CIS Benchmarks

CIS PostgreSQL 9.5 Benchmark

v1.0.0 - 07-13-2018

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 9.5 Benchmark, provides prescriptive guidance for establishing a secure configuration posture for PostgreSQL 9.5. This guide was tested against PostgreSQL 9.5 running on CentOS 6, 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 9.5.

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 on Linux**

  Items in this profile apply to PostgreSQL 9.5 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
Robert Bernier
Doug Hunley
Duane Rensby

Contributor
Philippe Langlois
Paul Harrington
Tairo Mendoza
Robert Blok
Kelvin Tan Thiam Teck CEH
Scott Mead

Editor
Karen Scarfone
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 `yum`), it's imperative that packages are sourced only from valid and authorized repositories. For PostgreSQL, a short list of valid repositories would include CentOS (www.centos.org) and the official PostgreSQL website (yum.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 yum 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 6 install illustrating the use of the `yum repolist all` command.

```
$ yum repolist all | grep enabled:
base                    CentOS-6 - Base                 enabled: 6,696
extras                  CentOS-6 - Extras               enabled:    62
updates                 CentOS-6 - Updates              enabled:   581
```

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.
Here is an example of adding an authorized repository:

1. Install the PGDG repository RPM from `yum.postgresql.org`

```
$ rpm -ivh
https://download.postgresql.org/pub/repos/yum/9.5/redhat/rhel-6-
x86_64/pgdg-centos95-9.5-3.noarch.rpm
Retrieving
https://download.postgresql.org/pub/repos/yum/9.5/redhat/rhel-6-
x86_64/pgdg-centos95-9.5-3.noarch.rpm
warning: /var/tmp/rpm-tmp.DAPqyf: Header V4 DSA/SHA1 Signature, key ID
442df0f8: NOKEY
Preparing...                ########################################
[100%]
   1:pgdg-centos95          ########################################
[100%]
```

2. Verify the repository has been added and is enabled.

```
$ yum repolist all | grep enabled:
base                    CentOS-6 - Base                 enabled: 6,696
extras                  CentOS-6 - Extras               enabled:    62
updates                 CentOS-6 - Updates              enabled:   581
pgdg                    PostgreSQL 9.5 6 - x86_64       enabled:   406
```

References:

1. http://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/Yellowdog_Updater,_Modified
4. https://www.howtoforge.com/creating_a_local_yum_repository_centos

CIS Controls:

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

1.2 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 Postgres, and no upgrade path migrating from one version of Postgres to the next. The worst set of circumstances is to be limited to a version of the DBMS 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://www.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 Postgres.

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. We will illustrate two methods `rpm` and `yum`. In both cases nothing is returned.

```
$ rpm -qa | grep postgres
$

$ yum list installed postgres*
Checksum type 'md5' disabled
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
* base: mirrors.gigenet.com
* epel: mirror.compevo.com
* extras: mirror.us-midwest-1.nexcess.net
* updates: pubmirrors.dal.corespace.com
Error: No matching Packages to list
```

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

Remediation:

Using a web browser, go to http://yum.postgresql.org and navigate to the repo download link for your OS and version. The following example blocks the outdated distro packages, adds the PGDG repository RPM for PostgreSQL version 9.5, and installs the client-server-contributions rpms to the host where you want to install the RDBMS:

```
$ vi /etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
repo
=os&infra=$infra
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
exclude=postgresql*  <-- add this line

#released updates
[updates]
name=CentOS-$releasever - Updates
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
repo
=updates&infra=$infra
#baseurl=http://mirror.centos.org/centos/$releasever/updates/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
exclude=postgresql*  <-- add this line

$ yum -y install https://yum.postgresql.org/9.5/redhat/rhel-6-x86_64/pgdg-
redhat95-9.5-3.noarch.rpm

$ yum -y groupinstall "PostgreSQL Database Server 9.5 PGDG"
Loaded plugins: fastestmirror
Setting up Group Process
Loading mirror speeds from cached hostfile
 * base: ftp.osuosl.org
 * extras: repo.us.bigstepcloud.com
 * updates: repo1.dal.innoscale.net
Resolving Dependencies
--> Running transaction check
---> Package postgresql95.x86_64 0:9.5.6-5PGDG.rhel6 will be installed
---> Package postgresql95-contrib.x86_64 0:9.5.6-5PGDG.rhel6 will be
installed
---> Package postgresql95-libs.x86_64 0:9.5.6-5PGDG.rhel6 will be installed
---> Package postgresql95-server.x86_64 0:9.5.6-5PGDG.rhel6 will be
installed
--> Finished Dependency Resolution

Dependencies Resolved
```

```
================================================================================
====
 Package                    Arch        Version                 Repository
Size
================================================================================
====
Installing:
 postgresql95               x86_64      9.5.6-5PGDG.rhel6       pgdg95
1.4 M
 postgresql95-contrib       x86_64      9.5.6-5PGDG.rhel6       pgdg95
491 k
 postgresql95-libs          x86_64      9.5.6-5PGDG.rhel6       pgdg95
285 k
 postgresql95-server        x86_64      9.5.6-5PGDG.rhel6       pgdg95
4.8 M

Transaction Summary
================================================================================
====
Install       4 Package(s)

Total download size: 6.9 M
Installed size: 27 M
Downloading Packages:
(1/4): postgresql95-9.5.6-5PGDG.rhel6.x86_64.rpm          | 1.4 MB     00:00
(2/4): postgresql95-contrib-9.5.6-5PGDG.rhel6.x86_64.rpm  | 491 kB     00:00
(3/4): postgresql95-libs-9.5.6-5PGDG.rhel6.x86_64.rpm     | 285 kB     00:00
(4/4): postgresql95-server-9.5.6-5PGDG.rhel6.x86_64.rpm   | 4.8 MB     00:00
--------------------------------------------------------------------------------
----
Total                                           2.1 MB/s | 6.9 MB     00:03
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : postgresql95-libs-9.5.6-5PGDG.rhel6.x86_64
1/4
  Installing : postgresql95-9.5.6-5PGDG.rhel6.x86_64
2/4
  Installing : postgresql95-server-9.5.6-5PGDG.rhel6.x86_64
3/4
  Installing : postgresql95-contrib-9.5.6-5PGDG.rhel6.x86_64
4/4
  Verifying  : postgresql95-libs-9.5.6-5PGDG.rhel6.x86_64
1/4
  Verifying  : postgresql95-server-9.5.6-5PGDG.rhel6.x86_64
2/4
  Verifying  : postgresql95-contrib-9.5.6-5PGDG.rhel6.x86_64
3/4
  Verifying  : postgresql95-9.5.6-5PGDG.rhel6.x86_64
4/4

Installed:
  postgresql95.x86_64 0:9.5.6-5PGDG.rhel6
  postgresql95-contrib.x86_64 0:9.5.6-5PGDG.rhel6
  postgresql95-libs.x86_64 0:9.5.6-5PGDG.rhel6
  postgresql95-server.x86_64 0:9.5.6-5PGDG.rhel6
```

```
Complete!
```

Note: The above-mentioned example is referenced as an illustration only. The distro, and version, varies according to one's requirements. Visit the Postgres community web portal for more information.

References:

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

CIS Controls:

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.

1.3 Ensure Installation of Binary Packages (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The PostgreSQL package(s) 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 the server and client packages. Contribution modules are optional depending upon one's architectural requirements.

From a security perspective, it's imperative to verify PostgreSQL binary packages are sourced from a valid Linux yum repository. The most common Linux repositories include RHEL base, CentOS base and PGDG base; however, it's up to the organization to validate. For a complete listing of all PostgreSQL binaries available via configured repositories inspect the output from `yum provides postgres*`.

Audit:

To inspect what versions of PostgreSQL packages are installed we can query using the `rpm` command. As illustrated below, PostgreSQL 8.4 packages are installed:

```
# rpm -qa|grep postgres
postgresql-8.4.20-8.el6_9.x86_64
postgresql-server-8.4.20-8.el6_9.x86_64
postgresql-libs-8.4.20-8.el6_9.x86_64
postgresql-contrib-8.4.20-8.el6_9.x86_64
```

If the expected binary packages are not installed or not the expected versions, this is a fail.

Remediation:

If the version of PostgreSQL installed is not 9.5.x, the packages may be uninstalled using this command:

```
yum 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:

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

1.4 Ensure Service Runlevel Is Registered And Set Correctly (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Confirm, and set if necessary, the PostgreSQL runlevel on system-V operating systems.

Rationale:

Setting the runlevel on a System V OS ensures the database service is active especially when a change of state occurs as in the case of a system startup, reboot or an explicit change of runlevel by the sys-admin.

Audit:

The default run-level on RedHat/CentOS operating systems is typically "3". One confirms the default run-level and those run-levels desirous of running PostgreSQL by executing the following;

```
$ grep initdefault: /etc/inittab
id:3:initdefault:

$ chkconfig --list | grep postgres
postgresql-9.5       0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

If the intended PostgreSQL service is not registered (no output for the`chkconfig` command) or is not configured to appropriate runlevel (`3:off` above), this is a fail.

Remediation:

Irrespective of package source, PostgreSQL services can be identified because it typically includes the text string "postgresql". Correct installs automatically register the service although it may still be off. Multiple instance of postgres services often distinguish themselves using a version number. Unregistered services must be added before its runlevel can be administrated.

```
$ chkconfig --add postgresql-9.5

$ chkconfig --level 3 postgresql-9.5 on

$ chkconfig --list | grep postgres
postgresql-9.5       0:off   1:off   2:off   3:on    4:off   5:off   6:off
```

References:

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

CIS Controls:

18 Application Software Security
Application Software Security

1.5 Ensure Data Cluster Initialized Successfully (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Create a new PostgreSQL database cluster. 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 Postgres 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 installing the Postgres binary package from either the CENTOS 6, or Community repository (rpm) installation; the standard method, as root, is to instantiate the cluster thusly:

```
$ service postgresql-9.5 initdb
Initializing database:                                     [  OK  ]
```

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

```
ls -al ~postgres/9.5
total 20
drwxr-xr-x  4 postgres postgres 4096 Nov  5 16:46 .
drwx------  3 postgres postgres 4096 Nov  5 15:45 ..
drwx------ 19 postgres postgres 4096 Nov  5 16:46 data
-rw-------  1 postgres postgres 1380 Nov  5 16:46 pgstartup.log
```

Remediation:

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

```
$ service postgresql-9.5 initdb
Data directory is not empty!
[root@pg1_centos ~]#                                        [FAILED]
```

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

```
$ rm -rf ~postgres/9.5
$ service postgresql-9.5 initdb
```

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`.

Audit:

To view the mask's current setting, as the postgres user, execute the command:

```
# umask
077
```

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:

```
umask 077
```

Default Value:

002

CIS Controls:

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.

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 Postgres server is installed. Membership in this group enables an ordinary user account to gain superuser access to a machine by using the su command. 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 Postgres 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 `groups` to confirm that a `pg_wheel` group exists. If no such group exists, this is a fail. If such a group does exist, view its membership and confirm that each user is authorized to act as an administrator.

Remediation:

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

```
$ getent group pg_wheel || 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:

```
$ gpasswd -a postgres pg_wheel
Adding user postgres to group pg_wheel
```

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

```
$ gpasswd -d pg_wheel user1
Removing user user1 from group pg_wheel
$ groups user1
user1 : user1
```

References:

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

CIS Controls:

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.

3 Logging Monitoring And Auditing (Centos 6)

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 auditing and logging behavior.

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 it is, it's cause, and what it's doing to the system.

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

The Postgres 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 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 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 finding.

Remediation:

Execute the following SQL statement(s) 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.

Default Value:

`stderr`

Notes:

`logging_collector` (section 3.1.2) must be enabled to generate CSV-format log output.

CIS Controls:

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.

3.1.3 Ensure the logging collector is enabled (Scored)

Profile Applicability:

- 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 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
```

As root, restart the PostgreSQL service for this change to take effect:

```
$ # service postgresql-9.5 restart
Stopping postgresql-9.5 service:                           [  OK  ]
Starting postgresql-9.5 service:                           [  OK  ]
```

Default Value:

`on` in the PGDG packages

CIS Controls:

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.

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

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The `log_directory` setting specifies the destination directory for log files. It can be specified as relative to the cluster data directory 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).

Audit:

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

```
postgres=# show log_directory;
 log_directory
---------------
 pg_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='logs';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`pg_log`

CIS Controls:

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.

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

Profile Applicability:

- 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)
```

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)
```

Default Value:

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

References:

1. http://man7.org/linux/man-pages/man3/strftime.3.html

CIS Controls:

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.

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

Profile Applicability:

- 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)
```

Default Value:

0600

CIS Controls:

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.

3.1.7 Ensure 'log_truncate_on_rotation' is enabled (Scored)

Profile Applicability:

- 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 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.

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
log_truncate_on_rotation = on
log_filename = 'postgresql-%Y%m%d.log'
log_rotation_age = '1h'
log_rotation_size = 0
# truncation is indeterminate (how quickly are you generating 100M of log
data)
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
--------------------------
 on
(1 row)
```

Remediation:

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

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

Default Value:

'on'

Notes:

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

CIS Controls:

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.

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

Profile Applicability:

- 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. After that 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)

CIS Controls:

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.

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

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The `log_rotation_size` setting determines the maximum size in kilobytes 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.

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

CIS Controls:

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.

3.1.10 Ensure the correct syslog facility is selected (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The `syslog_facility` setting specifies the syslog "facility" to be used when syslog is enabled. You can choose from any of the 'local' facilities (`LOCAL0` - `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

CIS Controls:

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

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

Profile Applicability:

- 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 "pg95"):

```
postgres=# alter system set syslog_ident = 'pg95';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`postgres`

References:

1. https://tools.ietf.org/html/rfc3164#section-4.1.3

CIS Controls:

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

3.1.12 Ensure the correct messages are sent to the database client (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The `client_min_messages` setting specifies the message levels that are sent to the database client (not the logs). Each level includes all the levels that follow it. The later the level, the fewer messages are sent. `NOTICE` is generally accepted as the best practice for this setting.

Valid values are:

- `DEBUG5`
- `DEBUG4`
- `DEBUG3`
- `DEBUG2`
- `DEBUG1`
- `LOG`
- `NOTICE`
- `WARNING`
- `ERROR`
- `FATAL`
- `PANIC`

Note: `LOG` has a different rank here than in `log_min_messages`.

Rationale:

If this is not set correctly, the database client may receive too many messages or too few messages.

Audit:

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

```
postgres=# show client_min_messages;
 client_min_messages
---------------------
 notice
(1 row)
```

Remediation:

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

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

Default Value:

`NOTICE`

CIS Controls:

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

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

Profile Applicability:

- 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 later the level, the fewer messages are sent.

Valid values are:

- `DEBUG5`
- `DEBUG4`
- `DEBUG3`
- `DEBUG2`
- `DEBUG1`
- `INFO`
- `NOTICE`
- `WARNING`
- `ERROR`
- `LOG`
- `FATAL`
- `PANIC`

Note: `LOG` has a different rank here than in `client_min_messages`.

`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`

CIS Controls:

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

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

Profile Applicability:

- 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 later the level, the fewer messages are recorded. Valid values are:

- `DEBUG5`
- `DEBUG4`
- `DEBUG3`
- `DEBUG2`
- `DEBUG1`
- `INFO`
- `NOTICE`
- `WARNING`
- `ERROR`
- `LOG`
- `FATAL`
- `PANIC`

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

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

Rationale:

If this is not set to the correct value, too many SQL statements or too few 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`

CIS Controls:

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

3.1.15 Ensure 'log_min_duration_statement' is disabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The `log_min_duration_statement` setting specifies the minimum execution time for a statement at which the statement will be logged. For example, if you set it to `250ms`, then all SQL statements that run `250ms` or longer will be logged. Setting it to `-1` disables this feature, which is recommended. Setting it to `0` records all statements regardless of duration.

Rationale:

Logging of SQL statements may include sensitive information that should not be recorded in logs.

Audit:

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

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

Remediation:

Execute the following SQL statement(s) as superuser to remediate this setting (in this example, to `-1`):

```
postgres=# alter system set log_min_duration_statement = -1;
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`-1`

CIS Controls:

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

3.1.16 Ensure 'debug_print_parse' is disabled (Scored)

Profile Applicability:

- 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`

CIS Controls:

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

3.1.17 Ensure 'debug_print_rewritten' is disabled (Scored)

Profile Applicability:

- 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`

CIS Controls:

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

3.1.18 Ensure 'debug_print_plan' is disabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The `debug_print_rewritten` 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`

CIS Controls:

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

3.1.19 Ensure 'debug_pretty_print' is enabled (Scored)

Profile Applicability:

- 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`.

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)
```

Default Value:

`on`

CIS Controls:

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

3.1.20 Ensure 'log_checkpoints' is enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_checkpoints` setting causes `checkpoints` and `restartpoints` to be logged in the server log. Some statistics are included in the log messages, including the number of buffers written and the time spent writing them.

Rationale:

Enabling the logging of checkpoints is the easiest method of tracking both the frequency and duration of the checkpoint operations.

Audit:

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

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

Remediation:

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

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

Default Value:

`off`

CIS Controls:

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

3.1.21 Ensure 'log_connections' is enabled (Scored)

Profile Applicability:

- 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 maintain an internal record of attempted connections to the database for later auditing. It is only be enabling the logging of these attempts that one can determine if unexpected attempts are being made.

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)
```

As root, restart the PostgreSQL service:

```
[root@localhost ~]# service postgresql-9.5 restart
Stopping postgresql-9.5 service:                           [  OK  ]
Starting postgresql-9.5 service:                           [  OK  ]
```

Default Value:

`off`

CIS Controls:

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

3.1.22 Ensure 'log_disconnections' is enabled (Scored)

Profile Applicability:

- 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.

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)
```

As root, restart the PostgreSQL service:

```
[root@localhost ~]# service postgresql-9.5 restart
Stopping postgresql-9.5 service:                           [  OK  ]
Starting postgresql-9.5 service:                           [  OK  ]
```

Default Value:

`off`

CIS Controls:

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

3.1.23 Ensure 'log_duration' is enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_duration` setting causes the duration of each completed SQL statement to be logged. For clients using the extended query protocol, durations of the `Parse`, `Bind`, and `Execute` steps are logged independently.

Rationale:

By logging the duration of statements, it is easy to identify both non-performant queries as well as possible DoS attempts (excessively long running queries may be attempts at resource starvation).

Audit:

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

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

Remediation:

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

```
postgres=# alter system set log_duration = `on`;
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`off`

CIS Controls:

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

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

Profile Applicability:

- 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`

CIS Controls:

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

3.1.25 Ensure 'log_hostname' is set correctly (Scored)

Profile Applicability:

- 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.

Rationale:

Depending on your hostname resolution setup, enabling this setting might impose a non-negligible performance penalty.

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`

CIS Controls:

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

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

Profile Applicability:

- 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. The default value is `< %m >`.

`%` 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.

The default is `< %m >`, but 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 = '%t [%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 >`

References:

1. https://dalibo.github.io/pgbadger/index.html

CIS Controls:

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

3.1.27 Ensure 'log_lock_waits' is enabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

The `log_lock_waits` setting specifies whether a log message is produced when a session waits longer than `deadlock_timeout` to acquire a lock. The setting should be enabled (set to `on`) unless otherwise directed by your organization's logging policy.

Rationale:

If this setting is disabled, it may be harder to determine if lock waits are causing poor performance or if a specially-crafted SQL is attempting to starve resources through holding locks for excessive amounts of time.

Audit:

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

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

Remediation:

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

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

Default Value:

`off`

CIS Controls:

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

3.1.28 Ensure 'log_statement' is set correctly (Scored)

Profile Applicability:

- 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, such as `CREATE`, `ALTER`, and `DROP` statements.

`mod` logs all `ddl` statements, plus data-modifying statements such as `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, and `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
---------------
 ddl
(1 row)
```

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

CIS Controls:

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

3.1.29 Ensure all temporary files are logged (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Temporary files are created for sorts, hashes, and temporary query results when these operations exceed `work_mem`. A log entry is made for each temporary file when it is deleted. Setting `log_temp_files` to `0` causes all temporary file information to be logged, while positive values log only files whose size is greater than or equal to the specified number of kilobytes. A value of `-1` disables temporary file information logging.

Unless directed otherwise by your organization's logging policy, you should set this to `0`.

Rationale:

If all temporary files are not logged, it may be more difficult to identify potential performance issues that may be either poor application coding or deliberate resource starvation attempts.

Audit:

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

```
postgres=# show log_temp_files ;
 log_temp_files
----------------
 0
(1 row)
```

Remediation:

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

```
postgres=# alter system set log_temp_files = 0;
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)
```

Default Value:

`-1`

CIS Controls:

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

3.1.30 Ensure 'log_timezone' is set correctly (Scored)

Profile Applicability:

- 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 timezone as defined by your organization's logging policy to ensure a lack of confusion around when a logged event occurred.

Audit:

Execute the following SQL statement:

```
postgres=# show log_timezone ;
 log_timezone
--------------
 GMT
(1 row)
```

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:

`GMT`

CIS Controls:

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

3.1.31 Ensure 'log_parser_stats' is disabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_parser_stats` setting causes parser performance statistics to be written to the server log. This is a crude profiling instrument, similar to the Unix `getrusage()` operating system facility. This module reports per-module statistics. The parser performance statistics logging is disabled (`off`) by default and should only be enabled if directed to do so by your organization's logging policy.

Rationale:

The logging of these additional statistics when not mandated by your organization's logging policy greatly reduces the signal-to-noise ratio of the PostgreSQL logs.

Audit:

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

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

Remediation:

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

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

Default Value:

`off`

References:

1. http://man7.org/linux/man-pages/man2/getrusage.2.html

CIS Controls:

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

3.1.32 Ensure 'log_planner_stats' is disabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_planner_stats` setting causes planner performance statistics to be written to the server log. This is a crude profiling instrument, similar to the Unix getrusage() operating system facility. This module reports per-module statistics. The planner performance statistics logging is disabled (off) by default and should only be enabled if directed to do so by your organization's logging policy.

Rationale:

The logging of these additional statistics when not mandated by your organization's logging policy greatly reduces the signal-to-noise ratio of the PostgreSQL logs.

Audit:

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

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

Remediation:

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

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

Default Value:

`off`

References:

1. http://man7.org/linux/man-pages/man2/getrusage.2.html

CIS Controls:

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

3.1.33 Ensure 'log_executor_stats' is disabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_executor_stats` setting causes executor performance statistics to be written to the server log. This is a crude profiling instrument, similar to the Unix getrusage() operating system facility. This module reports per-module statistics. The executor performance statistics logging is disabled (off) by default and should only be enabled if directed to do so by your organization's logging policy.

Rationale:

The logging of these additional statistics when not mandated by your organization's logging policy greatly reduces the signal-to-noise ratio of the PostgreSQL logs.

Audit:

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

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

Remediation:

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

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

Default Value:

`off`

References:

1. http://man7.org/linux/man-pages/man2/getrusage.2.html

CIS Controls:

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

3.1.34 Ensure 'log_statement_stats' is disabled (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Enabling the `log_statement_stats` setting causes cumulative performance statistics to be written to the server log for each query. This is a crude profiling instrument, similar to the Unix getrusage() operating system facility. This reports total statement statistics. Cumulative performance statistics logging is disabled (off) by default and should only be enabled if directed to do so by your organization's logging policy.

Note: `log_statement_stats` **cannot** be enabled together with any of the per-module options.

Rationale:

The logging of these additional statistics when not mandated by your organization's logging policy greatly reduces the signal-to-noise ratio of the PostgreSQL logs.

Audit:

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

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

Remediation:

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

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

Default Value:

`off`

References:

1. http://man7.org/linux/man-pages/man2/getrusage.2.html

CIS Controls:

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

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

Profile Applicability:

- 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 finding.
Next, verify that desired auditing components are enabled:

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

If the output does not contain desired auditing components, this is a finding.
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:

The following instructions assume PostgreSQL 9.5 Community Packages are installed to the `/usr/pgsql-9.5/` directory and the standard contribution modules directory exists at `/usr/pgsql-9.5/share/contrib/`. To install and enable pgAudit, first we need to clone the GitHub repository and build the project in the appropriate directory:

```
$ cd /usr/pgsql-9.5/share/contrib/
$ git clone https://github.com/pgaudit/pgaudit.git
$ cd ./pgaudit
$ PATH=/usr/pgsql-9.5/bin:$PATH
$ make USE_PGXS=1 install
```

pgaudit is now built 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 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 following ddl and write operations
pgaudit.log='ddl,write'
```

Restart the PostgreSQL server for changes to take affect:

```
[root@localhost ~]# service postgresql-9.5 restart
Stopping postgresql-9.5 service:                           [  OK  ]
Starting postgresql-9.5 service:                           [  OK  ]
```

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; specifically, pgAudit v1.0.X is intended to support PostgreSQL 9.5. Please consult the pgAudit documentation for versioning details.

CIS Controls:

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

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 having more than one authorized individual administrating the PostgreSQL service. 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. Administrating 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 a user authorized to escalate privileges and test the `sudo` invocation by executing the following:

```
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:

```
sudo su - postgres
[sudo] password for user1:
postgres@localhost:~$
```

shows the 'user1' user is configured properly for sudo access.

Remediation:

As superuser `root`, execute the command `visudo` to edit the `/etc/sudoers` file so the following line is present:

```
%pg_wheel ALL= /bin/su - postgres
```

Additionally, all user accounts needing superuser access must be members of the group `pg_wheel`. You can check by executing something similar to the following example:

```
groups <username>
```

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
3. http://man7.org/linux/man-pages/man1/groups.1.html

CIS Controls:

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.

4.2 Ensure valid public keys are installed (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Valid SSH public/private key pairs should be installed.

Rationale:

The most secure mechanism for management is to log in locally into the UNIX account that controls and maintains the server's environment with an SSH key and use the Command Line Interface (CLI) `psql`. SSH keys have other advantages too; it is simple to add and remove user authorization, it eliminates the redundant typing of passwords, and it enables administrating large number of servers from a centralized host using simple CLI scripts.

Audit:

Assuming one has previously created an SSH key pair, you must confirm that the public key is already installed in the remote host's `$HOME/.ssh/authorized_keys` file.

It is understood that the SSH server on the remote host has been installed and configured to accept connections for the `postgres` UNIX account.

A successful login to the remote host's `postgres` UNIX account via SSH keys should return a shell prompt *without* prompting for a password. Here is an example login attempt using the SSH cli in a terminal executed from your workstation:

```
$ hostname -s
dbmaster
$ ssh postgres@<remote host> hostname -s
dbslave
```

Failure to login (as indicated by `Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password),ssh: connect to host <host> port 22: Connection refused`, or similar messages) may be explained by issues other than one's public key not being present on the remote host. Suffice to say that it is beyond the scope of this documentation to debug Failures. There is further, comprehensive documentation available online.

Remediation:

For demonstration purposes, the following example highlights the various issues one must consider and is just one of many methods that can be used to install and use a public SSH

key. It is recommended that a configuration management tool, such as Puppet, be used as part of a larger, and automated, provisioning process where there are many DBAs authorized to administrate multiple servers.

After creating your SSH public/private key pair, login as `root` on the PostgreSQL server and assign a temporary password to the `postgres` user account.

Copy the SSH public key from your key pair to the PostgreSQL server. This step will prompt you for the temporary password you set above:

```
ssh -copy -id -i $HOME/.ssh/id_rsa.pub postgres@<remote host>
```

Now that your SSH public key is in place, test that you can SSH to the PostgreSQL server **without being prompted for a password**:

```
ssh postgres@<remote host>
```

Upon successful login without being prompted for a password, one can now lock the `postgres` account to prevent future logins via password:

```
passwd -l postgres
```

New public keys can be added by editing the `postgres` account's authorization file directly:

```
ssh postgres@<remote host>
vim $HOME/.ssh/authorized_keys
```

Alternatively, another implementation would be to add the public key to one's own personal account on the remote host and then `sudo` into `postgres`. Added security is implied because you would need to supply your account password:

```
ssh -copy -id -i $HOME/.ssh/id_rsa.pub <remote host>
ssh <remote host>
sudo su - postgres
```

References:

1. http://man7.org/linux/man-pages/man1/ssh.1.html
2. http://man7.org/linux/man-pages/man1/ssh-keygen.1.html
3. https://linux.die.net/man/1/ssh-copy-id
4. http://man7.org/linux/man-pages/man1/passwd.1.html
5. http://man7.org/linux/man-pages/man1/scp.1.html
6. https://linux.die.net/man/1/rsync
7. https://puppet.com/

Notes:

Due to the complexity of SSH, this recommendation must be considered only as a starting point.

CIS Controls:

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.

4.3 Ensure excessive administrative privileges are revoked (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

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 for a superuser account. 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)

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:

```
$ 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 finding.

```
$ 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.

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

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`:

```
$ 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:

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

References:

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

CIS Controls:

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.

4.4 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:

```
$ 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 finding.

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

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 finding.

Remediation:

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

```
$ 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:

```
REVOKE EXECUTE ON FUNCTION delete_customer(integer,boolean) FROM appreader;
REVOKE
```

Confirm that the `appreader` user may no longer execute the function:

```
SELECT proname, proacl FROM pg_proc WHERE proname = 'delete_customer';
     proname     |                          proacl
-----------------+-------------------------------------------------------
 delete_customer | {=X/postgres,postgres=X/postgres,appwriter=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/current/static/catalog-pg-proc.html
2. https://www.postgresql.org/docs/current/static/sql-grant.html
3. https://www.postgresql.org/docs/current/static/sql-revoke.html

CIS Controls:

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.

4.5 Ensure excessive DML privileges are revoked (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

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 define 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 in a multi-tenant instance. 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=# \du+ *

postgres=# -- display all schema.tables created in current database
postgres=# \dt+ *.*

postgres=# -- query all tables and user grants in current database
postgres=# -- system catalogs information_schema and pg_catalog 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');
```

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, 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 finding. 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 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 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/current/static/sql-grant.html
2. https://www.postgresql.org/docs/current/static/sql-revoke.html
3. https://www.postgresql.org/docs/current/static/functions-info.html#functions-info-access-table
4. https://wiki.postgresql.org/wiki/A_Guide_to_CVE-2018-1058:_Protect_Your_Search_Path

CIS Controls:

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.

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

Profile Applicability:

- Level 1 - PostgreSQL on Linux

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;
```

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` inadvertantly, 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/current/static/ddl-rowsecurity.html
2. https://www.postgresql.org/docs/current/static/sql-alterrole.html

CIS Controls:

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.

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 Postgres host-based authentication file, `pg_hba.conf`, from top to bottom. The first rule conforming to the condition of the request executes the METHOD. 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 SECTION 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 Postgres, 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 administrating the Postgres server. Connecting with the `psql` CLI, via UNIX DOMAIN SOCKETS, using the peer 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:

```
$ whoami
postgres
$ psql postgres
psql (9.5.10)
Type "help" for help.

postgres=#
```

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

```
$ su - <user1>
$ whoami
user1
$ psql -U postgres postgres
psql: FATAL:  Peer authentication failed for user "postgres"
$ exit
$ su - <user2>
$ whoami
user2
$ psql -U postgres postgres
psql: FATAL: Peer authentication failed for user "postgres"
$ psql -U user1 postgres
psql: FATAL: Peer authentication failed for user "user1"
```

This test demonstrates the rule permitting connections when the database ROLE matches the UNIX account.

Remediation:

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

```
$ psql -c "create role user1 with login password 'mypassword';"
CREATE ROLE
```

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

```
$ su - user1
$ psql postgres
psql (9.5.10)
Type "help" for help.

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 Postgres logs will report the outcome of the SIGHUP:

```
[root@localhost ~]# service postgresql-9.5 reload
```

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

```
# allow postgres user logins
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             postgres                                peer
```

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

```
# allow all local users only if they are connecting to a db named the same
as their username
# TYPE  DATABASE        USER                                    METHOD
local   samerole        all                                     peer
```

```
# allow only local users 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/current/static/client-authentication.html
2. https://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html

CIS Controls:

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.

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`
- `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 either in both encrypted and unencrypted sessions.

Use of the `gss`, `sspi`, `pam`, `ldap`, `radius`, and `cert` METHODs are dependent upon the availability of external authenticating processes/services.

Rationale:

Audit:

Newly created data clusters are empty of data and has 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 <my 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=<my host> user=postgres sslmode=disable'
Password:
```

Test encrypted sessions:

```
$ psql 'host=<my host> 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               md5
 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
# TYPE    DATABASE        USER            ADDRESS                 METHOD
host      all             postgres        127.0.0.1/32            md5

# allow users to connect remotely only to the database named after them:
# TYPE    DATABASE        USER            ADDRESS                 METHOD
host      samerole        all             0.0.0.0/0               md5

# allow only those users who are a member of the 'rw' role to connect
# only to the database named after them:
```

```
# TYPE    DATABASE        USER            ADDRESS                 METHOD
host      samerole        +rw             0.0.0.0/0               md5
```

References:

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

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 more.

CIS Controls:

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.

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 too. 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 on Linux

Description:

Understanding the vulnerability of postgres 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 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 postgres 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.

CIS Controls:

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.

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

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

In order to serve multiple clients efficiently, the PostgreSQL server launches a new "backend" process for each client. A new child process is created immediately after an incoming connection is detected. The runtime parameters in this benchmark 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.

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, unit from pg_settings where context like
'%backend%' order by 1;
         name          | setting | unit
-----------------------+---------+------
 ignore_system_indexes | off     |
 log_connections       | on      |
 log_disconnections    | on      |
 post_auth_delay       | 0       | s
(4 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, i.e., during a maintenance window.

Remediation:

Once detected, the unauthorized/undesired change can be made by corrected 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.

1. Query the view `pg_settings` and compare with previous query outputs for any changes.
2. Review configuration files `postgresql.conf`, `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, i.e. **`ps aux | grep postgres`** or **`ps aux | grep postmaster`**.

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/current/static/view-pg-settings.html

CIS Controls:

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.

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

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

PostgreSQL runtime parameters that are executed by the postmaster process.

Rationale:

The postmaster, or **postgres**, 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 i.e. typically UNIX user account `postgres`. Therefore, all exploits require the successful compromise of either the 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/9.6/data/postgresql.conf
 data_directory                      | /var/lib/pgsql/9.6/data
 dynamic_shared_memory_type          | posix
 event_source                        | PostgreSQL
 external_pid_file                   |
 hba_file                            | /var/lib/pgsql/9.6/data/pg_hba.conf
 hot_standby                         | off
 huge_pages                          | try
 ident_file                          | /var/lib/pgsql/9.6/data/pg_ident.conf
 listen_addresses                    | localhost
```

```
 logging_collector                   | on
 max_connections                     | 100
 max_files_per_process               | 1000
 max_locks_per_transaction           | 64
 max_pred_locks_per_transaction      | 64
 max_prepared_transactions           | 0
 max_replication_slots               | 0
 max_wal_senders                     | 0
 max_worker_processes                | 8
 old_snapshot_threshold              | -1
 port                                | 5432
 shared_buffers                      | 16384
 shared_preload_libraries            |
 ssl                                 | off
 ssl_ca_file                         |
 ssl_cert_file                       | server.crt
 ssl_ciphers                         | HIGH:MEDIUM:+3DES:!aNULL
 ssl_crl_file                        |
 ssl_ecdh_curve                      | prime256v1
 ssl_key_file                        | server.key
 ssl_prefer_server_ciphers           | on
 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                           | minimal
 wal_log_hints                       | off
(48 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 configuration files `postgresql.conf`, `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, i.e. `ps aux | grep postgres` or `ps aux | grep postmaster`

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, thus depriving other connections of much needed resources. Data can be corrupted by allowing damaged pages to load or by changing parameters to reinterpret values in an unexpected fashion as for example 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/current/static/view-pg-settings.html

CIS Controls:

18 Application Software Security
Application Software Security

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

Profile Applicability:

- 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 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 postgres server process and cluster i.e. typically UNIX user account `postgres`.

```
postgres=# select name, setting from pg_settings
where context = 'sighup' order by 1;
              name               |                   setting

---------------------------------+-----------------------------------------
----
--
 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    | 20
 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                 | on
 log_destination                 | stderr
 log_directory                   | pg_log
 log_file_mode                   | 0600
 log_filename                    | postgresql-%a.log
 log_hostname                    | off
 log_line_prefix                 | %t [%p]: [%l-1]
db=%d,user=%u,app=%a,client=%
h
 log_rotation_age                | 1440
 log_rotation_size               | 0
 log_timezone                    | UTC
 log_truncate_on_rotation        | on
 max_standby_archive_delay       | 30000
 max_standby_streaming_delay     | 30000
 max_wal_size                    | 64
 min_wal_size                    | 5
 pre_auth_delay                  | 0
 restart_after_crash             | 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_sender_timeout              | 60000
 wal_sync_method                 | fdatasync
 wal_writer_delay                | 200
 wal_writer_flush_after          | 128
(60 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, thus depriving other connections of much needed resources. Data can be corrupted by allowing damaged pages to load or by changing parameters to reinterpret values in an unexpected fashion as for example 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/current/static/view-pg-settings.html

CIS Controls:

18 Application Software Security
Application Software Security

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

Profile Applicability:

- 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 the 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
 lc_messages                 | en_US.UTF-8
 lo_compat_privileges        | off
 log_duration                | on
 log_error_verbosity         | default
 log_executor_stats          | off
 log_lock_waits              | on
 log_min_duration_statement  | 0
 log_min_error_statement     | error
 log_min_messages            | warning
 log_parser_stats            | off
 log_planner_stats           | off
 log_replication_commands    | off
 log_statement               | ddl
 log_statement_stats         | off
 log_temp_files              | 0
```

```
 max_stack_depth            | 2048
 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
 zero_damaged_pages         | off
(30 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 configuration files `postgreql.conf`, `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, i.e. **`ps aux | grep postgres`** or **`ps aux | grep postmaster`**.

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, thus depriving other connections of much needed resources. Data can be corrupted by allowing damaged pages to load or by changing parameters to reinterpret values in an unexpected fashion as for example 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/current/static/view-pg-settings.html

CIS Controls:

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.

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

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

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

Rationale:

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

Audit:

There are two methods that can be used to analyze the state of ROLE runtime parameters and to determine if they have been compromised:

PostgreSQL logging

A cursory review of the logging session instructions that are executed.

ROLE/ENTITY attributes

Inspect all catalogs and list attributes for database entities such as ROLE and database.

```
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_tablespace                  |
default_text_search_config          | pg_catalog.english
default_transaction_deferrable      | off
default_transaction_isolation       | read committed
default_transaction_read_only       | off
default_with_oids                   | off
effective_cache_size                | 524288
effective_io_concurrency            | 1
enable_bitmapscan                   | on
enable_hashagg                      | on
enable_hashjoin                     | on
enable_indexonlyscan                | on
enable_indexscan                    | on
enable_material                     | on
enable_mergejoin                    | on
enable_nestloop                     | on
enable_seqscan                      | on
enable_sort                         | on
enable_tidscan                      | on
escape_string_warning               | on
exit_on_error                       | off
extra_float_digits                  | 0
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
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_workers_per_gather     | 0
min_parallel_relation_size          | 1024
operator_precedence_warning         | off
parallel_setup_cost                 | 1000
parallel_tuple_cost                 | 0.1
password_encryption                 | on
quote_all_identifiers               | off
random_page_cost                    | 4
replacement_sort_tuples             | 150000
row_security                        | on
search_path                         | "$user", public
seq_page_cost                       | 1
sql_inheritance                     | on
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
 temp_buffers                       | 1024
 temp_tablespaces                   |
 TimeZone                           | US/Eastern
 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_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
 work_mem                           | 4096
 xmlbinary                          | base64
 xmloption                          | content
(104 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, thus depriving other connections of much needed resources. 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/current/static/view-pg-settings.html

CIS Controls:

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.

6.7 Ensure SSL is enabled and configured correctly (Scored)

Profile Applicability:

- 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 finding.

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

postgres=# select name, setting, source
           from pg_settings where name = 'ssl';
name | setting |       source
-----+---------+--------------------
ssl  | off     | configuration file
(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.

```
# create new certificate and enter details at prompts
$ openssl req -new -text -out server.req

# remove passphrase (required for automatic server start up)
$ openssl rsa -in privkey.pem -out server.key && rm privkey.pem

# 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 = on
```

Finally, restart PostgreSQL and confirm `ssl` using commands outlined in Audit Procedures. For this example, all commands executed as `postgres` account with appropriate `sudo` privileges granted.

```
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.

References:

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

CIS Controls:

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.

6.8 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 undermines the purposes of utilizing encryption to protect data. Postgres 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:

```
$ cat /proc/sys/crypto/fips_enabled
1
```

   If `fips_enabled` is not `1`, then the system is not FIPS enabled.

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

```
$ openssl version
OpenSSL 1.0.1e-fips 11 Feb 2013
```

   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:

- Disable prelinking

```
$ echo PRELINKING=no > /etc/sysconfig/prelink
```

- Undo any prelinking on any system files

```
$ prelink -u -a
```

- Install the `dracut-fips` package

```
$ yum -y install dracut-fips
```

- Recreate the `initramfs` file

```
$ dracut -f
```

- Modify the kernel command line of the current kernel in the `/boot/grub/grub.conf` file by adding the following option: `fips=1`
- Reboot the system for changes to take effect.
- Verify `fips_enabled` according to Audit Procedure above.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/sect-Security_Guide-Federal_Standards_And_Regulations-Federal_Information_Processing_Standard.html
2. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1758.pdf
3. http://csrc.nist.gov/publications/PubsFIPS.html

CIS Controls:

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.

6.9 Ensure the pgcrypto extension is installed and configured correctly (Not Scored)

Profile Applicability:

- 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 requiring "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).

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  | 1.2             |                   | cryptographic functions
(1 row)
```

If data in the database requires encryption and pgcrypto is not available, this is a finding. 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 finding. If controls do not exist or are not enabled, this is a finding.

Remediation:

The `pgcrypto` extension is included with the PostgreSQL 'contrib' package. Although included, it needs to be created in the database. This installation assumes that the database has been initialized and the 'contrib' package is installed.
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 | 1.2             | 1.2               | cryptographic functions
(1 row)
```

References:

1. http://www.postgresql.org/docs/current/static/pgcrypto.html

CIS Controls:

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.

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 in 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. Similarly, to standard client connections, it also honors the same authentication rules as expressed in the PostgreSQL host-based authentication file, `pg_hba.conf`. Although not required, streaming replication also has the ability to take advantage of WAL archiving making it an extremely flexible solution for diverse and complex configurations and operating conditions.

7.1 Ensure SSL Certificates are Configured For Replication (Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Creating and managing SSL certificates on the PRIMARY and STANDBY host(s).

Rationale:

Secure Sockets Layer (SSL) certificates enable encrypted communications between the PRIMARY and STANDBY hosts. SSL certificates can also be used to authenticate the identity of the host. The use of SSL certificates mitigates against sniffing of what would otherwise be sensitive information that's being transmitted in the clear.

Audit:

Encrypted sessions require the following sets of conditions:

- Both the server certificate and private key exist.
- The certificate and key are located as per the location set in the configuration file `postgresql.conf`.
- The runtime parameter `ssl` is marked as `on`.

In a client session:

- Confirm the default location of where you should place SSL certificates.

```
 postgres=# show ssl_cert_file;
  ssl_cert_file
 ---------------
  server.crt
 (1 row)

 postgres=# show ssl_key_file;
  ssl_key_file
 --------------
  server.key
 (1 row)
```

- Confirm state of parameter, `ssl`:

```
postgres=# show ssl;
 ssl
-----
```

```
 on
(1 row)
```

Note: One can choose the names of both the server certificate and private key but they must be correctly identified in the configuration file `postgresql.conf`.

Remediation:

Running a server with `ssl=on` is not possible until both a server certificate and key have been created, installed in the correct location, and are set with the correct permissions. Although generating certificates signed by a Certificate Authority, CA is ideal, one can use self-signed certificates too.

Use the following example as a starting point to generate a self-signed certificate, the script is executed on the server in question. Note that the value of the `SUBJ` variable contains a carriage return to allow for formatting - the entire value should appear on a single line in the script.

```
#!/bin/bash

set -e

state='Washington'
city='Seattle'
organization='My Company'
org_unit='My department'
cn=$(hostname -f)
email='you@company.com'

SUBJ="/C=US/ST=$state/L=$city/O=$organization/
      OU=$org_unit/CN=$cn/emailAddress=$email"

# Expire in 10 yrs
DAYS=3650

if [ -e "$PGDATA" ]; then
  KEY="$PGDATA/server.key"
  CRT="$PGDATA/server.crt"
else
  KEY="server.key"
  CRT="server.crt"
fi

openssl  req \
        -nodes \
        -x509 \
        -newkey rsa:2048 \
        -keyout $KEY \
        -out $CRT \
        -days $DAYS \
        -subj "$SUBJ"
```

```
chmod 600 $KEY
chmod 664 $CRT

echo "DONE"
```

References:

1. https://www.postgresql.org/docs/current/static/runtime-config-connection.html#RUNTIME-CONFIG-CONNECTION-SECURITY
2. https://linux.die.net/man/1/openssl

CIS Controls:

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.

7.2 Ensure base backups are configured and functional (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

A base backup is a copy of the PRIMARY host's data cluster and is used to create STANDBY hosts and Point In Time Recovery mechanisms. Base backups should be copied across networks in a secure manner using an encrypted transport mechanism. CLI examples includes `scp`, `sftp` and `rsync -e ssh`. Alternatively, the CLI `cp` can be used with an SSL-enabled implementation of an NFS mount point, or the PostgreSQL CLI `pg_basebackup` can be used. However, SSL encryption should be enabled on the server. Beware it is possible to use this utility without SSL encryption enabled.

Rationale:

Audit:

Remediation:

There are two methods of creating base backups; manual and simple. The "manual" method explicitly first signals that a backup is about to start then copy both the data cluster and WALs, using the appropriate tools/utilities, the last step involves once again signaling the PRIMARY host the completion of the copy process.

```
postgres=# SELECT pg_start_backup('my base backup');
 pg_start_backup
-----------------
 0/2000028
(1 row)
```

```
  # copy both the data cluster and
  # all WALs generated during the process
  # ex:
  $ scp -rp $PGDATA user@dest:/path
```

```
postgres=# SELECT pg_stop_backup();
 pg_start_backup
-----------------
 0/2000030
(1 row)
```

Executing base backups using `pg_basebackup` requires the following additional steps:

- A replication ROLE is created

```
postgres=# create role replicant with login replication password
'mypassword';
CREATEW ROLE
```

- Validate these runtime parameters in `postgresql.conf`
  - `ssl = on`
  - `ssl_cert_file = 'server.crt" # permissions 664`
  - `ssl_key_file = 'server.key' # permissions 600`
- Add a replication entry to `pg_hba.conf` (adjusting `0.0.0.0/0` to match your CIDR)

```
hostssl replication     replicant       0.0.0.0/0               md5
```

- Test the connection with an invocation similar to the following (notice the use of `sslmode`)

```
$ psql 'host=mySrcHost dbname=postgres user=replicant sslmode=require'
```

- Execute a command similar to the following from the target host:

```
export PGPASSWORD='mypassword'

pg_basebackup -h mySrcHost -p 5432 -U replicant \
                -D /opt/data/postgres/data \
                -P -v -R --xlog-method=stream \
                > pg_basebackup.log 2>&1
```

References:

1. https://www.postgresql.org/docs/current/static/functions-admin.html#FUNCTIONS-ADMIN-BACKUP-TABLE
2. https://www.postgresql.org/docs/current/static/app-pgbasebackup.html

CIS Controls:

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.

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, i.e. `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 standard.

References:

1. https://www.postgresql.org/docs/current/static/runtime-config-wal.html#RUNTIME-CONFIG-WAL-ARCHIVING
2. https://linux.die.net/man/1/ssh-keygen

CIS Controls:

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.

7.4 Ensure streaming replication parameters are configured correctly (Not Scored)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Streaming replication from a PRIMARY host transmits sensitive passwords, DDL, and DML activities as well as other sensitive 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, as for example this `psql` meta-command is used to list all ROLES:

```
postgres=# \du+
                                   List of roles
 Role name  |                 Attributes          | Member of | Description
------------+-------------------------------------+-----------+-------------
 postgres   | Superuser, Create role, Create DB, | {}        |
            | Replication, Bypass RLS             |           |
replicator  | Replication                         | {}        |
```

On the target/STANDBY host, execute a`psql` invocation similar to the following, confirming that SSL communications are possible:

```
$ psql 'host=mySrcHost dbname=postgres user=replicant password=mypassword
sslmode=require' -c 'select 1'
```

Remediation:

Review these benchmarks:

- "ABOUT BASE-BACKUPS"
- "ABOUT SSL CERTIFICATES"
- "ABOUT WAL ARCHIVING"

Create a role for replication, for example:

```
postgres=# create role replicant with login replication password
'mypassword';
CREATE ROLE
```

Confirm the file `recovery.conf` is present on the STANDBY host and contains lines similar to the following:

```
standby_mode=on
primary_conninfo = 'user=replicant password=mypassword host=mySrcHost
port=5432 sslmode=require sslcompression=1'
```

References:

1. https://www.postgresql.org/docs/current/static/runtime-config-connection.html#RUNTIME-CONFIG-CONNECTION-SECURITY
2. https://www.postgresql.org/docs/current/static/functions-admin.html#FUNCTIONS-ADMIN-BACKUP-TABLE
3. https://www.postgresql.org/docs/current/static/app-pgbasebackup.html
4. https://www.postgresql.org/docs/current/static/runtime-config-wal.html#RUNTIME-CONFIG-WAL-ARCHIVING
5. https://linux.die.net/man/1/openssl

CIS Controls:

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.

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 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 in a much large network environment 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. Alternatively, inspect the parameter settings in `postgresql.conf` configuration file.

```
postgres=# SELECT name, setting
postgres-# FROM pg_settings
postgres-# WHERE name IN ('hba_file'
postgres(#               ,'ident_file'
postgres(#               ,'ssl_cert_file'
postgres(#               ,'ssl_key_file'
postgres(#               ,'ssl_ca_file'
postgres(#               ,'ssl_crl_file'
postgres(#               ,'krb_server_keyfile'
```

```
postgres(#               );
        name         |                setting
---------------------+--------------------------------------
 hba_file            | /var/lib/pgsql/9.5/data/pg_hba.conf
 ident_file          | /var/lib/pgsql/9.5/data/pg_ident.conf
 krb_server_keyfile | FILE:/etc/sysconfig/pgsql/krb5.keytab
 ssl_ca_file         |
 ssl_cert_file       | server.crt
 ssl_crl_file        |
 ssl_key_file        | server.key
(7 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 finding.

Remediation:

Follow these steps to remediate the configuration file locations and permissions:

1. 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.
2. Ensure their file permissions are restricted as much as possible, i.e. only superuser read access.
3. Change the settings accordingly in the `postgresql.conf` configuration file.
4. Restart the database cluster for the changes to take effect.

Default Value:

The defaults for PostgreSQL configuration files are listed below.

```
#hba_file            = 'ConfigDir/pg_hba.conf'
#ident_file          = 'ConfigDir/pg_ident.conf'
#ssl_cert_file       = 'server.crt'
#ssl_key_file        = 'server.key'
#ssl_ca_file         = ''
#ssl_crl_file        = ''
#krb_server_keyfile = ''
#include_dir         = 'conf.d'
#include_if_exists   = 'exists.conf'
#include             = 'special.conf'
```

References:

1. https://www.postgresql.org/docs/current/static/runtime-config-file-locations.html
2. https://www.postgresql.org/docs/10/static/runtime-config-connection.html
3. https://www.postgresql.org/docs/10/static/config-setting.html#CONFIG-INCLUDES

CIS Controls:

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.

8.2 Ensure PostgreSQL subdirectory locations are outside the data cluster (Not Scored)

Profile Applicability:

- 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
postgres-# FROM pg_settings
postgres-# WHERE name IN ('data_directory'
postgres(#                ,'log_directory'
postgres(#                ,'default_tablespace'
postgres(#                ,'temp_tablespaces'
postgres(#                );
        name         |         setting
---------------------+-------------------------
 data_directory      | /var/lib/pgsql/9.5/data
 default_tablespace |
 log_directory       | pg_log
 temp_tablespaces    |
(4 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 finding.

Remediation:

Perform the following steps to remediate the subdirectory locations and permissions:

1. 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.
2. Ensure file permissions are restricted as much as possible, i.e. only superuser read access.
3. When directories are relocated to other partitions, ensure that they are of sufficient size to mitigate against excessive space utilization.
4. 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 `pg_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/current/static/runtime-config-file-locations.html

CIS Controls:

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.

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 backup 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, and more. `pgBackRest` is written in 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 (9.5-1): 00000008000098E000000036 /
                               0000000800009953000000BB

  full backup: 20171028-080001F
```

```
    timestamp start/stop: 2017-10-28 08:00:01 / 2017-10-28 12:04:21
    wal start/stop: 000000080000098E000000036 / 000000080000098E100000076
    database size: 2218.1GB, backup size: 2218.1GB
    repository size: 417.3GB, repository backup size: 417.3GB

  full backup: 20171104-080001F
    timestamp start/stop: 2017-11-04 08:00:01 / 2017-11-04 12:20:00
    wal start/stop: 000000080000099340000000B9 / 000000080000099360000000B
    database size: 2225.6GB, backup size: 2225.6GB
    repository size: 418.8GB, repository backup size: 418.8GB
```

Audit:

`pgBackRest` is not installed nor configured for PostgreSQL by default, but instead is maintained as a GitHub project. To determine whether or not `pgBackRest` has been installed on your system, simply invoke the executable `pgbackrest` as the PostgreSQL software owner, shown here as `postgres`. If `pgBackRest` is installed, then usage displays as below - otherwise the `pgbackrest: command not found` error returns. `pgBackRest` typically installs to `/usr/bin/pgbackrest` so you could also check for the executable in that location or use OS commands `find`, `locate`, `which`, or something similar to determine if the product is installed.

```
$ sudo su - postgres
$ pgbackrest
pgBackRest 1.25 - 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-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` installation is rather simple and straightforward. All steps below must be performed as `postgres` with appropriate sudo privileges granted.

- Install required perl libraries

```
$ sudo yum -y install libdbd-pg-perl libio-socket-ssl-perl libxml-
libxml-perl
```

- Download the project from GitHub

```
$ sudo wget -q -O - \
   https://github.com/pgbackrest/pgbackrest/archive/release/1.25.tar.gz
| \
   sudo tar zx -C /root
```

- Install `pgBackRest`

```
$ sudo cp -r /root/pgbackrest-release-1.25/lib/pgBackRest \
   /usr/share/perl5
$ sudo find /usr/share/perl5/pgBackRest -type f -exec chmod 644 {} +
$ sudo find /usr/share/perl5/pgBackRest -type d -exec chmod 755 {} +
$ sudo cp /root/pgbackrest-release-1.25/bin/pgbackrest
/usr/bin/pgbackrest
$ sudo chmod 755 /usr/bin/pgbackrest
$ sudo mkdir -m 770 /var/log/pgbackrest
$ sudo chown postgres:postgres /var/log/pgbackrest
$ sudo touch /etc/pgbackrest.conf
$ sudo chmod 640 /etc/pgbackrest.conf
$ sudo chown postgres:postgres /etc/pgbackrest.conf
```

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. http://http://pgbackrest.org/
2. https://github.com/pgbackrest/pgbackrest
3. https://www.postgresql.org/docs/current/static/app-pgdump.html
4. https://www.postgresql.org/docs/current/static/app-pgbasebackup.html

CIS Controls:

10 Data Recovery Capability

Data Recovery Capability

8.4 Ensure miscellaneous configuration settings are correct (Not Scored)

Profile Applicability:

- 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
postgres-# FROM pg_settings
postgres-# WHERE name IN ('external_pid_file'
postgres(#               ,'unix_socket_directories'
postgres(#               ,'shared_preload_libraries'
postgres(#               ,'dynamic_library_path'
postgres(#               ,'local_preload_libraries'
postgres(#               ,'session_preload_libraries'
postgres(#               );
           name             |          setting
----------------------------+---------------------------
 dynamic_library_path       | $libdir
 external_pid_file          |
 local_preload_libraries    |
 session_preload_libraries  |
 shared_preload_libraries   |
 unix_socket_directories    | /var/run/postgresql, /tmp
(6 rows)
```

Inspect the file and directory permissions for all returned values. Only superusers shouldhave access control rights for these files and directories. If permissions are not highly restricted, this is a finding.

Remediation:

Follow these steps to remediate the configuration:

1. Determine permissions based on your organization's security policies.
2. Relocate all files and ensure their permissions are restricted as much as possible, i.e. only superuser read access.
3. Ensure all directories where these files are located have restricted permissions such that the superuser can read but not write.
4. 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 `/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/current/static/runtime-config-file-locations.html
2. https://www.postgresql.org/docs/10/static/runtime-config-connection.html
3. https://www.postgresql.org/docs/10/static/runtime-config-client.html

CIS Controls:

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.

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 Community Packages (Not Scored)	☐	☐
1.3	Ensure Installation of Binary Packages (Not Scored)	☐	☐
1.4	Ensure Service Runlevel Is Registered And Set Correctly (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)	☐	☐
2.3	Ensure permissions for PostgreSQL binaries are set correctly (Scored)	☐	☐
3	**Logging Monitoring And Auditing (Centos 6)**		
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 sent to the database client (Not Scored)	☐	☐
3.1.13	Ensure the correct messages are written to the server log (Not Scored)	☐	☐
3.1.14	Ensure the correct SQL statements generating errors are recorded (Not Scored)	☐	☐

3.1.15	Ensure 'log_min_duration_statement' is disabled (Scored)	☐	☐
3.1.16	Ensure 'debug_print_parse' is disabled (Scored)	☐	☐
3.1.17	Ensure 'debug_print_rewritten' is disabled (Scored)	☐	☐
3.1.18	Ensure 'debug_print_plan' is disabled (Scored)	☐	☐
3.1.19	Ensure 'debug_pretty_print' is enabled (Scored)	☐	☐
3.1.20	Ensure 'log_checkpoints' is enabled (Scored)	☐	☐
3.1.21	Ensure 'log_connections' is enabled (Scored)	☐	☐
3.1.22	Ensure 'log_disconnections' is enabled (Scored)	☐	☐
3.1.23	Ensure 'log_duration' is enabled (Scored)	☐	☐
3.1.24	Ensure 'log_error_verbosity' is set correctly (Not Scored)	☐	☐
3.1.25	Ensure 'log_hostname' is set correctly (Scored)	☐	☐
3.1.26	Ensure 'log_line_prefix' is set correctly (Not Scored)	☐	☐
3.1.27	Ensure 'log_lock_waits' is enabled (Scored)	☐	☐
3.1.28	Ensure 'log_statement' is set correctly (Scored)	☐	☐
3.1.29	Ensure all temporary files are logged (Scored)	☐	☐
3.1.30	Ensure 'log_timezone' is set correctly (Scored)	☐	☐
3.1.31	Ensure 'log_parser_stats' is disabled (Scored)	☐	☐
3.1.32	Ensure 'log_planner_stats' is disabled (Scored)	☐	☐
3.1.33	Ensure 'log_executor_stats' is disabled (Scored)	☐	☐
3.1.34	Ensure 'log_statement_stats' is disabled (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 valid public keys are installed (Scored)	☐	☐
4.3	Ensure excessive administrative privileges are revoked (Scored)	☐	☐
4.4	Ensure excessive function privileges are revoked (Scored)	☐	☐
4.5	Ensure excessive DML privileges are revoked (Scored)	☐	☐
4.6	Ensure Row Level Security (RLS) is configured correctly (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 SSL is enabled and configured correctly (Scored)	☐	☐
6.8	Ensure FIPS 140-2 OpenSSL Cryptography Is Used (Scored)	☐	☐
6.9	Ensure the pgcrypto extension is installed and configured correctly (Not Scored)	☐	☐
7	**Replication**		
7.1	Ensure SSL Certificates are Configured For Replication (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