum_cost_delay           | 2
 autovacuum_vacuum_cost_limit           | -1
 autovacuum_vacuum_scale_factor         | 0.2
 autovacuum_vacuum_threshold            | 50
 autovacuum_work_mem                    | -1
 bgwriter_delay                         | 200
 bgwriter_flush_after                   | 64
 bgwriter_lru_maxpages                  | 100
 bgwriter_lru_multiplier                | 2
 checkpoint_completion_target           | 0.5
 checkpoint_flush_after                 | 32
 checkpoint_timeout                     | 300
 checkpoint_warning                     | 30
 db_user_namespace                      | off
 fsync                                  | on
 full_page_writes                       | on
 hot_standby_feedback                   | off
 krb_caseins_users                      | off
 krb_server_keyfile                     |
FILE:/etc/sysconfig/pgsql/krb5.keytab
 log_autovacuum_min_duration            | -1
 log_checkpoints                        | off
 log_destination                        | stderr
 log_directory                          | log
 log_file_mode                          | 0600
 log_filename                           | postgresql-%a.log
 log_hostname                           | off
 log_line_prefix                        | %m [%p]
 log_rotation_age                       | 1440
 log_rotation_size                      | 0
 log_timezone                           | GMT
 log_truncate_on_rotation               | on
 max_pred_locks_per_page                | 2
 max_pred_locks_per_relation            | -2
 max_standby_archive_delay              | 30000
 max_standby_streaming_delay            | 30000
 max_sync_workers_per_subscription      | 2
 max_wal_size                           | 1024
 min_wal_size                           | 80
 pre_auth_delay                         | 0
 promote_trigger_file                   |
 recovery_end_command                   |
 recovery_min_apply_delay               | 0
 restart_after_crash                    | on
 set_user.block_alter_system            | on
 set_user.block_copy_program            | on
 set_user.block_log_statement           | on
 set_user.nosuperuser_target_whitelist  | *
 set_user.superuser_audit_tag           | AUDIT
 set_user.superuser_whitelist           | *
 ssl                                    | off
 ssl_ca_file                            |
 ssl_cert_file                          | server.crt
 ssl_ciphers                            | HIGH:MEDIUM:+3DES:!aNULL
 ssl_crl_file                           |
 ssl_dh_params_file                     |
 ssl_ecdh_curve                         | prime256v1
 ssl_key_file                           | server.key
 ssl_max_protocol_version               |
 ssl_min_protocol_version               | TLSv1
 ssl_passphrase_command                 |
 ssl_passphrase_command_supports_reload | off
 ssl_prefer_server_ciphers              | on
 stats_temp_directory                   | pg_stat_tmp
 synchronous_standby_names              |
 syslog_facility                        | local0
 syslog_ident                               | postgres
 syslog_sequence_numbers                    | on
 syslog_split_messages                      | on
 trace_recovery_messages                    | log
 vacuum_defer_cleanup_age                   | 0
 wal_keep_segments                          | 0
 wal_receiver_status_interval               | 10
 wal_receiver_timeout                       | 60000
 wal_retrieve_retry_interval                | 5000
 wal_sync_method                            | fdatasync
 wal_writer_delay                           | 200
 wal_writer_flush_after                     | 128
(85 rows)
```

Remediation:

Restore all values in the PostgreSQL configuration files and invoke the server to reload the configuration files.

Impact:

All changes made on this level will affect the overall behavior of the server. These changes can be effected by editing the PostgreSQL configuration files and by either executing a server SIGHUP from the command line or, as superuser `postgres`, executing the SQL command `select pg_reload_conf()`. A denial of service is possible by the over-allocating of limited resources, such as RAM. Data can be corrupted by allowing damaged pages to load or by changing parameters to reinterpret values in an unexpected fashion, e.g. changing the time zone. Client messages can be altered in such a way as to interfere with the application logic. Logging can be altered and obfuscated inhibiting root cause analysis.

References:

1. https://www.postgresql.org/docs/12/static/view-pg-settings.html
2. https://www.postgresql.org/docs/12/static/runtime-config.html

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

Version 7

18.11 Use Standard Hardening Configuration Templates for Databases
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

6.5 Ensure 'Superuser' Runtime Parameters are Configured (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

PostgreSQL runtime parameters that can only be executed by the server's superuser, which is traditionally `postgres`.

Rationale:

In order to improve and optimize server performance, the server's superuser has the privilege of setting these parameters which are found in the configuration file `postgresql.conf`. Alternatively, they can be changed in a PostgreSQL login session via the SQL command `ALTER SYSTEM` which writes its changes in the configuration file `postgresql.auto.conf`.

Audit:

The following parameters can only be set at server start by the owner of the PostgreSQL server process and cluster i.e. typically UNIX user account `postgres`. Therefore, all exploits require the successful compromise of either that UNIX account or the `postgres` superuser account itself.

```
postgres=# SELECT name, setting FROM pg_settings WHERE context = 'superuser'
ORDER BY 1;
            name             |   setting
-----------------------------+-------------
 commit_delay                | 0
 deadlock_timeout            | 1000
 dynamic_library_path        | $libdir
 ignore_checksum_failure     | off
 jit_dump_bitcode            | off
 lc_messages                 | en_US.UTF-8
 lo_compat_privileges        | off
 log_duration                | off
 log_error_verbosity         | default
 log_executor_stats          | off
 log_lock_waits              | off
 log_min_duration_statement  | -1
 log_min_error_statement     | error
 log_min_messages            | warning
 log_parser_stats            | off
 log_planner_stats           | off
 log_replication_commands    | off
 log_statement               | none
 log_statement_stats         | off
 log_temp_files              | -1
 log_transaction_sample_rate | 0
 max_stack_depth             | 2048
 pgaudit.log                 | ddl,write
 pgaudit.log_catalog         | on
 pgaudit.log_client          | off
 pgaudit.log_level           | log
 pgaudit.log_parameter       | off
 pgaudit.log_relation        | off
 pgaudit.log_statement_once  | off
 pgaudit.role                |
 session_preload_libraries   |
 session_replication_role    | origin
 temp_file_limit             | -1
 track_activities            | on
 track_counts                | on
 track_functions             | none
 track_io_timing             | off
 update_process_title        | on
 wal_compression             | off
 wal_consistency_checking    |
 wal_init_zero               | on
 wal_recycle                 | on
 zero_damaged_pages          | off
(43 rows)
```

Remediation:

The exploit is made in the configuration files. These changes are effected upon server restart. Once detected, the unauthorized/undesired change can be made by editing the altered configuration file and executing a server restart. In the case where the parameter has been set on the command line invocation of `pg_ctl` the `restart` invocation is insufficient and an explicit `stop` and `start` must instead be made.

Detecting a change is possible by one of the following methods:

1. Query the view `pg_settings` and compare with previous query outputs for any changes.
2. Review the configuration files `postgreql.conf` and `postgreql.auto.conf` and compare with previously archived file copies for any changes
3. Examine the process output and look for parameters that were used at server startup:

```
ps aux | grep -E 'post' | grep -- '-[D]'
```

Impact:

All changes made on this level will affect the overall behavior of the server. These changes can only be affected by a server restart after the parameters have been altered in the configuration files. A denial of service is possible by the over allocating of limited resources, such as RAM. Data can be corrupted by allowing damaged pages to load or by changing parameters to reinterpret values in an unexpected fashion, e.g. changing the time zone. Client messages can be altered in such a way as to interfere with the application logic. Logging can be altered and obfuscated inhibiting root cause analysis.

References:

1. https://www.postgresql.org/docs/12/static/view-pg-settings.html
2. https://www.postgresql.org/docs/12/static/runtime-config.html

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges
Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

18.11 Use Standard Hardening Configuration Templates for Databases
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

6.6 Ensure 'User' Runtime Parameters are Configured (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

These PostgreSQL runtime parameters are managed at the user account (ROLE) level.

Rationale:

In order to improve performance and optimize features, a `ROLE` has the privilege of setting numerous parameters in a transaction, session, or as an entity attribute. Any `ROLE` can alter any of these parameters.

Audit:

The method used to analyze the state of ROLE runtime parameters and to determine if they have been compromised is to inspect all catalogs and list attributes for database entities such as `ROLE`s and databases:

```
postgres=# SELECT name, setting FROM pg_settings WHERE context = 'user' ORDER
BY 1;
                name                 |      setting
-------------------------------------+--------------------
 application_name                    | psql
 array_nulls                         | on
 backend_flush_after                 | 0
 backslash_quote                     | safe_encoding
 bytea_output                        | hex
 check_function_bodies               | on
 client_encoding                     | UTF8
 client_min_messages                 | notice
 commit_siblings                     | 5
 constraint_exclusion                | partition
 cpu_index_tuple_cost                | 0.005
 cpu_operator_cost                   | 0.0025
 cpu_tuple_cost                      | 0.01
 cursor_tuple_fraction               | 0.1
 DateStyle                           | ISO, MDY
 debug_pretty_print                  | on
 debug_print_parse                   | off
 debug_print_plan                    | off
 debug_print_rewritten               | off
 default_statistics_target           | 100
 default_table_access_method         | heap
 default_tablespace                  |
 default_text_search_config          | pg_catalog.english
 default_transaction_deferrable      | off
 default_transaction_isolation       | read committed
 default_transaction_read_only       | off
 effective_cache_size                | 524288
 effective_io_concurrency            | 1
 enable_bitmapscan                   | on
 enable_gathermerge                  | on
 enable_hashagg                      | on
 enable_hashjoin                     | on
 enable_indexonlyscan                | on
 enable_indexscan                    | on
 enable_material                     | on
 enable_mergejoin                    | on
 enable_nestloop                     | on
 enable_parallel_append              | on
 enable_parallel_hash                | on
 enable_partition_pruning            | on
 enable_partitionwise_aggregate      | off
 enable_partitionwise_join           | off
 enable_seqscan                      | on
 enable_sort                         | on
 enable_tidscan                      | on
 escape_string_warning               | on
 exit_on_error                       | off
 extra_float_digits                  | 1
 force_parallel_mode                 | off
 from_collapse_limit                 | 8
 geqo                                | on
 geqo_effort                         | 5
 geqo_generations                    | 0
 geqo_pool_size                      | 0
 geqo_seed                           | 0
 geqo_selection_bias                 | 2
 geqo_threshold                      | 12
 gin_fuzzy_search_limit              | 0
 gin_pending_list_limit              | 4096
 idle_in_transaction_session_timeout | 0
 IntervalStyle                       | postgres
 jit                                 | on
 jit_above_cost                      | 100000
 jit_expressions                     | on
 jit_inline_above_cost               | 500000
 jit_optimize_above_cost             | 500000
 jit_tuple_deforming                 | on
 join_collapse_limit                 | 8
 lc_monetary                         | en_US.UTF-8
 lc_numeric                          | en_US.UTF-8
 lc_time                             | en_US.UTF-8
 local_preload_libraries             |
 lock_timeout                        | 0
 maintenance_work_mem                | 65536
 max_parallel_maintenance_workers    | 2
 max_parallel_workers                | 8
 max_parallel_workers_per_gather     | 2
 min_parallel_index_scan_size        | 64
 min_parallel_table_scan_size        | 1024
 operator_precedence_warning          | off
 parallel_leader_participation        | on
 parallel_setup_cost                  | 1000
 parallel_tuple_cost                  | 0.1
 password_encryption                  | md5
 plan_cache_mode                      | auto
 quote_all_identifiers                | off
 random_page_cost                     | 4
 row_security                         | on
 search_path                          | "$user", public
 seq_page_cost                        | 1
 standard_conforming_strings          | on
 statement_timeout                    | 0
 synchronize_seqscans                 | on
 synchronous_commit                   | on
 tcp_keepalives_count                 | 0
 tcp_keepalives_idle                  | 0
 tcp_keepalives_interval              | 0
 tcp_user_timeout                     | 0
 temp_buffers                         | 1024
 temp_tablespaces                     |
 TimeZone                             | America/New_York
 timezone_abbreviations               | Default
 trace_notify                         | off
 trace_sort                           | off
 transaction_deferrable               | off
 transaction_isolation                | read committed
 transaction_read_only                | off
 transform_null_equals                | off
 vacuum_cleanup_index_scale_factor    | 0.1
 vacuum_cost_delay                    | 0
 vacuum_cost_limit                    | 200
 vacuum_cost_page_dirty               | 20
 vacuum_cost_page_hit                 | 1
 vacuum_cost_page_miss                | 10
 vacuum_freeze_min_age                | 50000000
 vacuum_freeze_table_age              | 150000000
 vacuum_multixact_freeze_min_age      | 5000000
 vacuum_multixact_freeze_table_age    | 150000000
 wal_sender_timeout                   | 60000
 work_mem                             | 4096
 xmlbinary                            | base64
 xmloption                            | content
(122 rows)
```

Remediation:

In the matter of a user session, the login sessions must be validated that it is not executing undesired parameter changes. In the matter of attributes that have been changed in entities, they must be manually reverted to its default value(s).

Impact:

A denial of service is possible by the over-allocating of limited resources, such as RAM. Changing `VACUUM` parameters can force a server shutdown which is standard procedure preventing data corruption from transaction ID wraparound. Data can be corrupted by changing parameters to reinterpret values in an unexpected fashion, e.g. changing the time zone. Logging can be altered and obfuscated to inhibit root cause analysis.

References:

1. https://www.postgresql.org/docs/12/static/view-pg-settings.html
2. https://www.postgresql.org/docs/12/static/runtime-config.html

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges
Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

18.11 Use Standard Hardening Configuration Templates for Databases
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

6.7 Ensure FIPS 140-2 OpenSSL Cryptography Is Used (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Install, configure, and use OpenSSL on a platform that has a NIST certified FIPS 140-2 installation of OpenSSL. This provides PostgreSQL instances the ability to generate and validate cryptographic hashes to protect unclassified information requiring confidentiality and cryptographic protection, in accordance with the data owner's requirements.

Rationale:

Federal Information Processing Standard (FIPS) Publication 140-2 is a computer security standard developed by a U.S. Government and industry working group for validating the quality of cryptographic modules. Use of weak, or untested, encryption algorithms undermine the purposes of utilizing encryption to protect data. PostgreSQL uses OpenSSL for the underlying encryption layer.

The database and application must implement cryptographic modules adhering to the higher standards approved by the federal government since this provides assurance they have been tested and validated. It is the responsibility of the data owner to assess the cryptography requirements in light of applicable federal laws, Executive Orders, directives, policies, regulations, and standards.

For detailed information, refer to NIST FIPS Publication 140-2, *Security Requirements for Cryptographic Modules*. Note that the product's cryptographic modules must be validated and certified by NIST as FIPS-compliant. The security functions validated as part of FIPS 140-2 for cryptographic modules are described in FIPS 140-2 Annex A. Currently only Red Hat Enterprise Linux is certified as a FIPS 140-2 distribution of OpenSSL. For other operating systems, users must obtain or build their own FIPS 140-2 OpenSSL libraries.

Audit:

If PostgreSQL is not installed on Red Hat Enterprise Linux (RHEL) or CentOS then FIPS cannot be enabled natively. Otherwise, the deployment must incorporate a custom build of the operating system.

As the system administrator:

1. Run the following to see if FIPS is enabled:

```
$ fips-mode-setup --check
FIPS mode is enabled
```

If `FIPS mode is anabled` is not displayed, then the system is not FIPS enabled.

2. Run the following (your results and version may vary):

```
$ openssl version
OpenSSL 1.1.1-fips  1 Sep 2019
```

If `fips` is not included in the OpenSSL version, then the system is **not** FIPS capable.

Remediation:

Configure OpenSSL to be FIPS compliant. PostgreSQL uses OpenSSL for cryptographic modules. To configure OpenSSL to be FIPS 140-2 compliant, see the official RHEL Documentation. Below is a general summary of the steps required:

To switch the system to FIPS mode in RHEL 8:

```
# fips-mode-setup --enable
Setting system policy to FIPS
FIPS mode will be enabled.
Please reboot the system for the setting to take effect.
```

Restart your system to allow the kernel to switch to FIPS mode:

```
# reboot
```

After the restart, you can check the current state of FIPS mode:

```
# fips-mode-setup --check
FIPS mode is enabled.
```

References:

1. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/security_hardening/using-the-system-wide-cryptographic-policies_security-hardening#switching-the-system-to-fips-mode_using-the-system-wide-cryptographic-policies
2. https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp1758.pdf
3. https://csrc.nist.gov/publications/fips

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

6.8 Ensure SSL is enabled and configured correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

SSL on a PostgreSQL server should be enabled (set to `on`) and configured to encrypt TCP traffic to and from the server.

Rationale:

If SSL is not enabled and configured correctly, this increases the risk of data being compromised in transit.

Audit:

To determine whether SSL is enabled (set to `on`), simply query the parameter value while logged into the database using either the `SHOW ssl` command or `SELECT` from system catalog view `pg_settings` as illustrated below. In both cases, `ssl` is `off`; this is a fail.

```
postgres=# SHOW ssl;
ssl
-----
off
(1 row)

postgres=# SELECT name, setting, source FROM pg_settings WHERE name = 'ssl';
name | setting |       source
-----+---------+--------------------
ssl  | off     | default
(1 row)
```

Remediation:

For this example, and ease of illustration, we will be using a self-signed certificate for the server generated via `openssl`, and the PostgreSQL defaults for file naming and location in the PostgreSQL `$PGDATA` directory.

```
$ whoami
postgres
$ # create new certificate and enter details at prompts
$ openssl req -new -text -out server.req
Generating a 2048 bit RSA private key
.....................+++
..................................................................+++
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:Ohio
Locality Name (eg, city) [Default City]:Columbus
Organization Name (eg, company) [Default Company Ltd]:Me Inc
Organizational Unit Name (eg, section) []:IT
Common Name (eg, your name or your server's hostname) []:my.me.inc
Email Address []:me@meinc.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

$ # remove passphrase (required for automatic server start up)
$ openssl rsa -in privkey.pem -out server.key && rm privkey.pem
Enter pass phrase for privkey.pem:
writing RSA key

$ # modify certificate to self signed, generate .key and .crt files
$ openssl req -x509 -in server.req -text -key server.key -out server.crt

$ # copy .key and .crt files to appropriate location, here default $PGDATA
$ cp server.key server.crt $PGDATA

$ # restrict file mode for server.key
$ chmod og-rwx server.key
```

Edit the PostgreSQL configuration file `postgresql.conf` to ensure the following items are set. Again, we are using defaults. Note that altering these parameters will require restarting the cluster.

```
# (change requires restart)
ssl = on

# allowed SSL ciphers
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'

# (change requires restart)
ssl_cert_file = 'server.crt'

# (change requires restart)
ssl_key_file = 'server.key'
```

```
password_encryption =  scram-sha-256
```

Finally, restart PostgreSQL and confirm `ssl` using commands outlined in Audit Procedures:

```
postgres=# show ssl;
 ssl
-----
 on
(1 row)
```

Impact:

A self-signed certificate can be used for testing, but a certificate signed by a certificate authority (CA) (either one of the global CAs or a local one) should be used in production so that clients can verify the server's identity. If all the database clients are local to the organization, using a local CA is recommended.

To ultimately enable and enforce `ssl` authentication for the server, appropriate `hostssl` records must be added to the `pg_hba.conf` file. Be sure to `reload` PostgreSQL after any changes (restart not required).

Note: The `hostssl` record matches connection attempts made using TCP/IP, but **only** when the connection is made with SSL encryption. The `host` record matches attempts made using TCP/IP, but allows both SSL and non-SSL connections. The `hostnossl` record matches attempts made using TCP/IP, but only those *without* SSL. *Care should be taken to enforce SSL as appropriate.*

References:

1. https://www.postgresql.org/docs/12/static/ssl-tcp.html
2. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf
3. https://www.postgresql.org/docs/12/static/libpq-ssl.html

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

6.9 Ensure the pgcrypto extension is installed and configured correctly (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

PostgreSQL must implement cryptographic mechanisms to prevent unauthorized disclosure or modification of organization-defined information at rest (to include, at a minimum, PII and classified information) on organization-defined information system components.

Rationale:

PostgreSQL handling data that requires "data at rest" protections must employ cryptographic mechanisms to prevent unauthorized disclosure and modification of the information at rest. These cryptographic mechanisms may be native to PostgreSQL or implemented via additional software or operating system/file system settings, as appropriate to the situation. Information at rest refers to the state of information when it is located on a secondary storage device (e.g. disk drive, tape drive) within an organizational information system.

Selection of a cryptographic mechanism is based on the need to protect the integrity of organizational information. The strength of the mechanism is commensurate with the security category and/or classification of the information. Organizations have the flexibility to either encrypt all information on storage devices (i.e. full disk encryption) or encrypt specific data structures (e.g. files, records, or fields). Organizations may also optionally choose to implement both to implement layered security.

The decision whether, and what, to encrypt rests with the data owner and is also influenced by the physical measures taken to secure the equipment and media on which the information resides. Organizations may choose to employ different mechanisms to achieve confidentiality and integrity protections, as appropriate. If the confidentiality and integrity of application data is not protected, the data will be open to compromise and unauthorized modification.

The PostgreSQL `pgcrypto` extension provides cryptographic functions for PostgreSQL and is intended to address the confidentiality and integrity of user and system information at rest in non-mobile devices.

Audit:

One possible way to encrypt data within PostgreSQL is to use the `pgcrypto` extension.

To check if `pgcrypto` is installed on PostgreSQL, as a database administrator run the following commands:

```
postgres=# SELECT * FROM pg_available_extensions WHERE name='pgcrypto';

name      | default_version | installed_version |         comment
----------+-----------------+-------------------+-------------------------
pgcrypto  | 1.3             |                   | cryptographic functions
(1 row)
```

If data in the database requires encryption and pgcrypto is not available, this is a fail.

If disk or filesystem requires encryption, ask the system owner, DBA, and SA to demonstrate the use of disk-level encryption. If this is required and is not found, this is a fail. If controls do not exist or are not enabled, this is also a fail.

Remediation:

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

As the database administrator, run the following:

```
postgres=# CREATE EXTENSION pgcrypto;
CREATE EXTENSION
```

Verify `pgcrypto` is installed:

```
postgres=# SELECT * FROM pg_available_extensions WHERE name='pgcrypto';
   name   | default_version | installed_version |         comment
----------+-----------------+-------------------+-------------------------
 pgcrypto | 1.3             | 1.3               | cryptographic functions
(1 row)
```

Impact:

When considering or undertaking any form of encryption, it is critical to understand the state of the encrypted data at all stages of the data lifecycle. The use of `pgcrypto` ensures that the data at rest in the tables (and therefore on disk) is encrypted, but for the data to be accessed by any users or applications, said users/applications will, by necessity, have access to the encrypt and decrypt keys and the data in question will be encrypted/decrypted in memory and then transferred to/from the user/application in that form.

References:

1. http://www.postgresql.org/docs/12/static/pgcrypto.html

CIS Controls:

Version 6

14.5 Encrypt At Rest Sensitive Information
Sensitive information stored on systems shall be encrypted at rest and require a secondary authentication mechanism, not integrated into the operating system, in order to access the information.

Version 7

14.8 Encrypt Sensitive Information at Rest
Encrypt all sensitive information at rest using a tool that requires a secondary authentication mechanism not integrated into the operating system, in order to access the information.

7 Replication

Data redundancy often plays a major role as part of an overall database strategy. Replication is an example of data redundancy and fulfills both High Availability and High Performance requirements. However, although the DBA may have expended much time and effort securing the PRIMARY host and taken the time to harden STANDBY configuration parameters, one sometimes overlooks the medium transmitting the data itself over the network. Consequently, replication is an appealing attack vector given that all DDL, and DML operations executed on the PRIMARY, or master, host is sent over the wire to the SECONDARY/STANDBY, or slave, hosts. Fortunately, when correctly understood, defeating such attacks can be implemented in a straight forward manner. This benchmark reviews those issues surrounding the most common mechanisms of replicating data between hosts. There are several PostgreSQL replication mechanisms and includes:

- Warm Standby (also known as LOG Shipping)
    - Transaction logs are copied from the PRIMARY to SECONDARY host that reads the logs in a "recovery" mode. For all intents and purposes the host ingesting the WAL cannot be read i.e. it's off-line.
- Hot Standby
    - Operates in the exact same fashion as the Warm Standby Server except that, in addition, it offers a read-only environment for client connections to connect and query.
- Point In Time Recovery (PITR)
    - Primarily used for database forensics and recovery at particular points in time such as in the case that important data may have been accidentally removed. One can restore the cluster to a point in time before the event occurred.
- Streaming Replication
    - Uses an explicit connection, which in a manner of speaking is similar to the standard client connection, between the PRIMARY and STANDBY host. It too reads the transaction logs and ingests into a read-only server. What's different is that the connection uses a special replication protocol which is faster and more efficient than log shipping. Similar to standard client connections, it also honors the same authentication rules as expressed in the PostgreSQL host-based authentication file, `pg_hba.conf`.

7.1 Ensure a replication-only user is created and used for streaming replication (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

Create a new user specifically for use by streaming replication instead of using the superuser account.

Rationale:

As it is not necessary to be a superuser to initiate a replication connection, it is proper to create an account specifically for replication. This allows further 'locking down' the uses of the superuser account and follows the general principle of using the least privileges necessary.

Audit:

Check which users currently have the replication permission:

```
postgres=# select rolname from pg_roles where rolreplication is true;
 rolname
----------
 postgres
(1 row)
```

In a default PostgreSQL cluster, only the `postgres` user will have this permission.

Remediation:

It will be necessary to create a new role for replication purposes:

```
postgres=# create user replication_user REPLICATION encrypted password 'XXX';
CREATE ROLE
postgres=# select rolname from pg_roles where rolreplication is true;
     rolname
------------------
 postgres
 replication_user
(2 rows)
```

When using pg_basebackup (or other replication tools) on your standby server, you would use the replication_user (and its password).

Ensure you allow the new user via your pg_hba.conf file:

```
# note that 'replication' in the 2nd column is required and is a special
# keyword, not a real database
hostssl replication     replication_user       0.0.0.0/0               md5
```

References:

1. https://www.postgresql.org/docs/12/static/app-pgbasebackup.html
2. https://www.postgresql.org/docs/12/high-availability.html

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges
Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

4.3 Ensure the Use of Dedicated Administrative Accounts
Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.

7.2 Ensure base backups are configured and functional (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

A 'base backup' is a copy of the PRIMARY host's data cluster ($PGDATA) and is used to create STANDBY hosts and for Point In Time Recovery (PITR) mechanisms. Base backups should be copied across networks in a secure manner using an encrypted transport mechanism. The PostgreSQL CLI `pg_basebackup` can be used, however, SSL encryption should be enabled on the server as per section 6.8 of this benchmark. The pgBackRest tool detailed in section 8.3 of this benchmark can also be used to create a 'base backup'.

Remediation:

Executing base backups using `pg_basebackup` requires the following steps on the **standby** server:

```
$ whoami
postgres
$ pg_basebackup -h name_or_IP_of_master \
-p 5432 \
-U replication_user \
-D ~postgres/11/data \
-P -v -R -Xs \
```

References:

1. https://www.postgresql.org/docs/12/static/functions-admin.html#FUNCTIONS-ADMIN-BACKUP-TABLE
2. https://www.postgresql.org/docs/12/static/app-pgbasebackup.html

CIS Controls:

Version 6

10.2 Test Backups Regularly
Test data on backup media on a regular basis by performing a data restoration process to ensure that the backup is properly working.

10.3 Test Data on Backup Media

Test data integrity on backup media on a regular basis by performing a data restoration process to ensure that the backup is properly working.

7.3 Ensure WAL archiving is configured and functional (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Write Ahead Log (WAL) Archiving, or Log Shipping, is the process of sending transaction log files from the PRIMARY host either to one or more STANDBY hosts or to be archived on a remote storage device for later use, e.g. `PITR`. There are several utilities that can copy WALs including, but not limited to, `cp`, `scp`, `sftp`, and `rynsc`. Basically, the server follows a set of runtime parameters which defines when the WAL should be copied using one of the aforementioned utilities.

Rationale:

Unless the server has been correctly configured, one runs the risk of sending WALs in an unsecured, unencrypted fashion.

Audit:

Review the following runtime parameters in `postgresql.conf`. The following example demonstrates `rsync` but requires that SSH as a transport medium be enabled on the source host:

```
archive_mode = on
archive_command = 'rsync -e ssh -a %p
postgres@remotehost:/var/lib/pgsql/WAL/%f'
```

Confirm SSH public/private keys have been generated on both the source and target hosts in their respective superuser home accounts.

Remediation:

Change parameters and restart the server as required.

Note: SSH public keys must be generated and installed as per industry standards.

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-wal.html#RUNTIME-CONFIG-WAL-ARCHIVING
2. https://linux.die.net/man/1/ssh-keygen
3. https://linux.die.net/man/1/rsync

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

7.4 Ensure streaming replication parameters are configured correctly (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

Streaming replication from a PRIMARY host transmits DDL, DML, passwords, and other potentially sensitive activities and data. These connections should be protected with Secure Sockets Layer (SSL).

Rationale:

Unencrypted transmissions could reveal sensitive information to unauthorized parties. Unauthenticated connections could enable man-in-the-middle attacks.

Audit:

Confirm a dedicated and non-superuser role with replication permission exists:

```
postgres=> select rolname from pg_roles where rolreplication is true;
     rolname
------------------
 postgres
 replication_user
(2 rows)
```

On the target/STANDBY host, execute a `psql` invocation similar to the following, confirming that SSL communications are possible:

```
$ whoami
postgres
$ psql 'host=mySrcHost dbname=postgres user=replication_user
password=mypassword sslmode=require' -c 'select 1;'
```

Remediation:

Review prior sections in this benchmark regarding SSL certificates, replication user, and WAL archiving.

Confirm the file `$PGDATA/standby.signal` is present on the STANDBY host and `$PGDATA/postgresql.auto.conf` contains lines similar to the following:

```
primary_conninfo = 'user=replication_user password=mypassword host=mySrcHost
port=5432 sslmode=require sslcompression=1'
```

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-connection.html#RUNTIME-CONFIG-CONNECTION-SECURITY
2. https://www.postgresql.org/docs/12/static/functions-admin.html#FUNCTIONS-ADMIN-BACKUP-TABLE
3. https://www.postgresql.org/docs/12/static/app-pgbasebackup.html
4. https://www.postgresql.org/docs/12/static/runtime-config-wal.html#RUNTIME-CONFIG-WAL-ARCHIVING
5. https://linux.die.net/man/1/openssl

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

8 Special Configuration Considerations

The recommendations proposed here are to try and address some of the less come use cases which may warrant additional configuration guidance/consideration.

8.1 Ensure PostgreSQL configuration files are outside the data cluster (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL

- Level 1 - PostgreSQL on Linux

Description:

PostgreSQL configuration files within the data cluster's directory tree can be changed by anyone logging into the data cluster as the superuser, i.e. `postgres`. As a matter of default policy, configuration files such as `postgresql.conf`, `pg_hba.conf`, and `pg_ident`, are placed in the data cluster's directory, `$PGDATA`. PostgreSQL can be configured to relocate these files to locations outside the data cluster which cannot then be accessed by an ordinary superuser login session.

Consideration should also be given to "include directives"; these are cluster subdirectories where one can locate files containing additional configuration parameters. Include directives are meant to add more flexibility for unique installs or large network environments while maintaining order and consistent architectural design.

Rationale:

Leaving PostgreSQL configuration files within the data cluster's directory tree increases the changes that they will be inadvertently or intentionally altered.

Audit:

Execute the following commands to verify the configuration is correct:

```
postgres=# select name, setting from pg_settings where name ~ '.*_file$';
         name         |                 setting
----------------------+-----------------------------------------
 config_file          | /var/lib/pgsql/12/data/postgresql.conf
 external_pid_file    |
 hba_file             | /var/lib/pgsql/12/data/pg_hba.conf
 ident_file           | /var/lib/pgsql/12/data/pg_ident.conf
 promote_trigger_file |
 ssl_ca_file          |
 ssl_cert_file        | server.crt
 ssl_crl_file         |
 ssl_dh_params_file   |
 ssl_key_file         | server.key
(10 rows)
```

Execute the following command to see any active include settings:

```
$ grep ^include $PGDATA/postgresql.{auto.,}conf
```

Inspect the file directories and permissions for all returned values. Only superusers and authorized users should have access control rights for these files. If permissions are not highly restricted, this is a fail.

Remediation:

Follow these steps to remediate the configuration file locations and permissions:

- Determine appropriate locations for relocatable configuration files based on your organization's security policies. If necessary, relocate and/or rename configuration files outside of the data cluster.
- Ensure their file permissions are restricted as much as possible, i.e. only superuser read access.
- Change the settings accordingly in the `postgresql.conf` configuration file.
- Restart the database cluster for the changes to take effect.

Default Value:

The defaults for PostgreSQL configuration files are listed below.

```
         name         |                setting

----------------------+----------------------------------------

 config_file          | /var/lib/pgsql/12/data/postgresql.conf

 external_pid_file    |

 hba_file             | /var/lib/pgsql/12/data/pg_hba.conf

 ident_file           | /var/lib/pgsql/12/data/pg_ident.conf

 promote_trigger_file |

 ssl_ca_file          |

 ssl_cert_file        | server.crt

 ssl_crl_file         |
 ssl_dh_params_file    |
 ssl_key_file          | server.key
(10 rows)
```

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-file-locations.html
2. https://www.postgresql.org/docs/12/static/runtime-config-connection.html
3. https://www.postgresql.org/docs/12/static/config-setting.html#CONFIG-INCLUDES

CIS Controls:

Version 6

18.7 Use Standard Database Hardening Templates
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

Version 7

18.11 Use Standard Hardening Configuration Templates for Databases
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

8.2 Ensure PostgreSQL subdirectory locations are outside the data cluster (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL

- Level 1 - PostgreSQL on Linux

Description:

The PostgreSQL cluster is organized to carry out specific tasks in subdirectories. For the purposes of performance, reliability, and security these subdirectories should be relocated outside the data cluster.

Rationale:

Some subdirectories contain information, such as logs, which can be of value to others such as developers. Other subdirectories can gain a performance benefit when placed on fast storage devices. Finally, relocating a subdirectory to a separate and distinct partition mitigates denial of service and involuntary server shutdown when excessive writes fill the data cluster's partition, e.g. `pg_xlog` and `pg_log`.

Audit:

Execute the following SQL statement to verify the configuration is correct. Alternatively, inspect the parameter settings in the `postgresql.conf` configuration file.

```
postgres=# select name, setting from pg_settings where (name ~ '_directory$'
or name ~ '_tablespace');
         name          |          setting
----------------------+-------------------------
 data_directory       | /var/lib/pgsql/12/data
 default_tablespace   |
 log_directory        | ../log
 stats_temp_directory | pg_stat_tmp
 temp_tablespaces     |
(5 rows)
```

Inspect the file and directory permissions for all returned values. Only superusers and authorized users should have access control rights for these files and directories. If permissions are not highly restrictive, this is a fail.

Remediation:

Perform the following steps to remediate the subdirectory locations and permissions:

- Determine appropriate data, log, and tablespace directories and locations based on your organization's security policies. If necessary, relocate all listed directories outside the data cluster.
- Ensure file permissions are restricted as much as possible, i.e. only superuser read access.
- When directories are relocated to other partitions, ensure that they are of sufficient size to mitigate against excessive space utilization.
- Lastly, change the settings accordingly in the `postgresql.conf` configuration file and restart the database cluster for changes to take effect.

Default Value:

The default for `data_directory` is `ConfigDir` and the default for `log_directory` is `log` (based on absolute path of `data_directory`). The defaults for tablespace settings are null, or not set, upon cluster creation.

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-file-locations.html

CIS Controls:

Version 6

18.7 Use Standard Database Hardening Templates
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

Version 7

18.11 Use Standard Hardening Configuration Templates for Databases
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

8.3 Ensure the backup and restore tool, 'pgBackRest', is installed and configured (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

pgBackRest aims to be a simple, reliable backup and restore system that can seamlessly scale up to the largest databases and workloads. Instead of relying on traditional backup tools like `tar` and `rsync`, pgBackRest implements all backup features internally and uses a custom protocol for communicating with remote systems. Removing reliance on `tar` and `rsync` allows for better solutions to database-specific backup challenges. The custom remote protocol allows for more flexibility and limits the types of connections that are required to perform a backup which increases security.

Rationale:

The native PostgreSQL backup facility `pg_dump` provides adequate logical backup operations but does not provide for Point In Time Recovery (PITR). The PostgreSQL facility `pg_basebackup` performs physical backup of the database files and does provide for PITR, but it is constrained by single threading. Both of these methodologies are standard in the PostgreSQL ecosystem and appropriate for particular backup/recovery needs. `pgBackRest` offers another option with much more robust features and flexibility.

`pgBackRest` is open source software developed to perform efficient backups on PostgreSQL databases that measure in tens of terabytes and greater. It supports per file checksums, compression, partial/failed backup resume, high-performance parallel transfer, asynchronous archiving, tablespaces, expiration, full/differential/incremental, local/remote operation via SSH, hard-linking, restore, **backup encryption**, and more. `pgBackRest` is written in C and Perl and does not depend on `rsync` or `tar` but instead performs its own deltas which gives it maximum flexibility. Finally, `pgBackRest` provides an easy to use internal repository listing backup details accessible via the `pgbackrest info` command, as illustrated below.

```
$ pgbackrest info
stanza: proddb01
status: ok

db (current)
  wal archive min/max (12.0-1): 000000010000000000000012 /
000000010000000000000017
```

```
      full backup: 20190603-153106F
          timestamp start/stop: 2019-06-03 15:31:06 / 2019-06-03 15:31:49
          wal start/stop: 000000010000000000000012 / 000000010000000000000012
          database size: 29.4MB, backup size: 29.4MB
          repository size: 3.4MB, repository backup size: 3.4MB

      diff backup: 20190603-153106F_20181002-173109D
          timestamp start/stop: 2019-06-03 17:31:09 / 2019-06-03 17:31:19
          wal start/stop: 000000010000000000000015 / 000000010000000000000015
          database size: 29.4MB, backup size: 2.6MB
          repository size: 3.4MB, repository backup size: 346.8KB
          backup reference list: 20190603-153106F

      incr backup: 20190603-153106F_20181002-183114I
          timestamp start/stop: 2019-06-03 18:31:14 / 2019-06-03 18:31:22
          wal start/stop: 000000010000000000000017 / 000000010000000000000017
          database size: 29.4MB, backup size: 8.2KB
          repository size: 3.4MB, repository backup size: 519B
          backup reference list: 20190603-153106F, 20190603-153106F_20190603-
173109D
```

Audit:

If installed, invoke it without arguments to see the help:

```
$ # not installed
# pgbackrest
-bash: pgbackrest: command not found
$ # instlled
$ pgbackrest
pgBackRest 2.18 - General help

Usage:
    pgbackrest [options] [command]

Commands:
    archive-get     Get a WAL segment from the archive.
    archive-push    Push a WAL segment to the archive.
    backup          Backup a database cluster.
    check           Check the configuration.
    expire          Expire backups that exceed retention.
    help            Get help.
    info            Retrieve information about backups.
    restore         Restore a database cluster.
    stanza-create   Create the required stanza data.
    stanza-delete   Delete a stanza.
    stanza-upgrade  Upgrade a stanza.
    start           Allow pgBackRest processes to run.
    stop            Stop pgBackRest processes from running.
    version         Get version.

Use 'pgbackrest help [command]' for more information.
```

Remediation:

`pgBackRest` is not installed nor configured for PostgreSQL by default, but instead is maintained as a GitHub project. Fortunately, it is a part of the PGDG repository and can be easily installed:

```
$ whoami
root
$ dnf -y install pgbackrest
Last metadata expiration check: 0:00:19 ago on Tue 29 Oct 2019 12:30:51 PM
EDT.
Dependencies resolved.
[snip]
Installed:
  pgbackrest-2.18-1.rhel8.x86_64                          perl-DBD-Pg-3.7.4-
2.module_el8.0.0+74+7e750437.x86_64
  perl-DBI-1.641-2.module_el8.0.0+66+fe1eca09.x86_64      perl-Data-Dump-
1.23-7.el8.noarch
  perl-Digest-HMAC-1.03-17.el8.noarch                     perl-File-Listing-
6.04-17.el8.noarch
  perl-HTML-Parser-3.72-14.el8.x86_64                     perl-HTML-Tagset-
3.20-33.el8.noarch
  perl-HTTP-Cookies-6.04-2.el8.noarch                     perl-HTTP-Date-
6.02-18.el8.noarch
  perl-HTTP-Message-6.18-1.el8.noarch                     perl-HTTP-
Negotiate-6.01-19.el8.noarch
  perl-IO-HTML-1.001-10.el8.noarch                        perl-LWP-
MediaTypes-6.02-14.el8.noarch
  perl-NTLM-1.09-17.el8.noarch                            perl-Net-HTTP-6.17-
2.el8.noarch
  perl-TimeDate-1:2.30-13.el8.noarch                      perl-Try-Tiny-0.30-
2.el8.noarch
  perl-WWW-RobotRules-6.02-18.el8.noarch                  perl-XML-LibXML-
1:2.0132-2.el8.x86_64
  perl-XML-NamespaceSupport-1.12-4.el8.noarch             perl-XML-SAX-1.00-
1.el8.noarch
  perl-XML-SAX-Base-1.09-4.el8.noarch                     perl-libwww-perl-
6.34-1.el8.noarch

Complete!
```

Once installed, `pgBackRest` must be configured for things like stanza name, backup location, retention policy, logging, etc. Please consult the configuration guide.

If employing `pgBackRest` for your backup/recovery solution, ensure the repository, base backups, and WAL archives are stored on a reliable file system separate from the database server. Further, the external storage system where backups resided should have limited access to only those system administrators as necessary. Finally, as with any backup/recovery solution, stringent testing must be conducted. **A backup is only good if it can be restored successfully.**

References:

1. https://pgbackrest.org/
2. https://github.com/pgbackrest/pgbackrest
3. https://www.postgresql.org/docs/12/static/app-pgdump.html
4. https://www.postgresql.org/docs/12/static/app-pgbasebackup.html

CIS Controls:

Version 6

10 Data Recovery Capability
Data Recovery Capability

Version 7

10.1 Ensure Regular Automated Back Ups
Ensure that all system data is automatically backed up on regular basis.

10.2 Perform Complete System Backups
Ensure that each of the organization's key systems are backed up as a complete system, through processes such as imaging, to enable the quick recovery of an entire system.

8.4 Ensure miscellaneous configuration settings are correct (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL
- Level 1 - PostgreSQL on Linux

Description:

This recommendation covers non-regular, special files, and dynamic libraries.

PostgreSQL permits local logins via the UNIX DOMAIN SOCKET and, for the most part, anyone with a legitimate Unix login account can make the attempt. Limiting PostgreSQL login attempts can be made by relocating the UNIX DOMAIN SOCKET to a subdirectory with restricted permissions.

The creation and implementation of user-defined dynamic libraries is an extraordinary powerful capability. In the hands of an experienced DBA/programmer, it can significantly enhance the power and flexibility of the RDBMS. But new and unexpected behavior can also be assigned to the RDBMS, resulting in a very dangerous environment in what should otherwise be trusted.

Rationale:

Audit:

Execute the following SQL statement to verify the configuration is correct. Alternatively, inspect the parameter settings in the `postgresql.conf` configuration file.

```
postgres=# select name, setting from pg_settings where name in
('external_pid_file',
'unix_socket_directories','shared_preload_libraries','dynamic_library_path','
local_preload_libraries','session_preload_libraries');
           name            |          setting
---------------------------+--------------------------
 dynamic_library_path      | $libdir
 external_pid_file         |
 local_preload_libraries   |
 session_preload_libraries |
 shared_preload_libraries  | pgaudit, set_user
 unix_socket_directories   | /var/run/postgresql, /tmp
(6 rows)
```

Inspect the file and directory permissions for all returned values. Only superusers should have access control rights for these files and directories. If permissions are not highly restricted, this is a fail.

Remediation:

Follow these steps to remediate the configuration:

- Determine permissions based on your organization's security policies.
- Relocate all files and ensure their permissions are restricted as much as possible, i.e. only superuser read access.
- Ensure all directories where these files are located have restricted permissions such that the superuser can read but not write.
- Lastly, change the settings accordingly in the `postgresql.conf` configuration file and restart the database cluster for changes to take effect.

Default Value:

The `dynamic_library_path` default is `$libdir` and unix_socket_directories default is `/var/run/postgresql, /tmp`. The default for `external_pid_file` and all library parameters are initially null, or not set, upon cluster creation.

References:

1. https://www.postgresql.org/docs/12/static/runtime-config-file-locations.html
2. https://www.postgresql.org/docs/12/static/runtime-config-connection.html
3. https://www.postgresql.org/docs/12/static/runtime-config-client.html

CIS Controls:

Version 6

18.7 Use Standard Database Hardening Templates
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

Version 7

18.11 Use Standard Hardening Configuration Templates for Databases
For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.

Appendix: Summary Table

Control		Set Correctly	
		Yes	No
1	**Installation and Patches**		
1.1	Ensure packages are obtained from authorized repositories (Not Scored)	□	□
1.2	Ensure Installation of Binary Packages (Not Scored)	□	□
1.3	Ensure Installation of Community Packages (Not Scored)	□	□
1.4	Ensure systemd Service Files Are Enabled (Scored)	□	□
1.5	Ensure Data Cluster Initialized Successfully (Scored)	□	□
2	**Directory and File Permissions**		
2.1	Ensure the file permissions mask is correct (Scored)	□	□
2.2	Ensure the PostgreSQL pg_wheel group membership is correct (Scored)	□	□
3	**Logging Monitoring And Auditing**		
3.1	**PostgreSQL Logging**		
3.1.1	**Logging Rationale**		
3.1.2	Ensure the log destinations are set correctly (Scored)	□	□
3.1.3	Ensure the logging collector is enabled (Scored)	□	□
3.1.4	Ensure the log file destination directory is set correctly (Scored)	□	□
3.1.5	Ensure the filename pattern for log files is set correctly (Scored)	□	□
3.1.6	Ensure the log file permissions are set correctly (Scored)	□	□
3.1.7	Ensure 'log_truncate_on_rotation' is enabled (Scored)	□	□
3.1.8	Ensure the maximum log file lifetime is set correctly (Scored)	□	□
3.1.9	Ensure the maximum log file size is set correctly (Scored)	□	□
3.1.10	Ensure the correct syslog facility is selected (Scored)	□	□
3.1.11	Ensure the program name for PostgreSQL syslog messages is correct (Scored)	□	□
3.1.12	Ensure the correct messages are written to the server log (Not Scored)	□	□
3.1.13	Ensure the correct SQL statements generating errors are recorded (Not Scored)	□	□
3.1.14	Ensure 'debug_print_parse' is disabled (Scored)	□	□
3.1.15	Ensure 'debug_print_rewritten' is disabled (Scored)	□	□
3.1.16	Ensure 'debug_print_plan' is disabled (Scored)	□	□
3.1.17	Ensure 'debug_pretty_print' is enabled (Scored)	□	□
3.1.18	Ensure 'log_connections' is enabled (Scored)	□	□
3.1.19	Ensure 'log_disconnections' is enabled (Scored)	□	□

3.1.20	Ensure 'log_error_verbosity' is set correctly (Not Scored)	□	□
3.1.21	Ensure 'log_hostname' is set correctly (Scored)	□	□
3.1.22	Ensure 'log_line_prefix' is set correctly (Not Scored)	□	□
3.1.23	Ensure 'log_statement' is set correctly (Scored)	□	□
3.1.24	Ensure 'log_timezone' is set correctly (Scored)	□	□
3.2	Ensure the PostgreSQL Audit Extension (pgAudit) is enabled (Scored)	□	□
4	**User Access and Authorization**		
4.1	Ensure sudo is configured correctly (Scored)	□	□
4.2	Ensure excessive administrative privileges are revoked (Scored)	□	□
4.3	Ensure excessive function privileges are revoked (Scored)	□	□
4.4	Ensure excessive DML privileges are revoked (Scored)	□	□
4.5	Use pg_permission extension to audit object permissions (Not Scored)	□	□
4.6	Ensure Row Level Security (RLS) is configured correctly (Not Scored)	□	□
4.7	Ensure the set_user extension is installed (Not Scored)	□	□
4.8	Make use of default roles (Not Scored)	□	□
5	**Connection and Login**		
5.1	Ensure login via "local" UNIX Domain Socket is configured correctly (Not Scored)	□	□
5.2	Ensure login via "host" TCP/IP Socket is configured correctly (Scored)	□	□
6	**PostgreSQL Settings**		
6.1	Ensure 'Attack Vectors' Runtime Parameters are Configured (Not Scored)	□	□
6.2	Ensure 'backend' runtime parameters are configured correctly (Scored)	□	□
6.3	Ensure 'Postmaster' Runtime Parameters are Configured (Not Scored)	□	□
6.4	Ensure 'SIGHUP' Runtime Parameters are Configured (Not Scored)	□	□
6.5	Ensure 'Superuser' Runtime Parameters are Configured (Not Scored)	□	□
6.6	Ensure 'User' Runtime Parameters are Configured (Not Scored)	□	□
6.7	Ensure FIPS 140-2 OpenSSL Cryptography Is Used (Scored)	□	□
6.8	Ensure SSL is enabled and configured correctly (Scored)	□	□
6.9	Ensure the pgcrypto extension is installed and configured correctly (Not Scored)	□	□

7	**Replication**		
7.1	Ensure a replication-only user is created and used for streaming replication (Not Scored)	□	□
7.2	Ensure base backups are configured and functional (Not Scored)	□	□
7.3	Ensure WAL archiving is configured and functional (Scored)	□	□
7.4	Ensure streaming replication parameters are configured correctly (Not Scored)	□	□
8	**Special Configuration Considerations**		
8.1	Ensure PostgreSQL configuration files are outside the data cluster (Not Scored)	□	□
8.2	Ensure PostgreSQL subdirectory locations are outside the data cluster (Not Scored)	□	□
8.3	Ensure the backup and restore tool, 'pgBackRest', is installed and configured (Not Scored)	□	□
8.4	Ensure miscellaneous configuration settings are correct (Not Scored)	□	□

Appendix: Change History

Date	Version	Changes for this version
Nov 19, 2019	1.0.0	Initial Release