

CIS PostgreSQL 17 Benchmark

v1.0.0 - 01-27-2025

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/>

For information on referencing and/or citing CIS Benchmarks in 3rd party documentation (including using portions of Benchmark Recommendations) please contact CIS Legal (CISLegal@cisecurity.org) and request guidance on copyright usage.

NOTE: It is **NEVER** acceptable to host a CIS Benchmark in **ANY** format (PDF, etc.) on a 3rd party (non-CIS owned) site.

Table of Contents

Terms of Use	1
Table of Contents	2
Overview	5
Important Usage Information	5
Key Stakeholders.....	5
Apply the Correct Version of a Benchmark	6
Exceptions.....	6
Remediation	7
Summary.....	7
Target Technology Details	8
Intended Audience	8
Consensus Guidance.....	9
Typographical Conventions.....	10
Recommendation Definitions	11
Title.....	11
Assessment Status	11
Automated	11
Manual.....	11
Profile.....	11
Description	11
Rationale Statement.....	11
Impact Statement.....	12
Audit Procedure.....	12
Remediation Procedure.....	12
Default Value.....	12
References	12
CIS Critical Security Controls® (CIS Controls®).....	12
Additional Information	12
Profile Definitions.....	13
Acknowledgements.....	14
Recommendations	15
1 Installation and Patches	15
1.1 Ensure packages are obtained from authorized repositories (Manual)	16
1.2 Install only required packages (Manual)	20
1.3 Ensure systemd Service Files Are Enabled (Automated)	22
1.4 Ensure Data Cluster Initialized Successfully (Automated).....	24

1.5 Ensure the Latest Security Patches are Applied (Manual)	26
1.6 Verify That 'PGPASSWORD' is Not Set in Users' Profiles (Automated).....	28
1.7 Verify That the 'PGPASSWORD' Environment Variable is Not in Use (Automated)	30
2 Directory and File Permissions	32
2.1 Ensure the file permissions mask is correct (Manual)	33
2.2 Ensure extension directory has appropriate ownership and permissions (Automated)	35
2.3 Disable PostgreSQL Command History (Automated)	37
2.4 Ensure Passwords are Not Stored in the service file (Manual)	39
3 Logging And Auditing.....	41
3.1 PostgreSQL Logging.....	41
3.1.1 Logging Rationale.....	41
3.1.2 Ensure the log destinations are set correctly (Automated)	42
3.1.3 Ensure the logging collector is enabled (Automated)	44
3.1.4 Ensure the log file destination directory is set correctly (Automated)	46
3.1.5 Ensure the filename pattern for log files is set correctly (Automated)	48
3.1.6 Ensure the log file permissions are set correctly (Automated)	50
3.1.7 Ensure 'log_truncate_on_rotation' is enabled (Automated)	52
3.1.8 Ensure the maximum log file lifetime is set correctly (Automated)	54
3.1.9 Ensure the maximum log file size is set correctly (Automated)	56
3.1.10 Ensure the correct syslog facility is selected (Manual)	58
3.1.11 Ensure syslog messages are not suppressed (Automated)	60
3.1.12 Ensure syslog messages are not lost due to size (Automated)	62
3.1.13 Ensure the program name for PostgreSQL syslog messages are correct (Automated).....	64
3.1.14 Ensure the correct messages are written to the server log (Automated).....	66
3.1.15 Ensure the correct SQL statements generating errors are recorded (Automated)	68
3.1.16 Ensure 'debug_print_parse' is disabled (Automated)	70
3.1.17 Ensure 'debug_print_rewritten' is disabled (Automated).....	72
3.1.18 Ensure 'debug_print_plan' is disabled (Automated).....	74
3.1.19 Ensure 'debug_pretty_print' is enabled (Automated)	76
3.1.20 Ensure 'log_connections' is enabled (Automated)	78
3.1.21 Ensure 'log_disconnections' is enabled (Automated)	80
3.1.22 Ensure 'log_error_verbosity' is set correctly (Automated).....	82
3.1.23 Ensure 'log_hostname' is set correctly (Automated).....	84
3.1.24 Ensure 'log_line_prefix' is set correctly (Automated)	86
3.1.25 Ensure 'log_statement' is set correctly (Automated)	88
3.1.26 Ensure 'log_timezone' is set correctly (Automated)	90
3.2 Ensure the PostgreSQL Audit Extension (pgAudit) is enabled (Automated)	92
4 User Access and Authorization.....	95
4.1 Ensure Interactive Login is Disabled (Manual).....	96
4.2 Ensure sudo is configured correctly (Manual)	98
4.3 Ensure excessive administrative privileges are revoked (Manual)	100
4.4 Lock Out Accounts if Not Currently in Use (Manual)	103
4.5 Ensure excessive function privileges are revoked (Automated)	104
4.6 Ensure excessive DML privileges are revoked (Manual)	107
4.7 Ensure Row Level Security (RLS) is configured correctly (Manual)	112
4.8 Ensure the set_user extension is installed (Automated).....	117
4.9 Make use of predefined roles (Manual).....	123
5 Connection and Login	127
5.1 Do Not Specify Passwords in the Command Line (Manual)	128
5.2 Ensure PostgreSQL is Bound to an IP Address (Manual)	130
5.3 Ensure login via "local" UNIX Domain Socket is configured correctly (Manual)	132
5.4 Ensure login via "host" TCP/IP Socket is configured correctly (Manual)	135
5.5 Ensure per-account connection limits are used (Automated)	139

5.6 Ensure Password Complexity is configured (Manual)	140
6 PostgreSQL Settings	142
6.1 Understanding attack vectors and runtime parameters (Manual)	143
6.2 Ensure 'backend' runtime parameters are configured correctly (Automated)	145
6.3 Ensure 'Postmaster' Runtime Parameters are Configured (Manual)	147
6.4 Ensure 'SIGHUP' Runtime Parameters are Configured (Manual)	150
6.5 Ensure 'Superuser' Runtime Parameters are Configured (Manual)	154
6.6 Ensure 'User' Runtime Parameters are Configured (Manual)	157
6.7 Ensure FIPS 140-2 OpenSSL Cryptography Is Used (Automated)	161
6.8 Ensure TLS is enabled and configured correctly (Automated)	164
6.9 Ensure that TLSv1.3, or later, is configured (Automated)	168
6.10 Ensure Weak SSL/TLS Ciphers Are Disabled (Automated)	170
6.11 Ensure the pgcrypto extension is installed and configured correctly (Automated)	172
7 Replication	175
7.1 Ensure a replication-only user is created and used for streaming replication (Manual) ...	176
7.2 Ensure logging of replication commands is configured (Automated)	178
7.3 Ensure base backups are configured and functional (Manual)	180
7.4 Ensure WAL archiving is configured and functional (Automated)	182
7.5 Ensure streaming replication parameters are configured correctly (Manual)	184
8 Special Configuration Considerations	186
8.1 Ensure PostgreSQL subdirectory locations are outside the data cluster (Manual)	187
8.2 Ensure the backup and restore tool, 'pgBackRest', is installed and configured (Automated)	189
8.3 Ensure miscellaneous configuration settings are correct (Manual)	192
Appendix: Summary Table	194
Appendix: Change History	199

Overview

All CIS Benchmarks™ (Benchmarks) focus on technical configuration settings used to maintain and/or increase the security of the addressed technology, and they should be used in **conjunction** with other essential cyber hygiene tasks like:

- Monitoring the base operating system and applications for vulnerabilities and quickly updating with the latest security patches.
- End-point protection (Antivirus software, Endpoint Detection and Response (EDR), etc.).
- Logging and monitoring user and system activity.

In the end, the Benchmarks are designed to be a key **component** of a comprehensive cybersecurity program.

Important Usage Information

All Benchmarks are available free for non-commercial use from the [CIS Website](#). They can be used to manually assess and remediate systems and applications. In lieu of manual assessment and remediation, there are several tools available to assist with assessment:

- [CIS Configuration Assessment Tool \(CIS-CAT® Pro Assessor\)](#)
- [CIS Benchmarks™ Certified 3rd Party Tooling](#)

These tools make the hardening process much more scalable for large numbers of systems and applications.

NOTE: Some tooling focuses only on the Benchmark Recommendations that can be fully automated (skipping ones marked **Manual**). It is important that **ALL** Recommendations (**Automated** and **Manual**) be addressed since all are important for properly securing systems and are typically in scope for audits.

Key Stakeholders

Cybersecurity is a collaborative effort, and cross functional cooperation is imperative within an organization to discuss, test, and deploy Benchmarks in an effective and efficient way. The Benchmarks are developed to be best practice configuration guidelines applicable to a wide range of use cases. In some organizations, exceptions to specific Recommendations will be needed, and this team should work to prioritize the problematic Recommendations based on several factors like risk, time, cost, and labor. These exceptions should be properly categorized and documented for auditing purposes.

Apply the Correct Version of a Benchmark

Benchmarks are developed and tested for a specific set of products and versions and applying an incorrect Benchmark to a system can cause the resulting pass/fail score to be incorrect. This is due to the assessment of settings that do not apply to the target systems. To assure the correct Benchmark is being assessed:

- **Deploy the Benchmark applicable to the way settings are managed in the environment:** An example of this is the Microsoft Windows family of Benchmarks, which have separate Benchmarks for Group Policy, Intune, and Stand-alone systems based upon how system management is deployed. Applying the wrong Benchmark in this case will give invalid results.
- **Use the most recent version of a Benchmark:** This is true for all Benchmarks, but especially true for cloud technologies. Cloud technologies change frequently and using an older version of a Benchmark may have invalid methods for auditing and remediation.

Exceptions

The guidance items in the Benchmarks are called recommendations and not requirements, and exceptions to some of them are expected and acceptable. The Benchmarks strive to be a secure baseline, or starting point, for a specific technology, with known issues identified during Benchmark development are documented in the Impact section of each Recommendation. In addition, organizational, system specific requirements, or local site policy may require changes as well, or an exception to a Recommendation or group of Recommendations (e.g. A Benchmark could Recommend that a Web server not be installed on the system, but if a system's primary purpose is to function as a Webserver, there should be a documented exception to this Recommendation for that specific server).

In the end, exceptions to some Benchmark Recommendations are common and acceptable, and should be handled as follows:

- The reasons for the exception should be reviewed cross-functionally and be well documented for audit purposes.
- A plan should be developed for mitigating, or eliminating, the exception in the future, if applicable.
- If the organization decides to accept the risk of this exception (not work toward mitigation or elimination), this should be documented for audit purposes.

It is the responsibility of the organization to determine their overall security policy, and which settings are applicable to their unique needs based on the overall risk profile for the organization.

Remediation

CIS has developed [Build Kits](#) for many technologies to assist in the automation of hardening systems. Build Kits are designed to correspond to Benchmark's "Remediation" section, which provides the manual remediation steps necessary to make that Recommendation compliant to the Benchmark.

When remediating systems (changing configuration settings on deployed systems as per the Benchmark's Recommendations), please approach this with caution and test thoroughly.

The following is a reasonable remediation approach to follow:

- CIS Build Kits, or internally developed remediation methods should never be applied to production systems without proper testing.
- Proper testing consists of the following:
 - Understand the configuration (including installed applications) of the targeted systems. Various parts of the organization may need different configurations (e.g., software developers vs standard office workers).
 - Read the Impact section of the given Recommendation to help determine if there might be an issue with the targeted systems.
 - Test the configuration changes with representative lab system(s). If issues arise during testing, they can be resolved prior to deploying to any production systems.
 - When testing is complete, initially deploy to a small sub-set of production systems and monitor closely for issues. If there are issues, they can be resolved prior to deploying more broadly.
 - When the initial deployment above is completed successfully, iteratively deploy to additional systems and monitor closely for issues. Repeat this process until the full deployment is complete.

Summary

Using the Benchmarks Certified tools, working as a team with key stakeholders, being selective with exceptions, and being careful with remediation deployment, it is possible to harden large numbers of deployed systems in a cost effective, efficient, and safe manner.

NOTE: As previously stated, the PDF versions of the CIS Benchmarks™ are available for free, non-commercial use on the [CIS Website](#). All other formats of the CIS Benchmarks™ (MS Word, Excel, and [Build Kits](#)) are available for CIS [SecureSuite®](#) members.

CIS-CAT® Pro is also available to CIS [SecureSuite®](#) members.

Target Technology Details

This document, CIS PostgreSQL 17 Benchmark, provides prescriptive guidance for establishing a secure configuration posture for PostgreSQL 17. This guide was tested against PostgreSQL 17 running on RHEL 9, 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 17.

Consensus Guidance

This CIS Benchmark™ was created using a consensus review process comprised of a global community of subject matter experts. The process combines real world experience with data-based information to create technology specific guidance to assist users to secure their environments. 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
<code>Stylized Monospace font</code>	Used for blocks of code, command, and script examples. Text should be interpreted exactly as presented.
<code>Monospace font</code>	Used for inline code, commands, UI/Menu selections or examples. Text should be interpreted exactly as presented.
<Monospace font in brackets>	Text set in angle brackets denote a variable requiring substitution for a real value.
<i>Italic font</i>	Used to reference other relevant settings, CIS Benchmarks and/or Benchmark Communities. Also, used to denote the title of a book, article, or other publication.
Bold font	Additional information or caveats things like Notes , Warnings , or Cautions (usually just the word itself and the rest of the text normal).

Recommendation Definitions

The following defines the various components included in a CIS recommendation as applicable. If any of the components are not applicable it will be noted, or the component will not be included in the recommendation.

Title

Concise description for the recommendation's intended configuration.

Assessment Status

An assessment status is included for every recommendation. The assessment status indicates whether the given recommendation can be automated or requires manual steps to implement. Both statuses are equally important and are determined and supported as defined below:

Automated

Represents recommendations for which assessment of a technical control can be fully automated and validated to a pass/fail state. Recommendations will include the necessary information to implement automation.

Manual

Represents recommendations for which assessment of a technical control cannot be fully automated and requires all or some manual steps to validate that the configured state is set as expected. The expected state can vary depending on the environment.

Profile

A collection of recommendations for securing a technology or a supporting platform. Most benchmarks include at least a Level 1 and Level 2 Profile. Level 2 extends Level 1 recommendations and is not a standalone profile. The Profile Definitions section in the benchmark provides the definitions as they pertain to the recommendations included for the technology.

Description

Detailed information pertaining to the setting with which the recommendation is concerned. In some cases, the description will include the recommended value.

Rationale Statement

Detailed reasoning for the recommendation to provide the user a clear and concise understanding on the importance of the recommendation.

Impact Statement

Any security, functionality, or operational consequences that can result from following the recommendation.

Audit Procedure

Systematic instructions for determining if the target system complies with the recommendation.

Remediation Procedure

Systematic instructions for applying recommendations to the target system to bring it into compliance according to the recommendation.

Default Value

Default value for the given setting in this recommendation, if known. If not known, either not configured or not defined will be applied.

References

Additional documentation relative to the recommendation.

CIS Critical Security Controls® (CIS Controls®)

The mapping between a recommendation and the CIS Controls is organized by CIS Controls version, Safeguard, and Implementation Group (IG). The Benchmark in its entirety addresses the CIS Controls safeguards of (v7) "5.1 - Establish Secure Configurations" and (v8) "4.1 - Establish and Maintain a Secure Configuration Process" so individual recommendations will not be mapped to these safeguards.

Additional Information

Supplementary information that does not correspond to any other field but may be useful to the user.

Profile Definitions

The following configuration profiles are defined by this Benchmark:

- **Level 1 - PostgreSQL**

Items in this profile apply to PostgreSQL 17 and intend to:

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

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

- **Level 1 - PostgreSQL on Linux**

Items in this profile apply to PostgreSQL 17 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

Douglas Hunley

Contributor

Emad Al-Mousa

Ross Moles

Douglas Hunley

Patrick Stoeckle, Siemens AG

Editor

Tim Harrison, Center for Internet Security, New York

Recommendations

1 Installation and Patches

One of the best ways to ensure 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 (Manual)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

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. Package installation should include both the server and client packages. Contribution modules are optional depending upon one's architectural requirements (they are recommended though).

When obtaining and installing software packages (typically via **dnf** or **apt**), it's imperative that packages are sourced only from valid and authorized repositories. For PostgreSQL, the canonical repositories are the official PostgreSQL YUM repository (yum.postgresql.org) and the official PostgreSQL APT repository (apt.postgresql.org). Your chosen PostgreSQL vendor may offer its own software repositories as well.

Rationale:

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

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

From a security perspective, it's imperative to verify the PostgreSQL binary packages are sourced from a valid software repository. For a complete listing of all PostgreSQL binaries available via configured repositories inspect the output from **dnf provides '*libpq.so'** or **apt-file search /usr/pgsql-1*/lib/libpq.so.5**.

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 RHEL 9 install illustrating the use of the **dnf repolist all** command.

```
# whoami
root
# dnf repolist all | grep -E 'enabled$'
rhel-9-for-x86_64-appstream-rpms          Red Hat Enterprise enabled
rhel-9-for-x86_64-baseos-rpms           Red Hat Enterprise enabled
#
```

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

To inspect what versions of PostgreSQL packages are currently installed, we can query using the **rpm** commands. As illustrated below, no PostgreSQL packages are installed:

```
# whoami
root
# rpm -qa | grep postgres
#
```

If packages were returned in the above, we can determine from which repo they came by combining **dnf** and **rpm**:

```
# whoami
root
# dnf info $(rpm -qa|grep postgres) | grep -E '^Name|^Version|^From'
Name       : postgresql1*
Version    : 1*.0
From repo  : pgdg1*
Name       : postgresql1*-contrib
Version    : 1*.0
From repo  : pgdg1*
Name       : postgresql1*-libs
Version    : 1*.0
From repo  : pgdg1*
Name       : postgresql1*-server
Version    : 1*.0
From repo  : pgdg1*
```

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

Remediation:

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

As an example of adding an authorized repository, we will install the PGDG repository RPM from 'yum.postgresql.org':

```
# whoami
root
# dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-9-
x86_64/pgdg-redhat-repo-latest.noarch.rpm
Last metadata expiration check: 0:01:35 ago on Mon 03 Oct 2022 01:19:37 PM
EDT.
[snip]
Installed:
  pgdg-redhat-repo-42.0-46.noarch

Complete!
# whoami
root
# dnf repolist all | egrep 'enabled$'
pgdg-common                PostgreSQL enabled
pgdg12                     PostgreSQL enabled
pgdg13                     PostgreSQL enabled
pgdg14                     PostgreSQL enabled
pgdg15                     PostgreSQL enabled
pgdg16                     PostgreSQL enabled
pgdg17                     PostgreSQL enabled
rhel-9-for-x86_64-appstream-rpms  Red Hat E enabled
rhel-9-for-x86_64-baseos-rpms    Red Hat E enabled
```

If the version of PostgreSQL installed is not 1*.x or they did not come from a valid repository, the packages may be uninstalled using this command:

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







To install the PGDG RPMs for PostgreSQL 1*.x, run:

```
# whoami
root
# dnf install -y postgresql17-{server,contrib}
<snip>
Installed:
<snip>
  postgresql17-17.0-1PGDG.rhel9.x86_64      postgresql17-contrib-17.0-
1PGDG.rhel9.x86_64
  postgresql17-libs-17.0-1PGDG.rhel9.x86_64  postgresql17-server-17.0-
1PGDG.rhel9.x86_64
Complete!
```

References:

1. [https://en.wikipedia.org/wiki/DNF_\(software\)](https://en.wikipedia.org/wiki/DNF_(software))
2. [https://en.wikipedia.org/wiki/APT_\(software\)](https://en.wikipedia.org/wiki/APT_(software))
3. <https://yum.postgresql.org>
4. <https://apt.postgresql.org>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.1 <u>Establish and Maintain a Software Inventory</u> Establish and maintain a detailed inventory of all licensed software installed on enterprise assets. The software inventory must document the title, publisher, initial install/use date, and business purpose for each entry; where appropriate, include the Uniform Resource Locator (URL), app store(s), version(s), deployment mechanism, and decommission date. Review and update the software inventory bi-annually, or more frequently.			
v7	2.1 <u>Maintain Inventory of Authorized Software</u> Maintain an up-to-date list of all authorized software that is required in the enterprise for any business purpose on any business system.			

1.2 Install only required packages (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

Depending on the distribution, several other packages next to the mandatory `postgresql` might have been installed upon a system. Typical add-on packages are:

- `postgresql-doc`: PostgreSQL documentation.
- `phpPgAdmin`: PostgreSQL web-based administration tool.
- ...

Rationale:

Unused packages can increase the potential attack surface of the system.

Audit:

On Debian, one can use the following command to see a complete list of the available packages.

```
apt search postgresql
```

RHEL:

```
dnf search postgresql
```

Remediation:

Examine the installed packages:

Debian

```
dpkg -l $(apt-cache search postgresql --names-only | awk '{print $1}') 2>&1 |  
grep -v 'no packages found'
```

RHEL:

```
rpm -q $(dnf search postgresql | cut -d: -f1 | cut -d. -f1) 2>&1 | grep -Ev  
'package.*is not installed'
```

Remove any identified packages that are undesired:



Debian:

```
apt purge <pkg>
```

RHEL:

```
dnf erase <pkg>
```

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.8 Uninstall or Disable Unnecessary Services on Enterprise Assets and Software</u> Uninstall or disable unnecessary services on enterprise assets and software, such as an unused file sharing service, web application module, or service function.			

1.3 Ensure systemd Service Files Are Enabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

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

Rationale:

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

Audit:

Confirm the PostgreSQL service is enabled by executing the following:

```
$ whoami
root
$ systemctl is-enabled postgresql-17.service
disabled
```

If the intended PostgreSQL service is not registered as a dependency (or "want") of the default target (anything other than 'enabled' is returned), this is a failure.

Remediation:







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

```
# whoami
root
# systemctl enable postgresql-17
Created symlink /etc/systemd/system/multi-user.target.wants/postgresql-17.service → /usr/lib/systemd/system/postgresql-17.service.
# systemctl is-enabled postgresql-17.service
enabled
```

References:

1. <https://man7.org/linux/man-pages/man1/systemctl.1.html>
2. <https://www.freedesktop.org/software/systemd/man/systemd.special.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.1 Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>5.1 Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

1.4 Ensure Data Cluster Initialized Successfully (Automated)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

First-time installs of a given PostgreSQL major release require the instantiation of the database cluster. A database cluster is a collection of databases that are managed by a single server instance.

Rationale:

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

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

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

Audit:

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

```
# whoami
root
# PGSETUP_INITDB_OPTIONS="-k" /usr/pgsql-1*/bin/postgresql-17-setup initdb
Initializing database ... OK
```

Note that we enabled checksums in the above command by setting **PGSETUP_INITDB_OPTIONS="-k"**.

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

```
# whoami
root
# ls -la ~postgres/17
total 8
drwx-----. 4 postgres postgres 51 Oct 3 14:01 .
drwx-----. 3 postgres postgres 37 Oct 3 13:54 ..
drwx-----. 2 postgres postgres 6 Oct 3 06:18 backups
drwx-----. 20 postgres postgres 4096 Oct 3 14:01 data
-rw-----. 1 postgres postgres 923 Oct 3 14:01 initdb.log
```

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

```
# whoami
root
# /usr/pgsql-17/bin/postgresql-17-check-db-dir ~postgres/17/data
# echo $?
0
```

As long as the return code is zero(0), as shown, everything is fine and you may start the PostgreSQL service:

```
# whoami
root
# systemctl start postgresql-17
# systemctl status postgresql-17 | grep Active
Active: active (running) since Mon 2024-10-07 14:36:13 UTC; 1min 5s ago
```

Remediation:

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

```
# whoami
root
# PGSETUP_INITDB_OPTIONS="-k" /usr/pgsql-17/bin/postgresql-17-setup initdb
Data directory is not empty!
```







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

```
# whoami
root
# rm -rf ~postgres/17
# PGSETUP_INITDB_OPTIONS="-k" /usr/pgsql-17/bin/postgresql-17-setup initdb
Initializing database ... OK
```

References:

1. <https://www.postgresql.org/docs/current/app-initdb.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

1.5 Ensure the Latest Security Patches are Applied (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

PostgreSQL updates are released to resolve bugs, and mitigate vulnerabilities quarterly (or sooner for drastic CVEs). It is recommended that PostgreSQL installations are kept up to date with the latest security updates. The PostgreSQL development group *guarantees* that point releases (or "minor releases") *will not* change the behavior of an existing install and as such are "safe" to install without fear of changes to your application's behavior.

Rationale:

Maintaining parity with PostgreSQL patches will help reduce the risk associated with known vulnerabilities present in the PostgreSQL server.

Without the latest security patches, PostgreSQL might have known vulnerabilities which could be used by an attacker to gain access.

Impact:

To update the PostgreSQL server a restart is required which will cause a momentary service outage.

Audit:

Execute the following SQL statement as low-privileged user to identify the PostgreSQL server version:

```
# whoami
postgres
# psql -c 'SHOW server_version'
server_version
-----
17.0
(1 row)
```

Now compare the version returned with the security announcements shown on the PostgreSQL [news](#) page. For convenience, the latest PostgreSQL release versions are always shown in a banner at the top of that page along with the release date.

Remediation:

Install the latest patches available for your version:

RHEL:

```
sudo dnf update $(rpm -qa | grep '^postgresql')
```




Debian:

```
sudo apt-get install --only-upgrade $(dpkg-query -W -f '${db:Status-Status}
${Package}\n' 'postgresql*' | awk '$1 != "not-installed" {print $NF}')
```

References:

1. <https://www.postgresql.org/support/security/>
2. <https://www.postgresql.org/support/versioning/>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.2 <u>Ensure Authorized Software is Currently Supported</u> Ensure that only currently supported software is designated as authorized in the software inventory for enterprise assets. If software is unsupported, yet necessary for the fulfillment of the enterprise's mission, document an exception detailing mitigating controls and residual risk acceptance. For any unsupported software without an exception documentation, designate as unauthorized. Review the software list to verify software support at least monthly, or more frequently.			

1.6 Verify That 'PGPASSWORD' is Not Set in Users' Profiles (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

PostgreSQL can read a default database password from an environment variable called **PGPASSWORD**.

Rationale:

Use of the **PGPASSWORD** environment variable implies PostgreSQL credentials are stored as clear text. Avoiding this may increase assurance that the confidentiality of PostgreSQL credentials is preserved.

Audit:

To assess this recommendation check if **PGPASSWORD** is set in login scripts using the following terminal command as privileged user:

```
# whoami
root
# grep PGPASSWORD --no-messages /home/*/{bashrc,profile,bash_profile}
# grep PGPASSWORD --no-messages /root/{bashrc,profile,bash_profile}
# grep PGPASSWORD --no-messages /etc/environment
```

Note that the above only covers Bash as the login shell. If OS users are configured to use Zsh, Csh, or other login shells, the list of files would need adjusted appropriately.



Remediation:

Check which users and/or scripts are setting **PGPASSWORD** and change them to use a more secure authentication method.

References:

1. <https://www.postgresql.org/docs/current/libpq-envvars.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.11 <u>Encrypt Sensitive Data at Rest</u> Encrypt sensitive data at rest on servers, applications, and databases containing sensitive data. Storage-layer encryption, also known as server-side encryption, meets the minimum requirement of this Safeguard. Additional encryption methods may include application-layer encryption, also known as client-side encryption, where access to the data storage device(s) does not permit access to the plain-text data.			

1.7 Verify That the 'PGPASSWORD' Environment Variable is Not in Use (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

PostgreSQL can read a default database password from an environment variable called **PGPASSWORD**.

Rationale:

Using the **PGPASSWORD** environment variable implies PostgreSQL credentials are stored as clear text. Avoiding use of this environment variable can better safeguard the confidentiality of PostgreSQL credentials.

Audit:

To assess this recommendation, use the **/proc** filesystem and the following terminal command as a privileged user to determine if **PGPASSWORD** is currently set for any process.

```
# whoami  
root  
# grep PGPASSWORD /proc/*/environ
```

This may return one false-positive entry for the process which is executing the grep command.



Remediation:

Check which users and/or scripts are setting **PGPASSWORD** and change them to use a more secure method.

References:

1. <https://www.postgresql.org/docs/15/libpq-envvars.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.11 <u>Encrypt Sensitive Data at Rest</u> Encrypt sensitive data at rest on servers, applications, and databases containing sensitive data. Storage-layer encryption, also known as server-side encryption, meets the minimum requirement of this Safeguard. Additional encryption methods may include application-layer encryption, also known as client-side encryption, where access to the data storage device(s) does not permit access to the plain-text data.			

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 (Manual)

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 **0077** to deny file access to all user accounts except the owner.

Rationale:

The Linux OS defaults the umask to **0022**, 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 **0077** allows other users to read, write, or even execute files and scripts created by the **postgres** user account. The alternative to using a umask is explicitly updating file permissions after file creation using the command line utility **chmod** (a manual and error-prone process that is not advised).

Audit:

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

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

The umask must be **0077** or more restrictive for the **postgres** user, otherwise, this is a fail.

Remediation:

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

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







Default Value:

0022

References:

1. <https://man7.org/linux/man-pages/man2/umask.2.html>
2. <https://man7.org/linux/man-pages/man1/umask.1p.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

2.2 Ensure extension directory has appropriate ownership and permissions (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

The extension directory is the location of the PostgreSQL extensions. Extensions are storage engines or user defined functions (UDFs).

Rationale:

Limiting the accessibility of these objects will protect the confidentiality, integrity, and availability of the PostgreSQL database. If someone can modify extensions, then these extensions can be used to execute illicit instructions.

Audit:

Determine the PostgreSQL share directory:

```
# whoami
root
# /usr/pgsql-17/bin/pg_config --sharedir
/usr/pgsql-17/share
```

The extension directory, lives under that share directory:

```
# whoami
root
# ls -ld $(/usr/pgsql-17/bin/pg_config --sharedir)/extension
```

This should return:

```
drwxr-xr-x 1 root root 12288 Oct  7 14:27 /usr/pgsql-17/share/extension
```

Any differences in permissions (the first field) is a failure.




Remediation:

If needed, correct the permissions on the extension dir by executing:

```
# whoami
root
# chown -c root:root $(/usr/pgsql-17/bin/pg_config --sharedir)/extension
# chmod -c 0755 $(/usr/pgsql-17/bin/pg_config --sharedir)/extension
```

If the permissions needed correct, it is *imperative* that all extensions found in `$(/usr/pgsql-17/bin/pg_config --sharedir)/extension` are evaluated to ensure they have not been modified!

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			

2.3 Disable PostgreSQL Command History (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

On Linux/UNIX, the PostgreSQL client logs most interactive statements to a history file. The default PostgreSQL history file is named **.psql_history** in the user's home directory.

The PostgreSQL command history should be disabled.

Rationale:

Disabling the PostgreSQL command history reduces the probability of exposing sensitive information, such as passwords, encryption keys, or sensitive data.

Audit:

Execute the following command as privileged user to assess this recommendation:

```
# whoami
root
# find /home -name ".psql_history" -exec ls -la {} \;
# find /root -name ".psql_history" -exec ls -la {} \;
```

For each file returned, determine whether that file is symbolically linked to **/dev/null**:

```
lrwxrwxrwx 1 doug doug 9 Feb 26 18:18 /home/doug/.psql_history -> /dev/null
lrwxrwxrwx 1 jim jim 9 Feb 26 18:18 /home/jim/.psql_history
```

In the above, Jim's history file is a finding, while Doug's is not.

Remediation:

For each OS user on the PostgreSQL server, perform the following steps to implement this setting:

1. Remove **.psql_history** if it exists.

```
# whoami
root
# rm -f ~<user>/.psql_history || true
```

2. Use either of the techniques below to prevent it from being created again:
 1. Set the **HISTFILE** variable to **/dev/null** in **~<user>/.psqlrc**

```
# whoami
root
# cat << EOF >> ~<user>/.psqlrc
\set HISTFILE /dev/null
EOF
```

2. Create **~<user>/.psql_history** as a symbolic to **/dev/null**.

```
# whoami
root
# ln -s /dev/null $HOME/.psql_history
\.
```




3. Set the **PSQL_HISTORY** variable for all users:

```
# whoami
root
# echo 'PSQL_HISTORY=/dev/null' >> /etc/environment
```

References:

1. <https://www.postgresql.org/docs/current/app-psql.html#APP-PSQL-VARIABLES-HISTFILE>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.5 <u>Securely Dispose of Data</u> Securely dispose of data as outlined in the enterprise's data management process. Ensure the disposal process and method are commensurate with the data sensitivity.			

2.4 Ensure Passwords are Not Stored in the service file (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

One can set a **password** in a PostgreSQL connection service file. Verify the **password** option is not used in a connection service file.

Rationale:

Using the **password** parameter may negatively impact the confidentiality of the user's password.

Impact:

The global configuration is by default readable for all users on the system. This is needed for global defaults (prompt, port, socket, etc.). If a password is present in this file then all users on the system may be able to access it.

Audit:

To assess this recommendation, perform the following steps:

```
# whoami
root
# find / -name .pg_service.conf -type f -print | xargs -I{} grep -H password {}
# grep password /root/.pg_service.conf
# test -z "${PGSERVICEFILE}" || grep password "${PGSERVICEFILE}"
# test -z "${PGSYSCONFDIR}" || grep password "${PGSYSCONFDIR}/pg_service.conf"
```

If any of the commands above returns a line containing **password=...**, this is a finding.



Remediation:

Delete every **password** entry in the file(s) previously identified.

References:

1. <https://www.postgresql.org/docs/current/libpq-pgservice.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.11 <u>Encrypt Sensitive Data at Rest</u> Encrypt sensitive data at rest on servers, applications, and databases containing sensitive data. Storage-layer encryption, also known as server-side encryption, meets the minimum requirement of this Safeguard. Additional encryption methods may include application-layer encryption, also known as client-side encryption, where access to the data storage device(s) does not permit access to the plain-text data.			

3 Logging And Auditing

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

3.1 PostgreSQL Logging

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

3.1.1 Logging Rationale

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

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

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

3.1.2 Ensure the log destinations are set correctly (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Audit:

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

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

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

Remediation:

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

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

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

Default Value:

stderr











References:

1. <https://www.postgresql.org/docs/current/runtime-config-logging.html>

Additional Information:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.2 <u>Collect Audit Logs</u> Collect audit logs. Ensure that logging, per the enterprise's audit log management process, has been enabled across enterprise assets.			
v8	8.5 <u>Collect Detailed Audit Logs</u> Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.2 <u>Activate audit logging</u> Ensure that local logging has been enabled on all systems and networking devices.			
v7	6.3 <u>Enable Detailed Logging</u> Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

3.1.3 Ensure the logging collector is enabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

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 the server start.

Rationale:

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

Note: This setting *must* be enabled when **log_destination** is either **stderr** or **csvlog** or logs *will be lost*. Certain other logging parameters require it as well.

Audit:

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

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

Remediation:

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

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

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

```
# whoami
root
# systemctl restart postgresql-17
# systemctl status postgresql-17|grep 'ago$'
Active: active (running) since <date>; <count>s ago
```











Default Value:

on

References:

1. <https://www.postgresql.org/docs/current/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.2 Collect Audit Logs Collect audit logs. Ensure that logging, per the enterprise's audit log management process, has been enabled across enterprise assets.			
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.2 Activate audit logging Ensure that local logging has been enabled on all systems and networking devices.			
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

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

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Audit:

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

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

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

Remediation:

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

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

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











Default Value:

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

References:

1. <https://www.postgresql.org/docs/current/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.2 Collect Audit Logs Collect audit logs. Ensure that logging, per the enterprise's audit log management process, has been enabled across enterprise assets.			
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.2 Activate audit logging Ensure that local logging has been enabled on all systems and networking devices.			
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

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

Profile Applicability:

- Level 1 - PostgreSQL

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

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

Rationale:

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

Audit:

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

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

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

Remediation:

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

```

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

```

Note: In this example, a new log file will be created for each day (e.g. **postgresql-20200804.log**)











Default Value:

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

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.2 Collect Audit Logs Collect audit logs. Ensure that logging, per the enterprise's audit log management process, has been enabled across enterprise assets.			
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.2 Activate audit logging Ensure that local logging has been enabled on all systems and networking devices.			
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

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

Profile Applicability:

- Level 1 - PostgreSQL

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 either alter the `log_directory` setting to store the log files outside the cluster data directory or use `PGSETUP_INITDB_OPTIONS="-k -g"` when initializing the cluster.

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 the desired value of `0600`):

```

postgres=# alter system set log_file_mode = '0600';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf 
-----
 t
(1 row)
postgres=# show log_file_mode;
 log_file_mode 
-----
 0600
(1 row)

```







Default Value:

0600

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 Configure Data Access Control Lists Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 Protect Information through Access Control Lists Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

3.1.7 Ensure 'log_truncate_on_rotation' is enabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

```
postgresql-00.log
postgresql-01.log
[...]
postgresql-23.log
```

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

Rationale:

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

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

```
# truncation is moot, as each rotation gets a unique filename (postgresql-
20180605.log)
log_truncate_on_rotation = on
log_filename = 'postgresql-%Y%m%d.log'
log_rotation_age = '1d'
log_rotation_size = 0
# truncation every hour, losing log data every hour until the date changes
log_truncate_on_rotation = on
log_filename = 'postgresql-%Y%m%d.log'
log_rotation_age = '1h'
log_rotation_size = 0
# no truncation if the date changed before generating 100M of log data,
truncation otherwise
log_truncate_on_rotation = on
log_filename = 'postgresql-%Y%m%d.log'
log_rotation_age = '0'
log_rotation_size = '100M'
```

Audit:

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

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

If it is not set to **on**, this is a fail (depending on your organization's logging policy).

Remediation:

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

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

Default Value:

on






References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

Additional Information:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.3 <u>Ensure Adequate Audit Log Storage</u> Ensure that logging destinations maintain adequate storage to comply with the enterprise's audit log management process.			
v7	6.4 <u>Ensure adequate storage for logs</u> Ensure that all systems that store logs have adequate storage space for the logs generated.			

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

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

Log rotation is a standard best practice for log management.

Audit:

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

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

Remediation:

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

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






Default Value:

`1d` (one day)

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.3 <u>Ensure Adequate Audit Log Storage</u> Ensure that logging destinations maintain adequate storage to comply with the enterprise's audit log management process.			
v7	6.4 <u>Ensure adequate storage for logs</u> Ensure that all systems that store logs have adequate storage space for the logs generated.			

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

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Audit:

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

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

Remediation:

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

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






Default Value:

0

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.3 <u>Ensure Adequate Audit Log Storage</u> Ensure that logging destinations maintain adequate storage to comply with the enterprise's audit log management process.			
v7	6.4 <u>Ensure adequate storage for logs</u> Ensure that all systems that store logs have adequate storage space for the logs generated.			

3.1.10 Ensure the correct syslog facility is selected (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

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

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

Rationale:

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

Audit:

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

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

Remediation:

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

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







Default Value:

LOCAL0

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.2 <u>Collect Audit Logs</u> Collect audit logs. Ensure that logging, per the enterprise's audit log management process, has been enabled across enterprise assets.			
v7	6.2 <u>Activate audit logging</u> Ensure that local logging has been enabled on all systems and networking devices.			

3.1.11 Ensure syslog messages are not suppressed (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

When logging to Syslog and `syslog_sequence_numbers` is on, then each message will be prefixed by an increasing sequence number (such as [2]).

Rationale:

Many modern Syslog implementations perform a log optimization and suppress repeated log entries while emitting "--- last message repeated N times ---". In more modern Syslog implementations, repeated message suppression can be configured (for example, `$RepeatedMsgReduction` in `rsyslog`).

Impact:

If disabled, messages sent to Syslog could be suppressed and not logged. While a message is emitted stating that a given message was repeated and suppressed, the timestamp associated with these suppressed messages is lost, potentially damaging the recreation of an incident timeline.

Audit:

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

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

Remediation:

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

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



Default Value:

on

References:

1. <https://www.postgresql.org/docs/current/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			

3.1.12 Ensure syslog messages are not lost due to size (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

PostgreSQL log messages can exceed 1024 bytes, which is a typical size limit for traditional Syslog implementations. When `syslog_split_messages` is off, PostgreSQL server log messages are delivered to the Syslog service as is, and it is up to the Syslog service to cope with the potentially bulky messages. When `syslog_split_messages` is on, messages are split by lines, and long lines are split so that they will fit into 1024 bytes.

If syslog is ultimately logging to a text file, then the effect will be the same either way, and it is best to leave the setting on, since most syslog implementations either cannot handle large messages or would need to be specially configured to handle them. But if syslog is ultimately writing into some other medium, it might be necessary or more useful to keep messages logically together.

Rationale:

Impact:

Depending on the Syslog server in use, log messages exceeding 1024 bytes may be lost or, potentially, cause the Syslog server processes to abort.

Audit:

Execute the following SQL statement to confirm that long log messages are split when logging to Syslog:

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

Remediation:

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

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



Default Value:

on

References:

1. <https://www.postgresql.org/docs/current/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			

3.1.13 Ensure the program name for PostgreSQL syslog messages are correct (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Audit:

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

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

Remediation:

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

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

Default Value:





postgres

References:

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

2. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 <u>Collect Detailed Audit Logs</u> Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 <u>Enable Detailed Logging</u> Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

3.1.14 Ensure the correct messages are written to the server log (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Valid values are:

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

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

Rationale:

If this is not set to the correct value, too many 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)
```

If logging is not configured to at least `warning`, this is a fail.

Remediation:

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

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






Default Value:

WARNING

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.3 <u>Ensure Adequate Audit Log Storage</u> Ensure that logging destinations maintain adequate storage to comply with the enterprise's audit log management process.			
v7	6.4 <u>Ensure adequate storage for logs</u> Ensure that all systems that store logs have adequate storage space for the logs generated.			

3.1.15 Ensure the correct SQL statements generating errors are recorded (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

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

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

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

Rationale:

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

Audit:

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

```
postgres=# show log_min_error_statement;
log_min_error_statement
-----
error
(1 row)
```

If not configured to at least `error`, this is a fail.

Remediation:

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

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






Default Value:

ERROR

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.3 <u>Ensure Adequate Audit Log Storage</u> Ensure that logging destinations maintain adequate storage to comply with the enterprise's audit log management process.			
v7	6.4 <u>Ensure adequate storage for logs</u> Ensure that all systems that store logs have adequate storage space for the logs generated.			

3.1.16 Ensure 'debug_print_parse' is disabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

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)
```

If not configured to **off**, this is a fail.

Remediation:

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

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







Default Value:

off

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.1 Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>5.1 Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

3.1.17 Ensure 'debug_print_rewritten' is disabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

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)
```

If not configured to `off`, this is a fail.

Remediation:

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

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







Default Value:

`off`

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.1 Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>5.1 Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

3.1.18 Ensure 'debug_print_plan' is disabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Audit:

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

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

If not configured to `off`, this is a fail.

Remediation:

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

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







Default Value:

`off`

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.1 Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>5.1 Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

3.1.19 Ensure 'debug_pretty_print' is enabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Impact:

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

Audit:

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

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

If not configured to `on`, this is a fail.

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`

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 <u>Collect Detailed Audit Logs</u> Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 <u>Enable Detailed Logging</u> Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

3.1.20 Ensure 'log_connections' is enabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

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 the session start.

Rationale:

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

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

Audit:

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

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

If not configured to `on`, this is a fail.

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)
```

Then, in a new connection to the database, verify the change:

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

Note that you cannot verify this change in the same connection in which it was changed; a new connection is needed.





Default Value:

off

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 <u>Collect Detailed Audit Logs</u> Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 <u>Enable Detailed Logging</u> Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

3.1.21 Ensure 'log_disconnections' is enabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

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

Audit:

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

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

If not configured to `on`, this is a fail.

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





Default Value:

`off`

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

3.1.22 Ensure 'log_error_verbosity' is set correctly (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

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)
```

If not configured to `verbose`, this is a fail.

Remediation:

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

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





Default Value:

DEFAULT

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 <u>Collect Detailed Audit Logs</u> Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 <u>Enable Detailed Logging</u> Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

3.1.23 Ensure 'log_hostname' is set correctly (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Audit:

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

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

If not configured to `off`, this is a fail.

Remediation:

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

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







Default Value:

`off`

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 <u>Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	5.1 <u>Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

3.1.24 Ensure 'log_line_prefix' is set correctly (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

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

Any of the following escape sequences can be used:

```
%a = application name
%u = user name
%d = database name
%r = remote host and port
%h = remote host
%b = backend type
%p = process ID
%P = process ID of parallel group leader
%t = timestamp without milliseconds
%m = timestamp with milliseconds
%n = timestamp with milliseconds (as a Unix epoch)
%Q = query ID (0 if none or not computed)
%i = command tag
%e = SQL state
%c = session ID
%l = session line number
%s = session start timestamp
%v = virtual transaction ID
%x = transaction ID (0 if none)
%q = stop here in non-session processes
%% = '%'
```

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 [%p]
(1 row)
```

If the prefix does not at a minimum include '**%m [%r] [%p]: [1-%l] %u@%d, app=%a, e=%e** ' (for non-Syslog logging), this is a fail. For Syslog logging, the prefix should include **user=%u, db=%d, app=%a, client=%h**.

Remediation:

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

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





Default Value:

%m [%p]

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

3.1.25 Ensure 'log_statement' is set correctly (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

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

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

ddl logs all data definition statements:

- **CREATE**
- **ALTER**
- **DROP**

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

- **INSERT**
- **UPDATE**
- **DELETE**
- **TRUNCATE**
- **COPY FROM**

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

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

Rationale:

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

Audit:

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

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

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

Remediation:

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

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





Default Value:

none

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

3.1.26 Ensure 'log_timezone' is set correctly (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

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

Audit:

Execute the following SQL statement:

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

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

Remediation:

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

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





Default Value:

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

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 <u>Collect Detailed Audit Logs</u> Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.			
v7	6.3 <u>Enable Detailed Logging</u> Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.			

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

Profile Applicability:

- Level 1 - PostgreSQL

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.

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.

Audit:

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

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

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

Next, verify that desired auditing components are enabled:

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

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

- READ: **SELECT** and **COPY** when the source is a relation or a query.
- WRITE: **INSERT**, **UPDATE**, **DELETE**, **TRUNCATE**, and **COPY** when the destination is a relation.
- FUNCTION: Function calls and **DO** blocks.
- ROLE: Statements related to roles and privileges: **GRANT**, **REVOKE**, **CREATE/ALTER/DROP ROLE**.
- DDL: All **DDL** that is not included in the **ROLE** class.
- MISC: Miscellaneous commands, e.g. **DISCARD**, **FETCH**, **CHECKPOINT**, **VACUUM**.

Remediation:

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

```
# whoami
root
# dnf -y install pgaudit_17
[snip]
Installed:
  pgaudit_17-17.0-1PGDG.rhel9.x86_64

Complete!
```

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

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

and, finally, restart the PostgreSQL service:

```
# whoami
root
# vi ~postgres/17/data/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 at the end of the file:

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

Restart the PostgreSQL server for changes to take affect:

```
# whoami
root
# systemctl restart postgresql-17
# systemctl status postgresql-17|grep 'ago$'
Active: active (running) since [date] 10s ago
```







References:

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

Additional Information:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.2 <u>Collect Audit Logs</u> Collect audit logs. Ensure that logging, per the enterprise's audit log management process, has been enabled across enterprise assets.			
v7	6.2 <u>Activate audit logging</u> Ensure that local logging has been enabled on all systems and networking devices.			

4 User Access and Authorization

The capability to use database resources at a given level, known as 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 Interactive Login is Disabled (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

When created, the PostgreSQL user may have interactive access to the operating system, which means that the PostgreSQL user could login to the host as any other user would.

Rationale:

Preventing the PostgreSQL user from logging in interactively may reduce the impact of a compromised PostgreSQL account. There is also more accountability, as accessing the operating system where the PostgreSQL server lies will require the user's own account and the appropriate **sudo** configuration. Interactive access by the PostgreSQL user is unnecessary and should be disabled.

Impact:

This setting will prevent the PostgreSQL administrator from interactively logging into the operating system using the PostgreSQL user. Instead, the administrator will need to log in using one's own account and then **sudo** to the PostgreSQL administrator account.

Audit:

Execute the following terminal command as low-privileged user to assess this recommendation:

```
# whoami
root
# grep postgres /etc/shadow | cut -d: -f1-2
```




If this output is not **postgres: !<something>** then this is a failure.

Remediation:

Execute the following command:

```
# whoami
root
# passwd -l postgres
```

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			

4.2 Ensure sudo is configured correctly (Manual)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

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

Rationale:

Without **sudo**, there would be no capabilities to strictly control access to the superuser account nor to securely and authoritatively audit its use.

Audit:

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

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

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

```
# whoami
user2
# groups
user2 dba
# sudo -iu postgres
[sudo] password for user2:
# whoami
postgres
```

This shows that the **user2** user is configured properly for **sudo** access by being a member of the **dba** group.

Remediation:

As superuser **root**, execute the following commands:

```
# whoami
root
# echo '%dba ALL=(postgres) PASSWD: ALL' > /etc/sudoers.d/postgres
# chmod 600 /etc/sudoers.d/postgres
```







This grants any Operating System user that is a member of the **dba** group the ability to use **sudo -iu postgres** to become the **postgres** user.

Ensure that all Operating System user's that need such access are members of the group.

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.4 <u>Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	4.3 <u>Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

4.3 Ensure excessive administrative privileges are revoked (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

With respect to PostgreSQL administrative SQL commands, only superusers should have elevated privileges. PostgreSQL regular, or application, users should not possess the ability to create roles, create new databases, manage replication, or perform any other action deemed privileged. Typically, regular users should only be granted the minimal set of privileges commensurate with managing the application:

- DDL (`create table`, `create view`, `create index`, etc.)
- DML (`select`, `insert`, `update`, `delete`)

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

- `payroll_owner`
- `payroll_user`

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

Rationale:

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

Audit:

First, inspect the privileges granted to the database superuser (identified here as `postgres`) using the display command `psql -c "\du postgres"` to establish a baseline for granted administrative privileges. Based on the output below, the `postgres` superuser can create roles, create databases, manage replication, and bypass row-level security (RLS):

```
# whoami
postgres
# psql -c "\du+ postgres"
```

Role name	Attributes	Description
postgres	Superuser, Create role, Create DB, Replication, Bypass RLS	

Now, let's inspect the same information for a mock regular user called **appuser** using the display command **psql -c "\du+ appuser"**. The output confirms that regular user **appuser** has the same elevated privileges as system administrator user **postgres**. This is a fail.

```
# whoami
postgres
# psql -c "\du+ appuser"
```

Role name	Attributes	Description
appuser	Superuser, Create role, Create DB, Replication, Bypass RLS	

While this example demonstrated excessive administrative privileges granted to a single user, a comprehensive audit should be conducted to inspect all database users for excessive administrative privileges. This can be accomplished via either of the commands below.

```
# whoami
postgres
# psql -c "\du+ *"
# psql -c "SELECT * FROM pg_user WHERE usesuper ORDER BY username;"
```

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

Remediation:

If any regular or application users have been granted excessive administrative rights, those privileges should be removed immediately via the PostgreSQL **ALTER ROLE** SQL command. Using the same example above, the following SQL statements revoke all unnecessary elevated administrative privileges from the regular user **appuser**:

```
# whoami
postgres
# psql -c "ALTER ROLE appuser NOSUPERUSER;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOCREATEROLE;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOCREATEDB;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOREPLICATION;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOBYPASSRLS;"
ALTER ROLE
# psql -c "ALTER ROLE appuser NOINHERIT;"
ALTER ROLE
```







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

```
# whoami
postgres
# psql -c "\du+ appuser"
List of roles
Role name | Attributes | Description
-----+-----+-----
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:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 <u>Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	5.1 <u>Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

4.4 Lock Out Accounts if Not Currently in Use (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

If users with database accounts will not be using the database for some time, disabling the account will reduce the risk of attacks or inappropriate account usage.

Rationale:

Only actively used database accounts should be allowed to login to the database.

Audit:

Review the status of all database accounts:

```
# whoami
postgres
# psql
SELECT rolname FROM pg_catalog.pg_roles WHERE rolname !~ '^pg_' AND
rolcanlogin;
```

Inactive accounts should not be shown in the output.

Remediation:

To lock accounts, as a superuser, run:

```
ALTER ROLE <account> NOLOGIN;
```

To unlock accounts, as a superuser, run:

```
ALTER ROLE <account> LOGIN;
```

Default Value:

Accounts created by **CREATE ROLE** are NOLOGIN by default. Accounts created by **CREATE USER** are LOGIN by default.

Additional Information:

It is possible to specify NOLOGIN when using both **CREATE ROLE** and **CREATE USER**:

```
CREATE ROLE <account> NOLOGIN;
CREATE USER <account> NOLOGIN;
```


4.5 Ensure excessive function privileges are revoked (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Audit:

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

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

```
# whoami
root
# sudo -iu postgres
# psql -c "SELECT nspname, proname, proargtypes, proconfig, 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 proname NOT LIKE 'pgaudit%' AND
(proconfig OR NOT proconfig IS NULL);"
```

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

If elevation privileges are utilized which are not required or are expressly forbidden by organizational guidance, this is a fail.

Remediation:

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

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

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







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

Based on the output above, **appreader=X/postgres** no longer exists in the **proacl** column results returned from the 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>
4. <https://www.postgresql.org/docs/current/static/sql-createfunction.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 <u>Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	5.1 <u>Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

4.6 Ensure excessive DML privileges are revoked (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Audit:

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

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

List of roles
-[ RECORD 1 ]-----
Role name   | pg_checkpoint
Attributes  | Cannot login
Description |
-[ RECORD 2 ]-----
Role name   | pg_create_subscription
Attributes  | Cannot login
Description |
-[ RECORD 3 ]-----
Role name   | pg_database_owner
Attributes  | Cannot login
Description |
-[ RECORD 4 ]-----
Role name   | pg_execute_server_program
Attributes  | Cannot login
Description |
-[ RECORD 5 ]-----
Role name   | pg_maintain
```

```

Attributes | Cannot login
Description |
-[ RECORD 6 ]-----
Role name   | pg_monitor
Attributes  | Cannot login
Description |
-[ RECORD 7 ]-----
Role name   | pg_read_all_data
Attributes  | Cannot login
Description |
-[ RECORD 8 ]-----
Role name   | pg_read_all_settings
Attributes  | Cannot login
Description |
-[ RECORD 9 ]-----
Role name   | pg_read_all_stats
Attributes  | Cannot login
Description |
-[ RECORD 10 ]-----
Role name   | pg_read_server_files
Attributes  | Cannot login
Description |
-[ RECORD 11 ]-----
Role name   | pg_signal_backend
Attributes  | Cannot login
Description |
-[ RECORD 12 ]-----
Role name   | pg_stat_scan_tables
Attributes  | Cannot login
Description |
-[ RECORD 13 ]-----
Role name   | pg_use_reserved_connections
Attributes  | Cannot login
Description |
-[ RECORD 14 ]-----
Role name   | pg_write_all_data
Attributes  | Cannot login
Description |
-[ RECORD 15 ]-----
Role name   | pg_write_server_files
Attributes  | Cannot login
Description |
-[ RECORD 16 ]-----
Role name   | postgres
Attributes  | Superuser, Create role, Create DB, Replication, Bypass RLS
Description |
postgres=# \x
Expanded display is off.
postgres=#

                                List of relations
   Schema   |      Name      |  Type  | Owner  |
Persistence | Access method |         |        |
|   Size   | Description    |         |        |
-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----
information_schema | sql_features | table | postgres |

```

```

permanent | heap
| 104 kB |
information_schema | sql_implementation_info | table | postgres |
permanent | heap
| 48 kB |
information_schema | sql_parts | table | postgres |
permanent | heap
| 48 kB |
information_schema | sql_sizing | table | postgres |
permanent | heap
| 48 kB |
<snip>

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

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

```

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

```

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

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

```

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

Remediation:

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

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

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

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

tablename	username	select	insert	update	delete
customer	appwriter	t	t	t	t
customer	appreader	t	f	f	f

(2 rows)

Note: For versions of PostgreSQL prior to version 15, [CVE-2018-1058](#) is applicable and it is recommended that all privileges be revoked from the **public** schema for all users on all databases. If you have upgraded from one of these earlier releases, this CVE is not fixed for you during an upgrade. You can correct this CVE by issuing:

```
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
5. <https://nvd.nist.gov/vuln/detail/CVE-2018-1058>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 <u>Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	5.1 <u>Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

4.7 Ensure Row Level Security (RLS) is configured correctly (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

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

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

Rationale:

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

Audit:

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

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

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

```

postgres=# CREATE TABLE passwd (
  user_name          text UNIQUE NOT NULL,
  pwhash             text,
  uid                int  PRIMARY KEY,
  gid                int  NOT NULL,
  real_name          text NOT NULL,
  home_phone         text,
  extra_info         text,
  home_dir           text NOT NULL,
  shell              text NOT NULL
);
postgres=# SELECT oid, relname, relrowsecurity FROM pg_class WHERE relname =
'passwd';
   oid | relname | relrowsecurity 
-----+-----+-----
 24679 | passwd | f
(1 row)

```

Further inspection of RLS policies is provided via the system catalog `pg_policy`, which records policy details including table OID, policy name, applicable commands, the roles assigned to 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 USER admin;
CREATE USER
postgres=# CREATE USER bob;
CREATE USER
postgres=# CREATE USER alice;
CREATE USER

```

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
postgres=# SELECT oid, relname, relrowsecurity FROM pg_class WHERE relname =
'passwd';
   oid | relname | relrowsecurity
-----+-----+-----
 24679 | passwd | t
(1 row)

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Grant update to only the sensitive columns:

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

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

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

```
postgres=# set role admin;
SET
postgres=# table passwd;
 user_name | pwhash | uid | gid | real_name | home_phone | extra_info |
home_dir  | shell
-----+-----+-----+-----+-----+-----+-----+-----
admin     | xxx    | 0   | 0   | Admin     | 111-222-3333 |           |
/root     | /bin/dash
bob       | xxx    | 1   | 1   | Bob       | 123-456-7890 |           |
/home/bob | /bin/zsh
alice     | xxx    | 2   | 1   | Alice     | 098-765-4321 |           |
/home/alice | /bin/zsh
(3 rows)
postgres=# set role alice;
SET
postgres=# table passwd;
ERROR:  permission denied for table passwd
postgres=# select user_name,real_name,home_phone,extra_info,home_dir,shell
from passwd;
 user_name | real_name | home_phone | extra_info | home_dir | shell
-----+-----+-----+-----+-----+-----
admin     | Admin     | 111-222-3333 |           | /root    | /bin/dash
bob       | Bob       | 123-456-7890 |           | /home/bob | /bin/zsh
alice     | Alice     | 098-765-4321 |           | /home/alice | /bin/zsh
(3 rows)
postgres=# update passwd set user_name = 'joe';
ERROR:  permission denied for table passwd
-- Alice is allowed to change her own real_name, but no others
postgres=# update passwd set real_name = 'Alice Doe';
UPDATE 1
postgres=# update passwd set real_name = 'John Doe' where user_name =
'admin';
UPDATE 0
postgres=# select user_name,real_name,home_phone,extra_info,home_dir,shell
from passwd;
 user_name | real_name | home_phone | extra_info | home_dir | shell
-----+-----+-----+-----+-----+-----
admin     | Admin     | 111-222-3333 |           | /root    | /bin/dash
bob       | Bob       | 123-456-7890 |           | /home/bob | /bin/zsh
alice     | Alice Doe | 098-765-4321 |           | /home/alice | /bin/zsh
(3 rows)
postgres=# update passwd set shell = '/bin/xx';
ERROR:  new row violates WITH CHECK OPTION for "passwd"
```

```







postgres=# delete from passwd;
ERROR:  permission denied for table passwd
postgres=# insert into passwd (user_name) values ('xxx');
ERROR:  permission denied for table passwd
-- Alice can change her own password; RLS silently prevents updating other
rows
postgres=# update passwd set pwhash = 'abc';
UPDATE 1

```

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:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 <u>Configure Data Access Control Lists</u> Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.			
v7	14.6 <u>Protect Information through Access Control Lists</u> Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.			

4.8 Ensure the set_user extension is installed (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Note: Prior to performing this audit you must create a **roletree** view. Here are the procedures to create this view:

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

Rationale:

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

Impact:

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

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

Audit:

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

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

If the extension is not listed this is a fail.

Identify roles that are superusers and can still login:

```
postgres=# SELECT rolname FROM pg_authid WHERE rolsuper and rolcanlogin;
 rolname 
-----
postgres
(1 rows)
```

Identify any unprivileged roles that can log in directly that are granted a superuser role even if it is multiple layers removed:

Note: If you have not done so already, follow the procedures in the description to create a **roletree** view.

```
-- Verify there are no unexpected unprivileged roles that can login directly

SELECT
    r.rolname,
    r.roloid,
    r.rolcanlogin,
    r.rolsuper,
    r.rolparents
FROM roletree r
ORDER BY 1;

-- Verify there are no roles granted a superuser role even if it is multiple
layers
-- removed
SELECT
    ro.rolname,
    ro.roloid,
    ro.rolcanlogin,
    ro.rolsuper,
    ro.rolparents
FROM roletree ro
WHERE (ro.rolcanlogin AND ro.rolsuper)
OR
(
    ro.rolcanlogin AND EXISTS
    (
        SELECT TRUE FROM roletree ri
        WHERE ri.rolname = ANY (ro.rolparents)
        AND ri.rolsuper
    )
);
rolname | roloid | rolcanlogin | rolsuper | rolparents
-----+-----+-----+-----+-----
postgres |      10 | t          | t        | {}
(1 row)
```

A lack of results is a pass.

Remediation:

We will install the **set_user** extension:

```
# whoami
root
# dnf -y install set_user_17
[snip]
Installed:
    set_user_17-4.1.0-1.rhel9.1.x86_64

Complete!
```

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

```
# whoami
root
# vi ~postgres/17/data/postgresql.conf
```


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

```
shared_preload_libraries = 'set_user'

OR

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

Restart the PostgreSQL server for changes to take effect:

```
# systemctl restart postgresql-17
# systemctl status postgresql-17|grep 'ago$'
Active: active (running) since [timestamp]; 1s ago
```

And now, we can install the extension with SQL:

```
# su - postgres
# psql
postgres=# select * from pg_available_extensions where name = 'set_user';
   name   | default_version | installed_version |               comment
-----+-----+-----+-----
set_user  | 4.1.0           |                   | similar to SET ROLE but with
          |                 |                   | added logging
(1 row)

postgres=# create extension set_user;
CREATE EXTENSION
postgres=# select * from pg_available_extensions where name = 'set_user';
   name   | default_version | installed_version |               comment
-----+-----+-----+-----
set_user  | 4.1.0           | 4.1.0            | similar to SET ROLE but with
          |                 |                   | added logging
(1 row)
```

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

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

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

```
# whoami
psql
# psql -U doug -d postgres -h 127.0.0.1
postgres=> select set_user('postgres');
ERROR:  switching to superuser not allowed
HINT:   Use 'set_user_u' to escalate.
postgres=> select set_user_u('postgres');
   set_user_u
-----
      OK
(1 row)
postgres=# select current_user, session_user;
   current_user | session_user
-----+-----
   postgres    | doug
(1 row)
postgres=# select reset_user();
   reset_user
-----
      OK
(1 row)
postgres=> select current_user, session_user;
   current_user | session_user
-----+-----
      doug      | doug
(1 row)
```

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

```
postgres=# ALTER USER postgres NOLOGIN;
ALTER ROLE
```

Which results in:

```
$ psql
psql: FATAL:  role "postgres" is not permitted to log in
$ psql -U doug -d postgres -h 127.0.0.1
psql (1*.0)
```

Revoke SUPERUSER and/or LOGIN from any other roles that were previously identified:

```
postgres=# ALTER USER usera NOSUPERUSER; -- revoke superuser
ALTER ROLE
postgres=# ALTER USER usera NOLOGIN; -- revoke login
ALTER ROLE
postgres=# ALTER USER usera NOSUPERUSER NOLOGIN; -- revoke both at once
ALTER ROLE
```

Note that we show dropping the privileges both individually and as one. Pick an appropriate version based on your application/business needs.







Remove any escalated privileges on users granted indirectly that were previously identified using the **roletree** view:

```
postgres=# REVOKE name_of_granting_role FROM bob; -- an example only
REVOKE ROLE
```

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	<u>4.3 Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

4.9 Make use of predefined roles (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Audit:

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

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

Remediation:

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

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

And then remove **superuser** from the account:

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

Default Value:

The following predefined roles exist in PostgreSQL 17.x:

- `pg_read_all_data`

Read all data (tables, views, sequences), as if having `SELECT` rights on those objects, and `USAGE` rights on all schemas, even without having it explicitly. This role does not have the role attribute `BYPASSRLS` set. If RLS is being used, an administrator may wish to set `BYPASSRLS` on roles which this role is `GRANTED` to.

- `pg_write_all_data`

Write all data (tables, views, sequences), as if having `INSERT`, `UPDATE`, and `DELETE` rights on those objects, and `USAGE` rights on all schemas, even without having it explicitly. This role does not have the role attribute `BYPASSRLS` set. If RLS is being used, an administrator may wish to set `BYPASSRLS` on roles which this role is `GRANTED` to.

- `pg_read_all_settings`

Read all configuration variables, even those normally visible only to superusers.

- `pg_read_all_stats`

Read all `pg_stat_*` views and use various statistics related extensions, even those normally visible only to superusers.

- `pg_stat_scan_tables`

Execute monitoring functions that may take `ACCESS SHARE` locks on tables, potentially for a long time.

- `pg_monitor`

Read/execute various monitoring views and functions. This role is a member of `pg_read_all_settings`, `pg_read_all_stats` and `pg_stat_scan_tables`.

- `pg_database_owner`

None. Membership consists, implicitly, of the current database owner.

- `pg_signal_backend`

Signal another backend to cancel a query or terminate its session.

- `pg_read_server_files`

Allow reading files from any location the database can access on the server with `COPY` and other file-access functions.

- `pg_write_server_files`

Allow writing to files in any location the database can access on the server with `COPY` and other file-access functions.

- `pg_execute_server_program`

Allow executing programs on the database server as the user the database runs as with `COPY` and other functions which allow executing a server-side program.

- `pg_checkpoint`

Allow executing the `CHECKPOINT` command.

- `pg_maintain`

Allow executing `VACUUM`, `ANALYZE`, `CLUSTER`, `REFRESH MATERIALIZED VIEW`, `REINDEX`, and `LOCK TABLE` on all relations, as if having `MAINTAIN` rights on those objects, even without having it explicitly.

- `pg_use_reserved_connections`

Allow use of connection slots reserved via `reserved_connections`.

- `pg_create_subscription`







Allow users with `CREATE` permission on the database to issue `CREATE SUBSCRIPTION`.

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

References:

1. <https://www.postgresql.org/docs/current/predefined-roles.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>4.1 Establish and Maintain a Secure Configuration Process</u> Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.			
v7	<u>5.1 Establish Secure Configurations</u> Maintain documented, standard security configuration standards for all authorized operating systems and software.			

5 Connection and Login

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

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

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

5.1 Do Not Specify Passwords in the Command Line (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

When a command is executed on the command line, for example

- `psql postgresql://postgres:PASSWORD@host`

the password may be visible in the user's shell/command history or in the process list, thus exposing the password to other entities on the server.

Rationale:

If the password is visible in the process list or user's shell/command history, an attacker will be able to access the PostgreSQL database using the stolen credentials.

Audit:

- Check the process or task list if the password is visible.

```
# whoami
root
# ps -few
```

- Check the shell or command history if the password is visible.

```
# history
```

Remediation:

1. Use the `--password` or `-W` terminal parameter without directly specifying the password and then enter the password when prompted.

Substitute `<user>` with your username, e.g., root:



```
psql -u <user> --password
```

2. Do not use a [Connection URI](#) with password included, e.g. `psql postgresql://postgres:PASSWORD@host`
3. If desired, configure a `.pgpass` file with the proper credentials and secure the file appropriately.

References:

1. <https://www.postgresql.org/docs/current/libpq-connect.html#LIBPQ-CONNECT-PASSWORD>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 <u>Encrypt Sensitive Data in Transit</u> Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).			

5.2 Ensure PostgreSQL is Bound to an IP Address (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

By default, `listen_addresses` is set to `localhost` which prevents any and all remote TCP connections to the PostgreSQL port.

Some Docker images may set `listen_addresses` to `*`, which corresponds to *all* available IP interfaces; thus, the PostgreSQL server then accepts TCP connections on all the container's/server's IPv6 and IPv4 interfaces. (The same is true for a setting of `0.0.0.0`.)

You can make this configuration more restrictive by setting the `listen_addresses` configuration option to a specific list of IPv4 or IPv6 address so that the server only accepts TCP connections on those addresses.

This parameter can only be set at server start.

Rationale:

Limiting the IP addresses that PostgreSQL listens on provides additional restrictions on where client applications/users can connect from.

Audit:

Run the following statement:

```
# whoami
postgres
# psql -c 'SHOW listen_addresses'
```

If `*` or `0.0.0.0` is returned, this is a failure.

Remediation:

To have the PostgreSQL server only accept connections on a specific IP address, add an entry similar to this in the PostgreSQL configuration file `postgresql.conf`:



```
listen_addresses = '<your IP>'
```

To listen on multiple addresses, a comma-separated list may be used:

```
listen_addresses = '<your first IP>, <your second IP>'
```

In this case, clients can connect to the server using `--host=<your IP>`, while connections on other server host addresses are not possible.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<p><u>16.10 Apply Secure Design Principles in Application Architectures</u></p> <p>Apply secure design principles in application architectures. Secure design principles include the concept of least privilege and enforcing mediation to validate every operation that the user makes, promoting the concept of "never trust user input." Examples include ensuring that explicit error checking is performed and documented for all input, including for size, data type, and acceptable ranges or formats. Secure design also means minimizing the application infrastructure attack surface, such as turning off unprotected ports and services, removing unnecessary programs and files, and renaming or removing default accounts.</p>			

5.3 Ensure login via "local" UNIX Domain Socket is configured correctly (Manual)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

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

Rationale:

Audit:

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

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

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

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

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

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

Remediation:

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

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

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

```
# su - user1
# whoami
user1
# psql -u user1 -d postgres
postgres=>
```

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

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

#	TYPE	DATABASE	USER	ADDRESS	METHOD
	local	all	all		peer

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

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






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

# allow only postgres user logins locally via UNIX socket					
# TYPE	DATABASE	USER	ADDRESS		METHOD
local	all	postgres			peer
# allow all local users via UNIX socket					
# TYPE	DATABASE	USER	ADDRESS		METHOD
local	all	all			peer
# allow all local users, via UNIX socket, only if they are connecting to a db named the same as their username					
# e.g. if user 'bob' is connecting to a db named 'bob'					
# TYPE	DATABASE	USER			METHOD
local	samerole	all			peer
# allow only local users, via UNIX socket, who are members of the 'rw' role in the db					
# TYPE	DATABASE	USER	ADDRESS		METHOD
local	all	+rw			peer

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	6.5 <u>Require MFA for Administrative Access</u> Require MFA for all administrative access accounts, where supported, on all enterprise assets, whether managed on-site or through a third-party provider.			
v7	4.5 <u>Use Multifactor Authentication For All Administrative Access</u> Use multi-factor authentication and encrypted channels for all administrative account access.			

5.4 Ensure login via "host" TCP/IP Socket is configured correctly (Manual)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

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

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

METHODs `trust`, `password`, and `ident` are **not** to be used for remote logins.

METHOD `md5` is the most popular and can be used in both encrypted and unencrypted sessions, however, *it is vulnerable to packet replay attacks*. **It is recommended that `scram-sha-256` be used instead of `md5`.**

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

Rationale:

Audit:

Newly created data clusters are empty of data and have 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 local host.

A password must be assigned to each login ROLE:

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


Test an unencrypted session:

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

Test an encrypted session:

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

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

Test unencrypted session:

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

Test encrypted sessions:

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

Remediation:

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

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

```
listen_addresses = '*'
```

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

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

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

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

```
# allow 'postgres' user only from 'localhost/loopback' connections
# and only if you know the password
# (accepts both SSL and non-SSL connections)
# TYPE      DATABASE      USER      ADDRESS      METHOD
host        all            postgres  127.0.0.1/32  scram-sha-
256

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

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

Default Value:

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

If a password was encrypted using the `scram-sha-256` setting, then it can be used for the authentication methods `scram-sha-256`, `md5`, and `password` (but password transmission will be in plain text in the latter case).

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

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

To check the currently stored password hashes, see the system catalog `pg_authid`.

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





References:

1. <https://www.postgresql.org/docs/current/static/client-authentication.html>
2. <https://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html>
3. <https://tools.ietf.org/html/rfc7677>

Additional Information:

1. Use TYPE **hostssl** when administrating the database cluster as a superuser.
2. Use TYPE **hostnoss1** for performance purposes and when DML operations are deemed safe without SSL connections.
3. No examples have been given for ADDRESS, i.e., CIDR, hostname, domain names, etc.
4. Only three (3) types of METHOD have been documented; there are many more.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 <u>Encrypt Sensitive Data in Transit</u> Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).			
v7	14.4 <u>Encrypt All Sensitive Information in Transit</u> Encrypt all sensitive information in transit.			

5.5 Ensure per-account connection limits are used (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

Limiting concurrent connections to a PostgreSQL server can be used to reduce the risk of Denial of Service (DoS) attacks.

Rationale:

Limiting the number of concurrent sessions at the user level helps to reduce the risk of DoS attacks.

Audit:

To check the connection limits for all users, run the following:

```
# whoami
postgres
# psql
postgres=# SELECT rolname, rolconlimit
FROM pg_roles
WHERE rolname NOT LIKE 'pg_%';
```

Any user with a connection limit of **-1** should be considered a failure.

Remediation:

Set a per-user connection limit by running:

```
ALTER USER <dbuser> CONNECTION LIMIT <reasonable concurrent connection
count>;
```

Default Value:

-1

5.6 Ensure Password Complexity is configured (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

Password complexity configuration is crucial to restrict unauthorized access to data. By default, PostgreSQL doesn't provide for password complexity. Moreover, many compliance frameworks such as PCI DSS, and HIPPA require both password complexity and length. It is worth stating that the NIST 800-63B Password Guidelines publication is a good reference of authentication management.

Rationale:

Having strong password management for your locally-authenticated PostgreSQL accounts will protect against attackers' brute force techniques. This is important especially if external authentication is not possible to implement due to application requirements or restrictions.

Audit:

Check parameter values of both `shared_preload_libraries` and `dynamic_library_path`

```
postgres=# SHOW shared_preload_libraries;
shared_preload_libraries
-----
set_user,pgaudit
(1 row)
postgres=# SHOW dynamic_library_path;
dynamic_library_path
-----
$libdir
(1 row)
```

If `$libdir/passwordcheck` is not listed in `shared_preload_libraries` this is a failure (based on `$libdir` being returned for `dynamic_library_path`).

Remediation:

Next, we need to alter the `postgresql.conf` configuration file to enable `passwordcheck` as an extension in the `shared_preload_libraries` parameter and restart the PostgreSQL service:

```
# whoami
root
# vi ~postgres/17/data/postgresql.conf
```

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

```
shared_preload_libraries = '$libdir/passwordcheck'
```

OR

```
shared_preload_libraries = 'pgaudit,$libdir/passwordcheck,somethingelse'
```

Restart the PostgreSQL server for changes to take affect:

```
# whoami
root
# systemctl restart postgresql-17
# systemctl status postgresql-17|grep 'ago$'
Active: active (running) since [date] 10s ago
```




References:

1. <https://www.postgresql.org/docs/current/passwordcheck.html>

Additional Information:

Note that the **passwordcheck** functionality is actually of little value. Please see the 'caution' notice in the [documentation](#).

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.2 Use Unique Passwords Use unique passwords for all enterprise assets. Best practice implementation includes, at a minimum, an 8-character password for accounts using MFA and a 14-character password for accounts not using MFA.			

6 PostgreSQL Settings

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

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

Rather than describing all the possible combinations 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 Understanding attack vectors and runtime parameters (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

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

Impact:

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

Audit:

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





Remediation:

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

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>16.7 Use Standard Hardening Configuration Templates for Application Infrastructure</u> Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.			
v7	<u>18.11 Use Standard Hardening Configuration Templates for Databases</u> 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 (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

A denial of service is possible by denying the use of indexes and by slowing down client access to an unreasonable level. Unsourced 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.

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.

Audit:

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

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

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

Remediation:

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

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

```
ps -few | grep -E -- '[p]ost.*-[D]'
```

4. Examine the contents of `$PGDATA/postmaster.opts`

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.7 Use Standard Hardening Configuration Templates for Application Infrastructure Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.		●	●
v7	18.11 Use Standard Hardening Configuration Templates for Databases For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.		●	●

6.3 Ensure 'Postmaster' Runtime Parameters are Configured (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

PostgreSQL runtime parameters that are executed by the postmaster process.

Rationale:

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

Impact:

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

Audit:

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

```
postgres=# SELECT name, setting FROM pg_settings WHERE context = 'postmaster'
ORDER BY 1;
```

name	setting
archive_mode	off
autovacuum_freeze_max_age	200000000
autovacuum_max_workers	3
autovacuum_multixact_freeze_max_age	400000000
bonjour	off
bonjour_name	
cluster_name	
commit_timestamp_buffers	32
config_file	/var/lib/pgsql/17/data/postgresql.conf

data_directory	/var/lib/pgsql/17/data
data_sync_retry	off
debug_io_direct	
dynamic_shared_memory_type	posix
event_source	PostgreSQL
external_pid_file	
hba_file	/var/lib/pgsql/17/data/pg_hba.conf
hot_standby	on
huge_pages	try
huge_page_size	0
ident_file	/var/lib/pgsql/17/data/pg_ident.conf
ignore_invalid_pages	off
jit_provider	llvmjit
listen_addresses	localhost
logging_collector	on
max_connections	100
max_files_per_process	1000
max_locks_per_transaction	64
max_logical_replication_workers	4
max_notify_queue_pages	1048576
max_pred_locks_per_transaction	64
max_prepared_transactions	0
max_replication_slots	10
max_wal_senders	10
max_worker_processes	8
min_dynamic_shared_memory	0
multixact_member_buffers	32
multixact_offset_buffers	16
notify_buffers	16
port	5432
recovery_target	
recovery_target_action	pause
recovery_target_inclusive	on
recovery_target_lsn	
recovery_target_name	
recovery_target_time	
recovery_target_timeline	latest
recovery_target_xid	
reserved_connections	0
serializable_buffers	32
shared_buffers	16384
shared_memory_type	mmap
shared_preload_libraries	set_user,pgaudit,passwordcheck
subtransaction_buffers	32
superuser_reserved_connections	3
trace_connection_negotiation	off
track_activity_query_size	1024
track_commit_timestamp	off
transaction_buffers	32
unix_socket_directories	/run/postgresql, /tmp
unix_socket_group	
unix_socket_permissions	0777
wal_buffers	512
wal_decode_buffer_size	524288
wal_level	replica
wal_log_hints	off

(65 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 specified on the command-line invocation of `pg_ctl` the `restart` invocation is insufficient and an explicit `stop` and `start` must instead be made.

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

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





```
ps -few | grep -E -- '[p]ost.*-[D]'
```

4. Examine the contents of `$PGDATA/postmaster.opts`

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.7 Use Standard Hardening Configuration Templates for Application Infrastructure Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.			
v7	18.11 Use Standard Hardening Configuration Templates for Databases For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.			

6.4 Ensure 'SIGHUP' Runtime Parameters are Configured (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

PostgreSQL runtime parameters that are executed by the SIGHUP signal.

Rationale:

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

Impact:

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

Audit:

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

```
postgres=# SELECT name, setting FROM pg_settings WHERE context = 'sighup'
ORDER BY 1;
```

name	setting
allow_alter_system	on
archive_cleanup_command	
archive_command	(disabled)
archive_library	
archive_timeout	0
authentication_timeout	60
autovacuum	on

autovacuum_analyze_scale_factor	0.1
autovacuum_analyze_threshold	50
autovacuum_naptime	60
autovacuum_vacuum_cost_delay	2
autovacuum_vacuum_cost_limit	-1
autovacuum_vacuum_insert_scale_factor	0.2
autovacuum_vacuum_insert_threshold	1000
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	0.9
checkpoint_flush_after	32
checkpoint_timeout	300
checkpoint_warning	30
fsync	on
full_page_writes	on
gss_accept_delegation	off
hot_standby_feedback	off
krb_caseins_users	off
krb_server_keyfile	
FILE:/etc/sysconfig/postgresql/krb5.keytab	
log_autovacuum_min_duration	600000
log_checkpoints	on
log_destination	csvlog
log_directory	log
log_file_mode	0600
log_filename	postgresql-%a.log
log_hostname	off
log_line_prefix	%m [%p]: [%l-1]
db=%d,user=%u,app=%a,client=%h	
log_recovery_conflict_waits	off
log_rotation_age	1440
log_rotation_size	0
log_startup_progress_interval	10000
log_timezone	UTC
log_truncate_on_rotation	on
max_parallel_apply_workers_per_subscription	2
max_pred_locks_per_page	2
max_pred_locks_per_relation	-2
max_slot_wal_keep_size	-1
max_standby_archive_delay	30000
max_standby_streaming_delay	30000
max_sync_workers_per_subscription	2
max_wal_size	1024
min_wal_size	80
pre_auth_delay	0
primary_conninfo	
primary_slot_name	
recovery_end_command	
recovery_init_sync_method	fsync
recovery_min_apply_delay	0
recovery_prefetch	try
remove_temp_files_after_crash	on

restart_after_crash	on
restore_command	
send_abort_for_crash	off
send_abort_for_kill	off
set_user.block_alter_system	on
set_user.block_copy_program	on
set_user.block_log_statement	on
set_user.exit_on_error	on
set_user.nosuperuser_target_allowlist	*
set_user.superuser_allowlist	*
set_user.superuser_audit_tag	AUDIT
ssl	off
ssl_ca_file	
ssl_cert_file	server.crt
ssl_ciphers	HIGH:MEDIUM:+3DES:!aNULL
ssl_crl_dir	
ssl_crl_file	
ssl_dh_params_file	
ssl_ecdh_curve	prime256v1
ssl_key_file	server.key
ssl_max_protocol_version	
ssl_min_protocol_version	TLSv1.2
ssl_passphrase_command	
ssl_passphrase_command_supports_reload	off
ssl_prefer_server_ciphers	on
summarize_wal	off
synchronized_standby_slots	
synchronous_standby_names	
sync_replication_slots	off
syslog_facility	local0
syslog_ident	postgres
syslog_sequence_numbers	on
syslog_split_messages	on
wal_keep_size	0
wal_receiver_create_temp_slot	off
wal_receiver_status_interval	10
wal_receiver_timeout	60000
wal_retrieve_retry_interval	5000
wal_summary_keep_time	14400
wal_sync_method	fdatasync
wal_writer_delay	200
wal_writer_flush_after	128
(104 rows)	





Remediation:

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

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<p><u>16.7 Use Standard Hardening Configuration Templates for Application Infrastructure</u></p> <p>Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.</p>			
v7	<p><u>18.11 Use Standard Hardening Configuration Templates for Databases</u></p> <p>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.</p>			

6.5 Ensure 'Superuser' Runtime Parameters are Configured (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

PostgreSQL runtime parameters that can only be executed by the server's superuser, **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**.

Impact:

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

Audit:

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

```
postgres=# SELECT name, setting FROM pg_settings WHERE context = 'superuser'
ORDER BY 1;
```

name	setting
allow_in_place_tablespaces	off
allow_system_table_mods	off
backtrace_functions	
commit_delay	0
compute_query_id	auto
deadlock_timeout	1000
debug_discard_caches	0
dynamic_library_path	\$libdir
event_triggers	on
ignore_checksum_failure	off

jit_dump_bitcode	off
lc_messages	en_US.UTF-8
lo_compat_privileges	off
log_duration	off
log_error_verbosity	verbose
log_executor_stats	off
log_lock_waits	off
log_min_duration_sample	-1
log_min_duration_statement	-1
log_min_error_statement	error
log_min_messages	warning
log_parameter_max_length	-1
log_parser_stats	off
log_planner_stats	off
log_replication_commands	off
log_statement	ddl
log_statement_sample_rate	1
log_statement_stats	off
log_temp_files	-1
log_transaction_sample_rate	0
max_stack_depth	2048
pgaudit.log	ddl,write
pgaudit.log_catalog	on
pgaudit.log_client	off
pgaudit.log_level	log
pgaudit.log_parameter	off
pgaudit.log_parameter_max_size	0
pgaudit.log_relation	off
pgaudit.log_rows	off
pgaudit.log_statement	on
pgaudit.log_statement_once	off
pgaudit.role	
session_preload_libraries	
session_replication_role	origin
temp_file_limit	-1
track_activities	on
track_counts	on
track_functions	none
track_io_timing	off
track_wal_io_timing	off
update_process_title	on
wal_compression	off
wal_consistency_checking	
wal_init_zero	on
wal_recycle	on
zero_damaged_pages	off
(56 rows)````	

Remediation:

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

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





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

4. Examine the contents of `$PGDATA/postmaster.opts`

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.7 Use Standard Hardening Configuration Templates for Application Infrastructure Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.			
v7	18.11 Use Standard Hardening Configuration Templates for Databases For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.			

6.6 Ensure 'User' Runtime Parameters are Configured (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

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

Rationale:

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

Impact:

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

Audit:

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

```
postgres=# SELECT name, setting FROM pg_settings WHERE context = 'user' ORDER BY 1;
```

name	setting
application_name	psql
array_nulls	on
backend_flush_after	0
backslash_quote	safe_encoding
bytea_output	hex
check_function_bodies	on
client_connection_check_interval	0
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
creatorole_self_grant	
cursor_tuple_fraction	0.1
DateStyle	ISO, MDY
debug_logical_replication_streaming	buffered
debug_parallel_query	off

debug_pretty_print	on
debug_print_parse	off
debug_print_plan	off
debug_print_rewritten	off
default_statistics_target	100
default_table_access_method	heap
default_tablespace	
default_text_search_config	pg_catalog.english
default_toast_compression	pglz
default_transaction_deferrable	off
default_transaction_isolation	read committed
default_transaction_read_only	off
effective_cache_size	524288
effective_io_concurrency	1
enable_async_append	on
enable_bitmapscan	on
enable_gathermerge	on
enable_group_by_reordering	on
enable_hashagg	on
enable_hashjoin	on
enable_incremental_sort	on
enable_indexonlyscan	on
enable_indexscan	on
enable_material	on
enable_memoize	on
enable_mergejoin	on
enable_nestloop	on
enable_parallel_append	on
enable_parallel_hash	on
enable_partition_pruning	on
enable_partitionwise_aggregate	off
enable_partitionwise_join	off
enable_presorted_aggregate	on
enable_seqscan	on
enable_sort	on
enable_tidscan	on
escape_string_warning	on
exit_on_error	off
extra_float_digits	1
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
hash_mem_multiplier	2
icu_validation_level	warning
idle_in_transaction_session_timeout	0
idle_session_timeout	0
IntervalStyle	postgres
io_combine_limit	16
jit	on
jit_above_cost	100000

jit_expressions	on
jit_inline_above_cost	500000
jit_optimize_above_cost	500000
jit_tuple_deforming	on
join_collapse_limit	8
lc_monetary	en_US.UTF-8
lc_numeric	en_US.UTF-8
lc_time	en_US.UTF-8
local_preload_libraries	
lock_timeout	0
logical_decoding_work_mem	65536
log_parameter_max_length_on_error	0
maintenance_io_concurrency	10
maintenance_work_mem	65536
max_parallel_maintenance_workers	2
max_parallel_workers	8
max_parallel_workers_per_gather	2
min_parallel_index_scan_size	64
min_parallel_table_scan_size	1024
parallel_leader_participation	on
parallel_setup_cost	1000
parallel_tuple_cost	0.1
password_encryption	scram-sha-256
plan_cache_mode	auto
quote_all_identifiers	off
random_page_cost	4
recursive_worktable_factor	10
restrict_nonsystem_relation_kind	
row_security	on
scram_iterations	4096
search_path	"\$user", public
seq_page_cost	1
standard_conforming_strings	on
statement_timeout	0
stats_fetch_consistency	cache
synchronize_seqscans	on
synchronous_commit	on
tcp_keepalives_count	0
tcp_keepalives_idle	0
tcp_keepalives_interval	0
tcp_user_timeout	0
temp_buffers	1024
temp_tablespace	
TimeZone	UTC
timezone_abbreviations	Default
trace_notify	off
trace_sort	off
transaction_deferrable	off
transaction_isolation	read committed
transaction_read_only	off
transaction_timeout	0
transform_null_equals	off
vacuum_buffer_usage_limit	2048
vacuum_cost_delay	0
vacuum_cost_limit	200
vacuum_cost_page_dirty	20
vacuum_cost_page_hit	1


```
vacuum_cost_page_miss | 2
vacuum_failsafe_age | 1600000000
vacuum_freeze_min_age | 50000000
vacuum_freeze_table_age | 150000000
vacuum_multixact_failsafe_age | 1600000000
vacuum_multixact_freeze_min_age | 5000000
vacuum_multixact_freeze_table_age | 150000000
wal_sender_timeout | 60000
wal_skip_threshold | 2048
work_mem | 4096
xmlbinary | base64
xmloption | content
(145 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 their default value(s).

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.7 <u>Use Standard Hardening Configuration Templates for Application Infrastructure</u> Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.			
v7	18.11 <u>Use Standard Hardening Configuration Templates for Databases</u> For applications that rely on a database, use standard hardening configuration templates. All systems that are part of critical business processes should also be tested.			

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

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. PostgreSQL uses OpenSSL for the underlying encryption layer.

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

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

Audit:

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

As the system administrator, run the following to see if FIPS is enabled:

```
# whoami
root
# fips-mode-setup --check
Installation of FIPS modules is not completed.
FIPS mode is disabled.
```

If **FIPS mode is enabled** is not displayed, then the system is not FIPS enabled and this is a fail.

Remediation:

Configure OpenSSL to be FIPS compliant as PostgreSQL uses OpenSSL for cryptographic modules. To configure OpenSSL to be FIPS 140-2 compliant, see the [official RHEL Documentation](#). Below is a general summary of the steps required:
To switch the system to FIPS mode in RHEL 9:

```
# whoami
root
# fips-mode-setup --enable
Kernel initramdisks are being regenerated. This might take some time.
Setting system policy to FIPS
Note: System-wide crypto policies are applied on application start-up.
It is recommended to restart the system for the change of policies
to fully take place.
FIPS mode will be enabled.
Please reboot the system for the setting to take effect.
```

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

```
# whoami
root
# reboot
```



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



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

References:

1. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/security_hardening/using-the-system-wide-cryptographic-policies_security-hardening#switching-the-system-to-fips-mode_using-the-system-wide-cryptographic-policies
2. <https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp1758.pdf>
3. <https://csrc.nist.gov/publications/fips>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).			

Controls Version	Control	IG 1	IG 2	IG 3
v7	14.4 <u>Encrypt All Sensitive Information in Transit</u> Encrypt all sensitive information in transit.			

6.8 Ensure TLS is enabled and configured correctly (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

TLS on a PostgreSQL server should be enabled and configured to encrypt TCP traffic to and from the server.

Rationale:

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

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 TLS 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 TLS encryption. The `host` record matches attempts made using TCP/IP, but allows both TLS and non-TLS connections. The `hostnoss1` record matches attempts made using TCP/IP, but only those *without* TLS. *Care should be taken to enforce TLS as appropriate.*

Audit:

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

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

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

Remediation:

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

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

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

# # remove passphrase (required for automatic server start up, if not using
`ssl_passphrase_command`)
# openssl rsa -in privkey.pem -out server.key && rm privkey.pem
Enter pass phrase for privkey.pem:
writing RSA key

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

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

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

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

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

# force clients to use TLS v1.3 or newer
ssl_min_protocol_version = 'TLSv1.3'

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

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

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

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

Default Value:

Note that **server.crt** and **server.key** are the default names used by PostgreSQL. These files can be named otherwise, just ensure you update the **postgresql.conf** to use these new names. The current names can be found via SQL:

```
postgres=# select name, setting from pg_settings where name like 'ssl%file';
```

name	setting
ssl_ca_file	
ssl_cert_file	server.crt
ssl_crl_file	
ssl_dh_params_file	
ssl_key_file	server.key

(5 rows)

References:

1. <https://www.postgresql.org/docs/current/static/ssl-tcp.html>
2. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf>
3. <https://www.postgresql.org/docs/current/static/libpq-ssl.html>
4. <https://www.postgresql.org/docs/current/runtime-config-connection.html>

Additional Information:

It is possible to leave the passphrase on **privkey.pem** for additional security if desired. Simply configure **postgresql.conf** to set:





```
ssl_passphrase_command = 'your_command_here'
```

This sets an external command to be invoked when a passphrase for decrypting an SSL file such as a private key needs to be obtained. By default, this parameter is empty, which means the built-in prompting mechanism is used.

The command must print the passphrase to the standard output and exit with code 0. In the parameter value, %p is replaced by a prompt string. (Write %% for a literal %.) Note that the prompt string will probably contain whitespace, so be sure to quote adequately. A single newline is stripped from the end of the output if present.

The command does not actually have to prompt the user for a passphrase. It can read it from a file, obtain it from a keychain facility, or similar. It is up to the user to make sure the chosen mechanism is adequately secure.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 <u>Encrypt Sensitive Data in Transit</u> Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).			
v7	14.4 <u>Encrypt All Sensitive Information in Transit</u> Encrypt all sensitive information in transit.			

6.9 Ensure that TLSv1.3, or later, is configured (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

Transport Layer Security (TLS), and its predecessor Secure Sockets Layer (SSL) are cryptographic protocols which can be used to encrypt data sent between client and server.

Rationale:

The TLSv1.0 protocol is vulnerable to the BEAST attack when used in CBC mode (October 2011). TLSv1.0 uses CBC modes for all of the block mode ciphers, which only leaves the RC4 streaming cipher which is also weak and therefore **not** recommended. As such, it is recommended that the TLSv1.0 protocol is disabled.

The TLSv1.1 protocol does not support *Authenticated Encryption with Associated Data* (AEAD) which is designed to simultaneously provide confidentiality, integrity, and authenticity and should therefore be disabled.

The TLSv1.2 protocol includes a number of older vulnerable cryptographic algorithms and lacks quantum secure algorithms. It should only be used in situations where TLSv1.3 *cannot* be used.

The TLSv1.3 removes older, known vulnerable, cryptographic algorithms, is much faster, and incorporates quantum secure algorithms. It should be deployed by default where possible.

IETF deprecated TLSv1.0 and TLSv1.1 in March 2021 (see *RFC 8996*).

Audit:

1. Execute the following command

```
postgres=# SHOW ssl_min_protocol_version;
ssl_min_protocol_version
-----
TLSv1.2
(1 row)
```

2. Check the output to verify that the `ssl_min_protocol_version` directive is set to **TLSv1.3**.

Remediation:

Adjust the `ssl_min_protocol_version` to at least TLSv1.3:

```
postgres=# ALTER SYSTEM SET ssl_min_protocol_version = 'TLSv1.3';
ALTER SYSTEM
```



Make the change active:

```
postgres=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 row)
postgres=# SHOW ssl_min_protocol_version;
ssl_min_protocol_version
-----
TLSv1.3
(1 row)
```

References:

1. <https://ssl-config.mozilla.org/>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 <u>Encrypt Sensitive Data in Transit</u> Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).			

6.10 Ensure Weak SSL/TLS Ciphers Are Disabled (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

The PostgreSQL `ssl_ciphers` directive specifies which Cipher Suites are allowed in the negotiation with the client.

In cryptography, *perfect forward secrecy* (PFS), also known as *forward secrecy* (FS), is a feature of specific key exchange protocols that give assurance that the session keys will not be compromised even if the private key of the server is compromised. For instance, `RSA` does not provide PFS, while the `ECDHE` (Elliptic-Curve Diffie-Hellman Ephemeral) and `DHE` (Diffie-Hellman Ephemeral) provides PFS.

`ECDHE` is the stronger protocol and should be preferred, while `DHE` may be allowed for greater compatibility with older clients. Only Cipher Suites with either the `ECDHE` or the `DHE` key exchange are allowed.

Rationale:

The SSL/TLS protocols support a large number of Cipher Suites including many weak and medium strength algorithms that are subject to man-in-the middle attacks and information disclosure. Some implementations even support the `NULL` Cipher Suite which allows a TLS connection without any cryptographic protection. Therefore, it is critical to ensure the configuration only allows strong algorithms greater than or equal to 128-bit to be negotiated with the client. Stronger 256-bit algorithms should be allowed and preferred.

Furthermore, during the TLS handshake, after the initial *Client Hello* and *Server Hello*, there is a pre-master secret generated, which is used to generate the master secret, and in turn generates the session key. When using protocols that do not provide forward secrecy, such as `RSA`, the pre-master secret is encrypted by the client with the server's public key and sent over the network. However, with protocols such as `ECDHE` (Elliptic-Curve Diffie-Hellman Ephemeral) the pre-master secret is not sent over the wire, even in encrypted format. The key exchange arrives at the shared secret in the clear using ephemeral keys that are not stored or used again. With forward secrecy, each session has a unique key exchange, so that future sessions are protected.

Note This recommendation is primarily targeted at those installs that cannot run in FIPS-mode, or need to further refine the allowable cipher list.

Audit:

1. Execute the following command

```
# whoami
root
# grep "ssl_ciphers" ~postgres/17/data/postgresql.conf | cut -d ' ' -f
3 | sed "s/'//g" | tr ":" "\n"
```

2. Check the output to verify that the **ssl_ciphers** directive is set to a subset of the following values in the PostgreSQL configuration file (**postgresql.conf**)

```
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_AES_128_CCM_SHA256
TLS_CHACHA20_POLY1305_SHA256
ECDHE-ECDSA-AES256-CCM
ECDHE-ECDSA-AES128-CCM
DHE-RSA-AES256-CCM
DHE-RSA-AES128-CCM
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES128-GCM-SHA256
DHE-DSS-AES256-GCM-SHA384
DHE-DSS-AES128-GCM-SHA256
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES128-GCM-SHA256
ECDHE-ECDSA-CHACHA20-POLY1305
ECDHE-RSA-CHACHA20-POLY1305
DHE-RSA-CHACHA20-POLY1305
```

Remediation:

Add or modify the **ssl_ciphers** directive to the following value in the PostgreSQL configuration file (**postgresql.conf**):

```
ssl_ciphers =
'TLS_AES_256_GCM_SHA384,TLS_AES_128_GCM_SHA256,TLS_AES_128_CCM_SHA256,TLS_CHA
CHA20_POLY1305_SHA256,ECDHE-ECDSA-AES256-CCM,ECDHE-ECDSA-AES128-CCM,DHE-RSA-
AES256-CCM,DHE-RSA-AES128-CCM,ECDHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-AES128-
GCM-SHA256,ECDHE-ECDSA-AES256-GCM-SHA384,ECDHE-ECDSA-AES128-GCM-SHA256,DHE-
DSS-AES256-GCM-SHA384,DHE-DSS-AES128-GCM-SHA256,DHE-RSA-AES256-GCM-
SHA384,DHE-RSA-AES128-GCM-SHA256'
```

References:

1. <https://ssl-config.mozilla.org/>

6.11 Ensure the pgcrypto extension is installed and configured correctly (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

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 instances handling data that requires "data at rest" protections must employ cryptographic mechanisms to prevent unauthorized disclosure and modification of the information at rest. These cryptographic mechanisms may be native to PostgreSQL or implemented via additional software or operating system/file system settings, as appropriate to the situation. Information at rest refers to the state of information when it is located on a secondary storage device (e.g. disk drive, tape drive) within an organizational information system.

The selection of a cryptographic mechanism is based on the need to protect the integrity of organizational information. The strength of the mechanism is commensurate with the security category and/or classification of the information. Organizations have the flexibility to either encrypt all information on storage devices (i.e. full disk encryption) or encrypt specific data structures (e.g. files, records, or fields). Organizations may also optionally choose to implement both to implement layered security.

The decision of 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 protection, as appropriate. If the confidentiality and integrity of application data are 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.

Impact:

When considering or undertaking any form of encryption, it is critical to understand the state of the encrypted data at all stages of the data lifecycle. The use of **pgcrypto** ensures that the data at rest in the tables (and therefore on disk) is encrypted, but for the data to be accessed by any users or applications, said users/applications will, by necessity, have access to the encrypt and decrypt keys and the data in question will be encrypted/decrypted in memory and then transferred to/from the user/application in that form.

Audit:

One possible way to encrypt data within PostgreSQL is to use the **pgcrypto** extension. To check if **pgcrypto** is installed on PostgreSQL, as a database administrator run the following commands:

```
postgres=# SELECT * FROM pg_available_extensions WHERE name='pgcrypto';
```

name	default_version	installed_version	comment
pgcrypto	1.3		cryptographic functions

(1 row)

If data in the database requires encryption and **pgcrypto** is not available, this is a fail.

If disk or filesystem requires encryption, ask the system owner, DBA, and SA to demonstrate the use of disk-level encryption. If this is required and is not found, this is a fail. If controls do not exist or are not enabled, this is also a fail.

Remediation:

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

As the database administrator, run the following:

```
postgres=# CREATE EXTENSION pgcrypto;
```

CREATE EXTENSION

Verify **pgcrypto** is installed:

```
postgres=# SELECT * FROM pg_available_extensions WHERE name='pgcrypto';
```




name	default_version	installed_version	comment
pgcrypto	1.3	1.3	cryptographic functions

(1 row)

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.11 <u>Encrypt Sensitive Data at Rest</u> Encrypt sensitive data at rest on servers, applications, and databases containing sensitive data. Storage-layer encryption, also known as server-side encryption, meets the minimum requirement of this Safeguard. Additional encryption methods may include application-layer encryption, also known as client-side encryption, where access to the data storage device(s) does not permit access to the plain-text data.			
v7	14.8 <u>Encrypt Sensitive Information at Rest</u> Encrypt all sensitive information at rest using a tool that requires a secondary authentication mechanism not integrated into the operating system, in order to access the information.			

7 Replication

Data redundancy often plays a major role as part of an overall database strategy. Replication is an example of data redundancy and fulfills both High Availability and High Performance requirements. However, although the DBA may have expended much time and effort securing the PRIMARY host and taken the time to harden STANDBY configuration parameters, one sometimes overlooks the medium transmitting the data itself over the network. Consequently, replication is an appealing attack vector given that all DDL, and DML operations executed on the PRIMARY host is sent over the wire to the SECONDARY/STANDBY host(s). Fortunately, when correctly understood, defeating such attacks can be implemented in a straightforward 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 them into a read-only server. What's different is that the connection uses a special replication protocol which is faster and more efficient than log shipping. Similar to standard client connections, it also honors the same authentication rules as expressed in the PostgreSQL host-based authentication file, `pg_hba.conf`.

7.1 Ensure a replication-only user is created and used for streaming replication (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

Create a new user specifically for use by streaming replication instead of using the superuser account.

Rationale:

As it is not necessary to be a superuser to initiate a replication connection, it is proper to create an account specifically for replication. This allows further 'locking down' the uses of the superuser account and follows the general principle of using the least privileges necessary.

Audit:

Check which users currently have the replication permission:

```
postgres=# SELECT rolname FROM pg_roles WHERE rolreplication;
rolname
-----
postgres
(1 row)
```

In a default PostgreSQL cluster, only the **postgres** user will have this permission.

Remediation:

It will be necessary to create a new role for replication purposes:

```
postgres=# CREATE USER replication_user REPLICATION ENCRYPTED PASSWORD 'xxx';
CREATE ROLE
postgres=# SELECT rolname FROM pg_roles WHERE rolreplication;
rolname
-----
postgres
replication_user
(2 rows)
```

When using **pg_basebackup** (or other replication tools) on your standby server, you would use the **replication_user** (and its password).







Ensure you allow the new user via your **pg_hba.conf** file:

```
# note that 'replication' in the 2nd column is required and is a special
# keyword, not a real database
hostssl replication      replication_user    0.0.0.0/0          scram-sha-256
```

References:

1. <https://www.postgresql.org/docs/current/static/app-pgbasebackup.html>
2. <https://www.postgresql.org/docs/current/high-availability.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.4 <u>Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	4.3 <u>Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

7.2 Ensure logging of replication commands is configured (Automated)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

Enabling the `log_replication_commands` setting causes each attempted replication from the server to be logged.

Rationale:

A successful replication connection allows for a complete copy of the data stored within the data cluster to be offloaded to another, potentially insecure, host. As such, it is advisable to log all replication commands that are executed in your database cluster to ensure the data is not off-loaded to an unexpected/undesired location.

Audit:

Check the current value of `log_replication_commands`:

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

Remediation:







To enable the logging of replication commands, execute the following:

```
postgres=# ALTER SYSTEM SET log_replication_commands = 'on';
ALTER SYSTEM
postgres=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 row)
postgres=# SHOW log_replication_commands ;
log_replication_commands
-----
on
(1 row)
```

References:

1. <https://www.postgresql.org/docs/current/runtime-config-logging.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	<u>5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts</u> Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.			
v7	<u>4.3 Ensure the Use of Dedicated Administrative Accounts</u> Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.			

7.3 Ensure base backups are configured and functional (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

A 'base backup' is a copy of the PRIMARY host's data cluster (**\$PGDATA**) and is used to create STANDBY hosts and for Point In Time Recovery (PITR) mechanisms. Base backups should be copied across networks in a secure manner using an encrypted transport mechanism. The PostgreSQL CLI **pg_basebackup** can be used, however, TLS encryption should be enabled on the server as per section 6.8 of this benchmark. The pgBackRest tool detailed in section 8.2 of this benchmark can also be used to create a 'base backup'.

Remediation:





Executing base backups using **pg_basebackup** requires the following steps on the **standby** server:

```
$ whoami
postgres
$ pg_basebackup --host=name_or_IP_of_master \
--port=5432 \
--username=replication_user \
--pgdata=~postgres/17/data \
--progress \
--verbose \
--write-recovery-conf \
--wal-method=stream \
--checkpoint=fast
```

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:

Controls Version	Control	IG 1	IG 2	IG 3
v8	11.5 <u>Test Data Recovery</u> Test backup recovery quarterly, or more frequently, for a sampling of in-scope enterprise assets.			
v7	10.3 <u>Test Data on Backup Media</u> Test data integrity on backup media on a regular basis by performing a data restoration process to ensure that the backup is properly working.			

7.4 Ensure WAL archiving is configured and functional (Automated)

Profile Applicability:

- Level 1 - PostgreSQL on Linux

Description:

Write Ahead Log (WAL) Archiving, or Log Shipping, is the process of sending transaction log files from the PRIMARY host either to one or more STANDBY hosts or to be archived on a remote storage device for later use, e.g. **PITR**. There are several utilities that can copy WALs including, but not limited to, **cp**, **scp**, **sftp**, and **rynsc**. Basically, the server follows a set of runtime parameters which define 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 parameters:

```
postgres=# SELECT name, setting
FROM pg_settings
WHERE name IN ('archive_mode', 'archive_command', 'archive_library')
AND setting IS NOT NULL
AND setting <> 'off'
AND setting <> '(disabled)'
AND setting <> '';

 name | setting
-----+-----
(0 rows)
```

If no rows are returned, as shown, this is a failure. If rows are returned, note that **archive_mode** must be 'on' and either **archive_command** or **archive_library** must be set for WAL archiving to be fully enabled.

To verify that WAL archiving is functioning successfully, one can:

```
postgres=# SELECT * FROM pg_stat_archiver; \watch
```

And ensure that **archived_count**, **last_archived_wal**, and **last_archived_time** are changing while **failed_count**, **last_failed_wal**, and **last_failed_time** are not changing.





Remediation:

Change parameters and restart the server as required.

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-wal.html#RUNTIME-CONFIG-WAL-ARCHIVING>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 <u>Encrypt Sensitive Data in Transit</u> Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).			
v7	14.4 <u>Encrypt All Sensitive Information in Transit</u> Encrypt all sensitive information in transit.			

7.5 Ensure streaming replication parameters are configured correctly (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

Streaming replication from a PRIMARY host transmits DDL, DML, passwords, and other potentially sensitive activities and data. These connections should be protected with Secure Sockets Layer (SSL).

Rationale:

Unencrypted transmissions could reveal sensitive information to unauthorized parties. Unauthenticated connections could enable man-in-the-middle attacks.

Audit:

Confirm a dedicated and non-superuser role with replication permission exists:

```
postgres=> SELECT rolname FROM pg_roles WHERE rolreplication;
           rolname
-----
postgres
replication_user
(2 rows)
```

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

```
$ whoami
postgres
$ psql 'host=mySrcHost dbname=postgres user=replication_user
password=mypassword sslmode=require' -c 'SELECT 1;'
```

Remediation:

Review prior sections in this benchmark regarding TLS certificates, replication user, and WAL archiving.

Confirm the file `$PGDATA/standby.signal` is present on the STANDBY host and `$PGDATA/postgresql.auto.conf` contains lines similar to the following:





```
primary_conninfo = 'user=replication_user password=mypassword host=mySrcHost
port=5432 sslmode=require sslcompression=1'
```

References:

1. <https://www.postgresql.org/docs/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:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 <u>Encrypt Sensitive Data in Transit</u> Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).			
v7	14.4 <u>Encrypt All Sensitive Information in Transit</u> Encrypt all sensitive information in transit.			

8 Special Configuration Considerations

The recommendations proposed here try to address some of the less common use cases which may warrant additional configuration guidance/consideration.

8.1 Ensure PostgreSQL subdirectory locations are outside the data cluster (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

The PostgreSQL cluster is organized to carry out specific tasks in subdirectories. For the purposes of performance, reliability, and security some of 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. Other subdirectories contain temporary files created and used during processing. 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_wal`, `pg_log`, and `temp_tablespaces`.

Audit:

Execute the following SQL statement to verify the configuration is correct. Alternatively, inspect the parameter settings in the `postgresql.conf` configuration file.

```
postgres=# SELECT name, setting FROM pg_settings WHERE (name ~ '_directory$'
OR name ~ '_tablespace');
      name      |      setting
-----+-----
allow_in_place_tablespaces | off
data_directory  | /var/lib/pgsql/17/data
default_tablespace | 
log_directory   | log
temp_tablespaces | 
(5 rows)
```

Inspect the file and directory permissions for all returned values. Only superusers and authorized users should have access control rights for these files and directories. If permissions are not highly restrictive, this is a fail.

If `temp_tablespaces` is undefined and `temp_file_limit` has not been set, this is a fail.

Remediation:

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

- Determine appropriate data, log, and tablespace directories and locations based on your organization's security policies. If necessary, relocate all listed directories outside the data cluster.
- If not relocating **temp_tablespaces**, the **temp_file_limit** parameter must be changed from its default value.
- Ensure file permissions are restricted as much as possible, i.e. only superuser read access.
- When directories are relocated to other partitions, ensure that they are of sufficient size to mitigate against excessive space utilization.
- Lastly, change the settings accordingly in the **postgresql.conf** configuration file and restart the database cluster for changes to take effect.

To relocate **temp_tablespaces** to an existing mount point outside the data cluster is accomplished by:

```
postgres=# CREATE TABLESPACE temp_tablespace LOCATION
'/path/to/existing/desired/mount/point';
postgres=# ALTER SYSTEM SET temp_tablespaces = 'temp_tablespace';
postgres=# SELECT pg_reload_conf();
```

Default Value:

The default for **data_directory** is **ConfigDir** and the default for **log_directory** is **log** (based on absolute path of **data_directory**). The defaults for tablespace settings are null, or not set, upon cluster creation.

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-file-locations.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.7 <u>Use Standard Hardening Configuration Templates for Application Infrastructure</u> Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.		●	●
v7	18.11 <u>Use Standard Hardening Configuration Templates for Databases</u> 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 the backup and restore tool, 'pgBackRest', is installed and configured (Automated)

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 a physical backup of the database files and does provide for PITR, but it is constrained by single threading. Both of these methodologies are standard in the PostgreSQL ecosystem and appropriate for particular backup/recovery needs. **pgBackRest** offers another option with much more robust features and flexibility.

pgBackRest is open-source software developed to perform efficient backups on PostgreSQL databases that measure in tens of terabytes and greater. It supports per-file checksums, compression, partial/failed backup resume, high-performance parallel transfer, asynchronous archiving, tablespaces, expiration, full/differential/incremental backups, local/remote operation via SSH or TLS, hard-linking, restore, **backup encryption**, and more. **pgBackRest** is written in C and does not depend on **rsync** or **tar** but instead performs its own deltas which give 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 (17.0-1): 000000010000000000000012 /
0000000100000000000000017

    full backup: 20231012-153106F
      timestamp start/stop: 2023-10-12 15:31:06 / 2023-10-12 15:31:49
      wal start/stop: 0000000100000000000000012 / 000000010000000000000012
      database size: 29.4MB, backup size: 29.4MB
      repository size: 3.4MB, repository backup size: 3.4MB
```

```

diff backup: 20231012-153106F_20231012-173109D
timestamp start/stop: 2023-10-12 17:31:09 / 2023-10-12 17:31:19
wal start/stop: 0000000100000000000000015 / 0000000100000000000000015
database size: 29.4MB, backup size: 2.6MB
repository size: 3.4MB, repository backup size: 346.8KB
backup reference list: 20231012-153106F

incr backup: 20231012-153106F_20231012-183114I
timestamp start/stop: 2023-10-12 18:31:14 / 2023-10-12 18:31:22
wal start/stop: 0000000100000000000000017 / 0000000100000000000000017
database size: 29.4MB, backup size: 8.2KB
repository size: 3.4MB, repository backup size: 519B
backup reference list: 20231012-153106F, 20231012-153106F_20231012-
173109D

```

Audit:

If installed, invoke it without arguments to see the help:

```

# not installed
$ pgbackrest
-bash: pgbackrest: command not found
# installed
$ pgbackrest
pgBackRest 2.53.1 - General help

Usage:
    pgbackrest [options] [command]
<snip>

```

If pgBackRest is not installed, this is (potentially) a fail.

Remediation:

pgBackRest is not installed nor configured for PostgreSQL by default, but instead is maintained as a GitHub project. Fortunately, it is a part of the PGDG repository and can be easily installed:

```

# whoami
root
# subscription-manager repos --enable codeready-builder-for-rhel-9-$(arch)-
rpms && dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-
latest-9.noarch.rpm
<snip>
Installed:
    epel-release-9-8.el9.noarch

Complete!
# dnf -y install pgbackrest
<snip>
Installed:
    libssh2-1.11.0-1.el9.x86_64                                pgbackrest-2.53.1-
1PGDG.rhel9.x86_64

Complete!

```










Once installed, **pgBackRest** must be configured for things like stanza name, backup location, retention policy, logging, etc. Please consult the [configuration guide](#).

If employing **pgBackRest** for your backup/recovery solution, ensure the repository, base backups, and WAL archives are stored on a reliable file system separate from the database server. Further, the external storage system where backups reside should have limited access to only those system administrators as necessary. Finally, as with any backup/recovery solution, stringent testing must be conducted. **A backup is only good if it can be restored successfully.**

References:

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

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	11.2 Perform Automated Backups Perform automated backups of in-scope enterprise assets. Run backups weekly, or more frequently, based on the sensitivity of the data.			
v7	10.1 Ensure Regular Automated Back Ups Ensure that all system data is automatically backed up on regular basis.			
v7	10.2 Perform Complete System Backups Ensure that each of the organization's key systems are backed up as a complete system, through processes such as imaging, to enable the quick recovery of an entire system.			

8.3 Ensure miscellaneous configuration settings are correct (Manual)

Profile Applicability:

- Level 1 - PostgreSQL

Description:

This recommendation covers non-regular, special files, and dynamic libraries.

PostgreSQL permits local logins via the UNIX DOMAIN SOCKET and, for the most part, anyone with a legitimate Unix login account can make the attempt. Limiting PostgreSQL login attempts can be made by relocating the UNIX DOMAIN SOCKET to a subdirectory with restricted permissions.

The creation and implementation of user-defined dynamic libraries is an extraordinary powerful capability. In the hands of an experienced DBA/programmer, it can significantly enhance the power and flexibility of the RDBMS; but new and unexpected behavior can also be assigned to the RDBMS, resulting in a very dangerous environment in what should otherwise be trusted.

Rationale:

Audit:

Execute the following SQL statement to verify the configuration is correct. Alternatively, inspect the parameter settings in the **postgresql.conf** configuration file.

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

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

Remediation:

Follow these steps to remediate the configuration:

- Determine permissions based on your organization's security policies.

- Relocate all files and ensure their permissions are restricted as much as possible, i.e. only superuser read access.
- Ensure all directories where these files are located have restricted permissions such that the superuser can read but not write.
- Lastly, change the settings accordingly in the **postgresql.conf** configuration file and restart the database cluster for changes to take effect.

Default Value:

The **dynamic_library_path** default is **\$libdir** and **unix_socket_directories** default is **/var/run/postgresql, /tmp**. The default for **external_pid_file** and all library parameters are initially null, or not set, upon cluster creation.

References:

1. <https://www.postgresql.org/docs/current/static/runtime-config-file-locations.html>
2. <https://www.postgresql.org/docs/current/static/runtime-config-connection.html>
3. <https://www.postgresql.org/docs/current/static/runtime-config-client.html>

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.7 <u>Use Standard Hardening Configuration Templates for Application Infrastructure</u> Use standard, industry-recommended hardening configuration templates for application infrastructure components. This includes underlying servers, databases, and web servers, and applies to cloud containers, Platform as a Service (PaaS) components, and SaaS components. Do not allow in-house developed software to weaken configuration hardening.		●	●
v7	18.11 <u>Use Standard Hardening Configuration Templates for Databases</u> 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

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
1	Installation and Patches		
1.1	Ensure packages are obtained from authorized repositories (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
1.2	Install only required packages (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
1.3	Ensure systemd Service Files Are Enabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
1.4	Ensure Data Cluster Initialized Successfully (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
1.5	Ensure the Latest Security Patches are Applied (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
1.6	Verify That 'PGPASSWORD' is Not Set in Users' Profiles (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
1.7	Verify That the 'PGPASSWORD' Environment Variable is Not in Use (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
2	Directory and File Permissions		
2.1	Ensure the file permissions mask is correct (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
2.2	Ensure extension directory has appropriate ownership and permissions (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
2.3	Disable PostgreSQL Command History (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
2.4	Ensure Passwords are Not Stored in the service file (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
3	Logging And Auditing		
3.1	PostgreSQL Logging		
3.1.1	Logging Rationale		
3.1.2	Ensure the log destinations are set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
3.1.3	Ensure the logging collector is enabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.4	Ensure the log file destination directory is set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.5	Ensure the filename pattern for log files is set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.6	Ensure the log file permissions are set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.7	Ensure 'log_truncate_on_rotation' is enabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.8	Ensure the maximum log file lifetime is set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.9	Ensure the maximum log file size is set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.10	Ensure the correct syslog facility is selected (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.11	Ensure syslog messages are not suppressed (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.12	Ensure syslog messages are not lost due to size (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.13	Ensure the program name for PostgreSQL syslog messages are correct (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.14	Ensure the correct messages are written to the server log (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.15	Ensure the correct SQL statements generating errors are recorded (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.16	Ensure 'debug_print_parse' is disabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.17	Ensure 'debug_print_rewritten' is disabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.18	Ensure 'debug_print_plan' is disabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
3.1.19	Ensure 'debug_pretty_print' is enabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.20	Ensure 'log_connections' is enabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.21	Ensure 'log_disconnections' is enabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.22	Ensure 'log_error_verbosity' is set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.23	Ensure 'log_hostname' is set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.24	Ensure 'log_line_prefix' is set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.25	Ensure 'log_statement' is set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.1.26	Ensure 'log_timezone' is set correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
3.2	Ensure the PostgreSQL Audit Extension (pgAudit) is enabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
4	User Access and Authorization		
4.1	Ensure Interactive Login is Disabled (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
4.2	Ensure sudo is configured correctly (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
4.3	Ensure excessive administrative privileges are revoked (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
4.4	Lock Out Accounts if Not Currently in Use (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
4.5	Ensure excessive function privileges are revoked (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
4.6	Ensure excessive DML privileges are revoked (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
4.7	Ensure Row Level Security (RLS) is configured correctly (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
4.8	Ensure the set_user extension is installed (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
4.9	Make use of predefined roles (Manual)	<input type="checkbox"/>	<input type="checkbox"/>

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
5	Connection and Login		
5.1	Do Not Specify Passwords in the Command Line (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
5.2	Ensure PostgreSQL is Bound to an IP Address (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
5.3	Ensure login via "local" UNIX Domain Socket is configured correctly (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
5.4	Ensure login via "host" TCP/IP Socket is configured correctly (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
5.5	Ensure per-account connection limits are used (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
5.6	Ensure Password Complexity is configured (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
6	PostgreSQL Settings		
6.1	Understanding attack vectors and runtime parameters (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
6.2	Ensure 'backend' runtime parameters are configured correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
6.3	Ensure 'Postmaster' Runtime Parameters are Configured (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
6.4	Ensure 'SIGHUP' Runtime Parameters are Configured (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
6.5	Ensure 'Superuser' Runtime Parameters are Configured (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
6.6	Ensure 'User' Runtime Parameters are Configured (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
6.7	Ensure FIPS 140-2 OpenSSL Cryptography Is Used (Automated)	<input type="checkbox"/>	<input type="checkbox"/>

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
6.8	Ensure TLS is enabled and configured correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
6.9	Ensure that TLSv1.3, or later, is configured (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
6.10	Ensure Weak SSL/TLS Ciphers Are Disabled (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
6.11	Ensure the pgcrypto extension is installed and configured correctly (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
7	Replication		
7.1	Ensure a replication-only user is created and used for streaming replication (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
7.2	Ensure logging of replication commands is configured (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
7.3	Ensure base backups are configured and functional (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
7.4	Ensure WAL archiving is configured and functional (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
7.5	Ensure streaming replication parameters are configured correctly (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
8	Special Configuration Considerations		
8.1	Ensure PostgreSQL subdirectory locations are outside the data cluster (Manual)	<input type="checkbox"/>	<input type="checkbox"/>
8.2	Ensure the backup and restore tool, 'pgBackRest', is installed and configured (Automated)	<input type="checkbox"/>	<input type="checkbox"/>
8.3	Ensure miscellaneous configuration settings are correct (Manual)	<input type="checkbox"/>	<input type="checkbox"/>

Appendix: Change History

Date	Version	Changes for this version
Feb 16, 2023	1.0.0	Revised the umask values to include the sticky bit for recommendation 2.1.
Feb 16, 2023	1.0.0	Updated recommendation 3.1.24 to offer different guidance for syslog logging.
Feb 16, 2023	1.0.0	Revised procedures for switching to the postgres user to use `sudo -iu` instead of `su -`.
Feb 16, 2023	1.0.0	Added procedures for creating a `roletree` view which must be completed prior to the audit of recommendation 4.6.
Feb 16, 2023	1.0.0	Added audit and remediation procedure to recommendation 4.6 for roles that are superusers and can still login.
Feb 16, 2023	1.0.0	Revised the remediation procedures for recommendation 4.6 to install `set_user` using `dnf` instead of building it from source.
Feb 16, 2023	1.0.0	Added `-u user1` to the remediation procedures for connecting to postgres on recommendation 5.1.
Feb 16, 2023	1.0.0	Revised the remediation procedure for recommendation 6.2, 6.3, 6.5 to combine the two pipes to grep into single pipe.

Date	Version	Changes for this version
Feb 16, 2023	1.0.0	Revised the remediation procedures for recommendation 6.8 to replace setting <code>`ssl_ciphers`</code> with <code>`set_min_ssl_protocol_version`</code> .
Feb 16, 2023	1.0.0	Removed recommendation 8.1 and renumbered following recommendations accordingly.
Feb 16, 2023	1.0.0	Added guidance for configuring <code>`temp_tablespace`</code> and <code>`temp_file_limit`</code> to the audit and remediation procedures for recommendation 8.1, previously 8.2.
Oct 18, 2023	1.0.0	Typo in Remediation (Ticket 17937)
Oct 18, 2023	1.0.0	Typo in remediation (Ticket 17938)
Oct 18, 2023	1.0.0	Wrong reference in the Description (Ticket 19230)
Jan 23, 2024	1.0.0	Please update to reflect database in the collections field when searching CIS benchmarks for PostgreSQL 16 Benchmark v1.0.0 (Ticket 20414)
Mar 12, 2024	1.0.0	(4.5 Ensure excessive function privileges are revoked) failed due to pgAudit extension (Ticket 19065)
Mar 12, 2024	1.0.0	PostgreSQL 3-Logging, Monitoring and Auditing (Ticket 19071)

Date	Version	Changes for this version
Mar 12, 2024	1.0.0	section 5.3 Ensure Password Complexity is configured (Manual) is added (Ticket 19714)
Mar 12, 2024	1.0.0	Add a recommendation for `passwordcheck` (Ticket 18472)
Mar 12, 2024	1.0.0	4.5 Ensure excessive function privileges are revoked incorrect profile (Ticket 15174)