UNCLASSIFIED

PostgreSQL 9.x Security Technical Implementation Guide

Version: 1

Release: 1

20 Jan 2017

XSL Release 1/29/2015 Sort by: STIGID
Description: This Security Technical Implementation Guide is published as a tool to improve the security of Department of Defense (DoD) information systems. The requirements are derived from the National Institute of Standards and Technology (NIST) 800-53 and related documents. Comments or proposed revisions to this document should be sent via email to the following address: disa.stig_spt@mail.mil.

Group ID (Vulid): V-72841
Group Title: SRG-APP-000142-DB-000094
Rule ID: SV-87493r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-000100
Rule Title: PostgreSQL must be configured to prohibit or restrict the use of organization-defined functions, ports, protocols, and/or services, as defined in the PPSM CAL and vulnerability assessments.

Vulnerability Discussion: In order to prevent unauthorized connection of devices, unauthorized transfer of information, or unauthorized tunneling (i.e., embedding of data types within data types), organizations must disable or restrict unused or unnecessary physical and logical ports/protocols/services on information systems.

Applications are capable of providing a wide variety of functions and services. Some of the functions and services provided by default may not be necessary to support essential organizational operations. Additionally, it is sometimes convenient to provide multiple services from a single component (e.g., email and web services); however, doing so increases risk over limiting the services provided by any one component.

To support the requirements and principles of least functionality, the application must support the organizational requirements providing only essential capabilities and limiting the use of ports, protocols, and/or services to only those required, authorized, and approved to conduct official business or to address authorized quality of life issues.

Database Management Systems using ports, protocols, and services deemed unsafe are open to attack through those ports, protocols, and services. This can allow unauthorized access to the database and through the database to other components of the information system.

Check Content:
As the database administrator, run the following SQL:

$ psql -c "SHOW port"

If the currently defined port configuration is deemed prohibited, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To change the listening port of the database, as the database administrator, change the following setting in postgresql.conf:

$ sudo su - postgres
$ vi $PGDATA/postgresql.conf

Change the port parameter to the desired port.

Next, restart the database:

$ sudo su - postgres

# SYSTEMD SERVER ONLY
$ systemctl restart postgresql-9.5

# INITD SERVER ONLY
$ service postgresql-9.5 restart

Note: psql uses the default port 5432 by default. This can be changed by specifying the port with psql or by setting the PGPORT environment variable:
$ psql -p 5432 -c "SHOW port"
$ export PGPORT=5432

CCI: CCI-000382

CCI: CCI-001762

Group ID (Vulid): V-72843
Group Title: SRG-APP-000099-DB-000043
Rule ID: SV-87495r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-000200
Rule Title: PostgreSQL must produce audit records containing sufficient information to establish the outcome (success or failure) of the events.

Vulnerability Discussion: Information system auditing capability is critical for accurate forensic analysis. Without information about the outcome of events, security personnel cannot make an accurate assessment as to whether an attack was successful or if changes were made to the security state of the system.

Event outcomes can include indicators of event success or failure and event-specific results (e.g., the security state of the information system after the event occurred). As such, they also provide a means to measure the impact of an event and help authorized personnel to determine the appropriate response.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

As a database administrator (shown here as "postgres"), create a table, insert a value, alter the table and update the table by running the following SQL:

CREATE TABLE stig_test(id INT);
INSERT INTO stig_test(id) VALUES (0);
ALTER TABLE stig_test ADD COLUMN name text;
UPDATE stig_test SET id = 1 WHERE id = 0;

Next, as a user without access to the stig_test table, run the following SQL:

INSERT INTO stig_test(id) VALUES (1);
ALTER TABLE stig_test DROP COLUMN name;
UPDATE stig_test SET id = 0 WHERE id = 1;

The prior SQL should generate errors:

ERROR: permission denied for relation stig_test
ERROR: must be owner of relation stig_test
ERROR: permission denied for relation stig_test

Now, as the database administrator, drop the test table by running the following SQL:

DROP TABLE stig_test;

Now verify the errors were logged:

$ sudo su - postgres

$ cat ${PGDATA?}/pg_log/<latest_logfile>$PGDATA/
< 2016-02-23 14:51:31.103 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >LOG: AUDIT: SESSION,1,1,DDL,CREATE TABLE,,,CREATE TABLE stig_test(id INT);,<none>
< 2016-02-23 14:51:44.835 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >LOG: AUDIT: SESSION,2,1,WRITE,INSERT,,,INSERT INTO stig_test(id) VALUES (0);,<none>
< 2016-02-23 14:53:25.805 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >LOG: AUDIT: SESSION,3,1,DDL,ALTER TABLE,,,ALTER TABLE stig_test ADD COLUMN name text;,<none>
< 2016-02-23 14:53:54.381 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >LOG: AUDIT: SESSION,4,1,WRITE,UPDATE,,,UPDATE stig_test SET id = 1 WHERE id = 0;,<none>
< 2016-02-23 14:54:20.832 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >ERROR: permission denied for relation stig_test
< 2016-02-23 14:54:20.832 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >STATEMENT: INSERT INTO stig_test(id) VALUES (1);
< 2016-02-23 14:54:41.032 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >ERROR: must be owner of relation stig_test
< 2016-02-23 14:54:41.032 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >STATEMENT: ALTER TABLE stig_test DROP COLUMN name;
< 2016-02-23 14:54:54.378 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >ERROR: permission denied for relation stig_test
< 2016-02-23 14:54:54.378 EDT psql postgres postgres 570bf22a.3af2 2016-04-11 14:51:22 EDT [local] >STATEMENT: UPDATE stig_test SET id = 0 WHERE id = 1;
< 2016-02-23 14:55:23.723 EDT psql postgres postgres 570bf307.3b0a 2016-04-11 14:55:03 EDT [local] >LOG: AUDIT: SESSION,1,1,DDL,DROP TABLE,,,DROP TABLE stig_test;,<none>

If audit records exist without the outcome of the event that occurred, this is a finding.

Fix Text: Using pgaudit PostgreSQL can be configured to audit various facets of PostgreSQL. See supplementary content APPENDIX-B for documentation on installing pgaudit.

All errors, denials and unsuccessful requests are logged if logging is enabled. See supplementary content APPENDIX-C for documentation on enabling logging.

Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

With pgaudit and logging enabled, set the following configuration settings in postgresql.conf, as the database administrator (shown here as "postgres"), to the following:

$ vi ${PGDATA?}/postgresql.conf
pgaudit.log_catalog='on'
pgaudit.log_level='log'
pgaudit.log_parameter='on'
pgaudit.log_statement_once='off'
pgaudit.log='all, -misc'

Next, tune the following logging configurations in postgresql.conf:

$ sudo vi ${PGDATA?}/postgresql.conf
log_line_prefix = '%m %u %d %e : '
log_error_verbosity = default

Last, as the system administrator, restart PostgreSQL:

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

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

CCI: CCI-000134

Group ID (Vulid): V-72845
Group Title: SRG-APP-000456-DB-000390
Rule ID: SV-87497r1_rule
Severity: CAT I
Rule Version (STIG-ID): PGS9-00-000300
Rule Title: Security-relevant software updates to PostgreSQL must be installed within the time period directed by an authoritative source (e.g., IAVM, CTOs, DTMs, and STIGs).

Vulnerability Discussion: Security flaws with software applications, including database management systems, are discovered daily. Vendors are constantly updating and patching their products to address newly discovered security vulnerabilities. Organizations (including any contractor to the organization) are required to promptly install security-relevant software updates (e.g., patches, service packs, and hot fixes). Flaws discovered during security assessments, continuous monitoring, incident response activities, or information system error handling must also be addressed expeditiously.

Organization-defined time periods for updating security-relevant software may vary based on a variety of factors including, for example, the security category of the information system or the criticality of the update (i.e., severity of the vulnerability related to the discovered flaw).

This requirement will apply to software patch management solutions that are used to install patches across the enclave and also to applications themselves that are not part of that patch management solution. For example, many browsers today provide the capability to install their own patch software. Patch criticality, as well as system criticality, will vary. Therefore, the tactical situations regarding the patch management process will also vary. This means that the time period utilized must be a configurable parameter. Time frames for application of security-relevant software updates may be dependent upon the Information Assurance Vulnerability Management (IAVM) process.

The application will be configured to check for and install security-relevant software updates within an identified time period from the availability of the update. The specific time period will be defined by an authoritative source (e.g., IAVM, CTOs, DTMs, and STIGs).

Check Content:
If new packages are available for PostgreSQL, they can be reviewed in the package manager appropriate for the server operating system:

To list the version of installed PostgreSQL using psql:

$ sudo su - postgres
$ psql -–version

To list the current version of software for RPM:

$ rpm -qa | grep postgres

To list the current version of software for APT:

$ apt-cache policy postgres

All versions of PostgreSQL will be listed on:

http://www.postgresql.org/support/versioning/

All security-relevant software updates for PostgreSQL will be listed on:

http://www.postgresql.org/support/security/

If PostgreSQL is not at the latest version, this is a finding.

If PostgreSQL is not at the latest version and the evaluated version has CVEs (IAVAs), then this is a CAT I finding.

Fix Text: Institute and adhere to policies and procedures to ensure that patches are consistently applied to PostgreSQL within the time allowed.

CCI: CCI-002605

Group ID (Vulid): V-72847
Group Title: SRG-APP-000119-DB-000060
Rule ID: SV-87499r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-000400
Rule Title: The audit information produced by PostgreSQL must be protected from unauthorized modification.

Vulnerability Discussion: If audit data were to become compromised, then competent forensic analysis and discovery of the true source of potentially malicious system activity is impossible to achieve.

To ensure the veracity of audit data the information system and/or the application must protect audit information from unauthorized modification.

This requirement can be achieved through multiple methods that will depend upon system architecture and design. Some commonly employed methods include ensuring log files enjoy the proper file system permissions and limiting log data locations.

Applications providing a user interface to audit data will leverage user permissions and roles identifying the user accessing the data and the corresponding rights that the user enjoys in order to make access decisions regarding the modification of audit data.

Audit information includes all information (e.g., audit records, audit settings, and audit reports) needed to successfully audit information system activity.

Modification of database audit data could mask the theft of, or the unauthorized modification of, sensitive data stored in the database.

Check Content:
Review locations of audit logs, both internal to the database and database audit logs located at the operating system level.

Verify there are appropriate controls and permissions to protect the audit information from unauthorized modification.

Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

stderr Logging

If the PostgreSQL server is configured to use stderr for logging, the logs will be owned by the database owner (usually postgres user) with a default permissions level of 0600. The permissions can be configured in postgresql.conf.

To check the permissions for log files in postgresql.conf, as the database owner (shown here as "postgres"), run the following command:

$ sudo su - postgres
$ grep "log_file_mode" ${PGDATA?}/postgresql.conf

If the permissions are not 0600, this is a finding.

Next, navigate to where the logs are stored. This can be found by running the following command against postgresql.conf as the database owner (shown here as "postgres"):

$ sudo su - postgres
$ grep "log_directory" ${PGDATA?}/postgresql.conf

With the log directory identified, as the database owner (shown here as "postgres"), list the permissions of the logs:

$ sudo su - postgres
$ ls -la ${PGDATA?}/pg_log

If logs are not owned by the database owner (shown here as "postgres") and are not the same permissions as configured in postgresql.conf, this is a finding.

syslog Logging

If the PostgreSQL server is configured to use syslog for logging, consult the organizations syslog setting for permissions and ownership of logs.

Fix Text: To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

stderr Logging

With stderr logging enabled, as the database owner (shown here as "postgres"), set the following parameter in postgresql.conf:

$ vi ${PGDATA?}/postgresql.conf
log_file_mode = 0600

To change the owner and permissions of the log files, run the following:

$ chown postgres:postgres ${PGDATA?}/<log directory name>
$ chmod 0700 ${PGDATA?}/<log directory name>
$ chmod 600 ${PGDATA?}/<log directory name>/*.log

syslog Logging

If PostgreSQL is configured to use syslog for logging, the log files must be configured to be owned by root with 0600 permissions.

$ chown root:root <log directory name>/<log_filename>
$ chmod 0700 <log directory name>
$ chmod 0600 <log directory name>/*.log

CCI: CCI-000163

Group ID (Vulid): V-72849
Group Title: SRG-APP-000023-DB-000001
Rule ID: SV-87501r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-000500
Rule Title: PostgreSQL must integrate with an organization-level authentication/access mechanism providing account management and automation for all users, groups, roles, and any other principals.

Vulnerability Discussion: Enterprise environments make account management for applications and databases challenging and complex. A manual process for account management functions adds the risk of a potential oversight or other error. Managing accounts for the same person in multiple places is inefficient and prone to problems with consistency and synchronization.

A comprehensive application account management process that includes automation helps to ensure that accounts designated as requiring attention are consistently and promptly addressed.

Examples include, but are not limited to, using automation to take action on multiple accounts designated as inactive, suspended, or terminated, or by disabling accounts located in non-centralized account stores, such as multiple servers. Account management functions can also include: assignment of group or role membership; identifying account type; specifying user access authorizations (i.e., privileges); account removal, update, or termination; and administrative alerts. The use of automated mechanisms can include, for example: using email or text messaging to notify account managers when users

are terminated or transferred; using the information system to monitor account usage; and using automated telephone notification to report atypical system account usage.

PostgreSQL must be configured to automatically utilize organization-level account management functions, and these functions must immediately enforce the organization's current account policy.

Automation may be comprised of differing technologies that when placed together contain an overall mechanism supporting an organization's automated account management requirements.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

If all accounts are authenticated by the organization-level authentication/access mechanism, such as LDAP or Kerberos and not by PostgreSQL, this is not a finding.

As the database administrator (shown here as "postgres"), review pg_hba.conf authentication file settings:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_hba.conf

All records must use an auth-method of gss, sspi, or ldap. For details on the specifics of these authentication methods see: http://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html

If there are any records with a different auth-method than gss, sspi, or ldap, review the system documentation for justification and approval of these records.

If there are any records with a different auth-method than gss, sspi, or ldap, that are not documented and approved, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Integrate PostgreSQL security with an organization-level authentication/access mechanism providing account management for all users, groups, roles, and any other principals.

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

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

For each PostgreSQL-managed account that is not documented and approved, either transfer it to management by the external mechanism, or document the need for it and obtain approval, as appropriate.

CCI: CCI-000015

Group ID (Vulid): V-72851
Group Title: SRG-APP-000266-DB-000162
Rule ID: SV-87503r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-000600
Rule Title: PostgreSQL must provide non-privileged users with error messages that provide information necessary for corrective actions without revealing information that could be exploited by adversaries.

Vulnerability Discussion: Any PostgreSQL or associated application providing too much information in error messages on the screen or printout risks compromising the data and security of the system. The structure and content of error messages need to be carefully considered by the organization and development team.

Databases can inadvertently provide a wealth of information to an attacker through improperly handled error messages. In addition to sensitive business or personal information, database errors can provide host names, IP addresses, user names, and other system information not required for troubleshooting but very useful to someone targeting the system.

Carefully consider the structure/content of error messages. The extent to which information systems are able to identify and handle error conditions is guided by organizational policy and operational requirements. Information that could be exploited by adversaries includes, for example, logon attempts with passwords entered by mistake as the username, mission/business information that can be derived from (if not stated explicitly by) information recorded, and personal information, such as account numbers, social security numbers, and credit card numbers.

Check Content:
As the database administrator, run the following SQL:

SELECT current_setting('client_min_messages');

If client_min_messages is not set to error, this is a finding.

Fix Text: As the database administrator, edit postgresql.conf:

$ sudo su - postgres
$ vi $PGDATA/postgresql.conf

Change the client_min_messages parameter to be error:

client_min_messages = error

Now reload the server with the new configuration (this just reloads settings currently in memory, will not cause an interruption):

$ sudo su - postgres

# SYSTEMD SERVER ONLY
$ systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ service postgresql-9.5 reload

CCI: CCI-001312

Group ID (Vulid): V-72853
Group Title: SRG-APP-000133-DB-000179
Rule ID: SV-87505r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-000700
Rule Title: Privileges to change PostgreSQL software modules must be limited.

Vulnerability Discussion: If the system were to allow any user to make changes to software libraries, those changes might be implemented without undergoing the appropriate testing and approvals that are part of a robust change management process.

Accordingly, only qualified and authorized individuals must be allowed to obtain access to information system components for purposes of initiating changes, including upgrades and modifications.

Unmanaged changes that occur to the database software libraries or configuration can lead to unauthorized or compromised installations.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

As the database administrator (shown here as "postgres"), check the permissions of configuration files for the database:

$ sudo su - postgres
$ ls -la ${PGDATA?}

If any files are not owned by the database owner or have permissions allowing others to modify (write) configuration files, this is a finding.

As the server administrator, check the permissions on the shared libraries for PostgreSQL:

$ sudo ls -la /usr/pgsql-9.5
$ sudo ls -la /usr/pgsql-9.5/bin
$ sudo ls -la /usr/pgsql-9.5/include
$ sudo ls -la /usr/pgsql-9.5/lib
$ sudo ls -la /usr/pgsql-9.5/share

If any files are not owned by root or have permissions allowing others to modify (write) configuration files, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

As the database administrator (shown here as "postgres"), change the ownership and permissions of configuration files in PGDATA:

$ sudo su - postgres
$ chown postgres:postgres ${PGDATA?}/postgresql.conf
$ chmod 0600 ${PGDATA?}/postgresql.conf

As the server administrator, change the ownership and permissions of shared objects in /usr/pgsql-9.5/*.so

$ sudo chown root:root /usr/pgsql-9.5/lib/*.so
$ sudo chmod 0755 /usr/pgsql-9.5/lib/*.so

As the service administrator, change the ownership and permissions of executables in /usr/pgsql-9.5/bin:

$ sudo chown root:root /usr/pgsql-9.5/bin/*
$ sudo chmod 0755 /usr/pgsql-9.5/bin/*

CCI: CCI-001499

Group ID (Vulid): V-72855
Group Title: SRG-APP-000133-DB-000179
Rule ID: SV-87507r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-000710
Rule Title: PostgreSQL must limit privileges to change functions and triggers, and links to software external to PostgreSQL.

Vulnerability Discussion: If the system were to allow any user to make changes to software libraries, those changes might be implemented without undergoing the appropriate testing and approvals that are part of a robust change management process.

Accordingly, only qualified and authorized individuals must be allowed to obtain access to information system components for purposes of initiating changes, including upgrades and modifications.

Unmanaged changes that occur to the database code can lead to unauthorized or compromised installations.

Check Content:
Only owners of objects can change them. To view all functions, triggers, and trigger procedures, their ownership and source, as the database administrator (shown here as "postgres") run the following SQL:

$ sudo su - postgres
$ psql -x -c "\df+"

Only the OS database owner user (shown here as "postgres") or a PostgreSQL superuser can change links to external software. As the database administrator (shown here as "postgres"), check the permissions of configuration files for the database:

$ sudo su - postgres
$ ls -la ${PGDATA?}

If any files are not owned by the database owner or have permissions allowing others to modify (write) configuration files, this is a finding.

Fix Text: To change ownership of an object, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su – postgres
$ psql -c "ALTER FUNCTION function_name OWNER TO new_role_name"

To change ownership of postgresql.conf, as the database administrator (shown here as "postgres"), run the following commands:

$ sudo su - postgres
$ chown postgres:postgres ${PGDATA?}/postgresql.conf
$ chmod 0600 ${PGDATA?}/postgresql.conf

To remove superuser from a role, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "ALTER ROLE rolename WITH NOSUPERUSER"

CCI: CCI-001499

Group ID (Vulid): V-72857
Group Title: SRG-APP-000172-DB-000075
Rule ID: SV-87509r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-000800
Rule Title: If passwords are used for authentication, PostgreSQL must transmit only encrypted representations of passwords.

Vulnerability Discussion: The DoD standard for authentication is DoD-approved PKI certificates.

Authentication based on User ID and Password may be used only when it is not possible to employ a PKI certificate, and requires AO approval.

In such cases, passwords need to be protected at all times, and encryption is the standard method for protecting passwords during transmission.

PostgreSQL passwords sent in clear text format across the network are vulnerable to discovery by unauthorized users. Disclosure of passwords may easily lead to unauthorized access to the database.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

As the database administrator (shown here as "postgres"), review the authentication entries in pg_hba.conf:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_hba.conf

If any entries use the auth_method (last column in records) "password", this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

As the database administrator (shown here as "postgres"), edit pg_hba.conf authentication file and change all entries of "password" to "md5":

$ sudo su - postgres
$ vi ${PGDATA?}/pg_hba.conf
host all all .example.com md5

CCI: CCI-000197

Group ID (Vulid): V-72859
Group Title: SRG-APP-000033-DB-000084
Rule ID: SV-87511r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-000900
Rule Title: PostgreSQL must enforce approved authorizations for logical access to information and system resources in accordance with applicable access control policies.

Vulnerability Discussion: Authentication with a DoD-approved PKI certificate does not necessarily imply authorization to access PostgreSQL. To mitigate the risk of unauthorized access to sensitive information by entities that have been issued certificates by DoD-approved PKIs, all DoD systems, including databases, must be properly configured to implement access control policies.

Successful authentication must not automatically give an entity access to an asset or security boundary. Authorization procedures and controls must be implemented to ensure each authenticated entity also has a validated and current authorization. Authorization is the process of determining whether an entity, once authenticated, is permitted to access a specific asset. Information systems use access control policies and enforcement mechanisms to implement this requirement.

Access control policies include identity-based policies, role-based policies, and attribute-based policies. Access enforcement mechanisms include access control lists, access control matrices, and cryptography. These policies and mechanisms must be employed by the application to control access between users (or processes acting on behalf of users) and objects (e.g., devices, files, records, processes, programs, and domains) in the information system.

This requirement is applicable to access control enforcement applications, a category that includes database management systems. If PostgreSQL does not follow applicable policy when approving access, it may be in conflict with networks or other applications in the information system. This may result in users either gaining or being denied access inappropriately and in conflict with applicable policy.

Check Content:
From the system security plan or equivalent documentation, determine the appropriate permissions on database objects for each kind (group role) of user. If this documentation is missing, this is a finding.

First, as the database administrator (shown here as "postgres"), check the privileges of all roles in the database by running the following SQL:

$ sudo su - postgres
$ psql -c '\du'

Review all roles and their associated privileges. If any roles' privileges exceed those documented, this is a finding.

Next, as the database administrator (shown here as "postgres"), check the configured privileges for tables and columns by running the following SQL:

$ sudo su - postgres
$ psql -c '\dp'

Review all access privileges and column access privileges list. If any roles' privileges exceed those documented, this is a finding.

Next, as the database administrator (shown here as "postgres"), check the configured authentication settings in pg_hba.conf:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_hba.conf

Review all entries and their associated authentication methods. If any entries do not have their documented authentication requirements, this is a finding.

Fix Text: Create and/or maintain documentation of each group role's appropriate permissions on database objects.

Implement these permissions in the database, and remove any permissions that exceed those documented.

- - - - -

The following are examples of how to use role privileges in PostgreSQL to enforce access controls. For a complete list of privileges, see the official

documentation: https://www.postgresql.org/docs/current/static/sql-createrole.html

Roles Example 1

The following example demonstrates how to create an admin role with CREATEDB and CREATEROLE privileges.

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

```
$ sudo su - postgres
$ psql -c "CREATE ROLE admin WITH CREATEDB CREATEROLE"
```

Roles Example 2

The following example demonstrates how to create a role with a password that expires and makes the role a member of the "admin" group.

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

```
$ sudo su - postgres
$ psql -c "CREATE ROLE joe LOGIN ENCRYPTED PASSWORD 'stig2016!' VALID UNTIL '2016-09-20' IN ROLE admin"
```

Roles Example 3

The following demonstrates how to revoke privileges from a role using REVOKE.

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

```
$ sudo su - postgres
$ psql -c "REVOKE admin FROM joe"
```

Roles Example 4

The following demonstrates how to alter privileges in a role using ALTER.

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

```
$ sudo su - postgres
$ psql -c "ALTER ROLE joe NOLOGIN"
```

The following are examples of how to use grant privileges in PostgreSQL to enforce access controls on objects. For a complete list of privileges, see the official documentation: https://www.postgresql.org/docs/current/static/sql-grant.html

Grant Example 1

The following example demonstrates how to grant INSERT on a table to a role.

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

```
$ sudo su - postgres
$ psql -c "GRANT SELECT ON stig_test TO joe"
```

Grant Example 2

The following example demonstrates how to grant ALL PRIVILEGES on a table to a role.

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

```
$ sudo su - postgres
$ psql -c "GRANT ALL PRIVILEGES ON stig_test TO joe"
```

Grant Example 3

The following example demonstrates how to grant a role to a role.

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

```
$ sudo su - postgres
$ psql -c "GRANT admin TO joe"
```

Revoke Example 1

The following example demonstrates how to revoke access from a role.

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

```
$ sudo su - postgres
$ psql -c "REVOKE admin FROM joe"
```

To change authentication requirements for the database, as the database administrator (shown here as "postgres"), edit pg_hba.conf:

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

Edit authentication requirements to the organizational requirements. See the official documentation for the complete list of options for authentication: http://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html

After changes to pg_hba.conf, reload the server:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000213

Group ID (Vulid): V-72861
Group Title: SRG-APP-000314-DB-000310
Rule ID: SV-87513r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-001100
Rule Title: PostgreSQL must associate organization-defined types of security labels having organization-defined security label values with information in transmission.

Vulnerability Discussion: Without the association of security labels to information, there is no basis for PostgreSQL to make security-related access-control decisions.

Security labels are abstractions representing the basic properties or characteristics of an entity (e.g., subjects and objects) with respect to safeguarding information.

These labels are typically associated with internal data structures (e.g., tables, rows) within the database and are used to enable the implementation of access control and flow control policies, reflect special dissemination, handling or distribution instructions, or support other aspects of the information security policy.

One example includes marking data as classified or FOUO. These security labels may be assigned manually or during data processing, but, either way, it is imperative these assignments are maintained while the data is in storage. If the security labels are lost when the data is stored, there is the risk of a data compromise.

Check Content:
If security labeling is not required, this is not a finding.

First, as the database administrator (shown here as "postgres"), run the following SQL against each table that requires security labels:

$ sudo su - postgres
$ psql -c "\d+ <schema_name>.<table_name>"

If security labeling is required and the results of the SQL above do not show a policy attached to the table, this is a finding.

If security labeling is required and not implemented according to the system documentation, such as SSP, this is a finding.

If security labeling requirements have been specified, but the security labeling is not implemented or does not reliably maintain labels on information in storage, this is a finding.

Fix Text: 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 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).

RLS policies can be very different depending on their use case. For one example of using RLS for Security Labels, see supplementary content APPENDIX-D.

CCI: CCI-002264

Group ID (Vulid): V-72863
Group Title: SRG-APP-000001-DB-000031
Rule ID: SV-87515r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-001200
Rule Title: PostgreSQL must limit the number of concurrent sessions to an organization-defined number per user for all accounts and/or account types.

Vulnerability Discussion: Database management includes the ability to control the number of users and user sessions utilizing PostgreSQL. Unlimited

concurrent connections to PostgreSQL could allow a successful Denial of Service (DoS) attack by exhausting connection resources; and a system can also fail or be degraded by an overload of legitimate users. Limiting the number of concurrent sessions per user is helpful in reducing these risks.

This requirement addresses concurrent session control for a single account. It does not address concurrent sessions by a single user via multiple system accounts; and it does not deal with the total number of sessions across all accounts.

The capability to limit the number of concurrent sessions per user must be configured in or added to PostgreSQL (for example, by use of a logon trigger), when this is technically feasible. Note that it is not sufficient to limit sessions via a web server or application server alone, because legitimate users and adversaries can potentially connect to PostgreSQL by other means.

The organization will need to define the maximum number of concurrent sessions by account type, by account, or a combination thereof. In deciding on the appropriate number, it is important to consider the work requirements of the various types of users. For example, 2 might be an acceptable limit for general users accessing the database via an application; but 10 might be too few for a database administrator using a database management GUI tool, where each query tab and navigation pane may count as a separate session.

(Sessions may also be referred to as connections or logons, which for the purposes of this requirement are synonyms.)

Check Content:
To check the total amount of connections allowed by the database, as the database administrator, run the following SQL:

$ sudo su - postgres
$ psql -c "SHOW max_connections"

If the total amount of connections is greater than documented by an organization, this is a finding.

To check the amount of connections allowed for each role, as the database administrator, run the following SQL:

$ sudo su - postgres
$ psql -c "SELECT rolname, rolconnlimit from pg_authid"

If any roles have more connections configured than documented, this is a finding. A value of -1 indicates Unlimited, and is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To configure the maximum amount of connections allowed to the database, as the database administrator (shown here as "postgres") change the following in postgresql.conf (the value 10 is an example; set the value to suit local conditions):

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
max_connections = 10

Next, restart the database:

# SYSTEMD SERVER ONLY
$ sudo systemctl restart postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 restart

To limit the amount of connections allowed by a specific role, as the database administrator, run the following SQL:

$ psql -c "ALTER ROLE <rolname> CONNECTION LIMIT 1";

CCI: CCI-000054

Group ID (Vulid): V-72865
Group Title: SRG-APP-000133-DB-000362
Rule ID: SV-87517r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-001300
Rule Title: The role(s)/group(s) used to modify database structure (including but not necessarily limited to tables, indexes, storage, etc.) and logic modules (functions, trigger procedures, links to software external to PostgreSQL, etc.) must be restricted to authorized users.

Vulnerability Discussion: If PostgreSQL were to allow any user to make changes to database structure or logic, those changes might be implemented without undergoing the appropriate testing and approvals that are part of a robust change management process.

Accordingly, only qualified and authorized individuals must be allowed to obtain access to information system components for purposes of initiating changes, including upgrades and modifications.

Unmanaged changes that occur to the database software libraries or configuration can lead to unauthorized or compromised installations.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring

PGDATA.

As the database administrator (shown here as "postgres"), list all users and their permissions by running the following SQL:

$ sudo su - postgres
$ psql -c "\dp *.*"

Verify that all objects have the correct privileges. If they do not, this is a finding.

Next, as the database administrator (shown here as "postgres"), verify the permissions of the database directory on the filesystem:

$ ls -la ${PGDATA?}

If permissions of the database directory are not limited to an authorized user account, this is a finding.

Fix Text: As the database administrator, revoke any permissions from a role that are deemed unnecessary by running the following SQL:

ALTER ROLE bob NOCREATEDB;
ALTER ROLE bob NOCREATEROLE;
ALTER ROLE bob NOSUPERUSER;
ALTER ROLE bob NOINHERIT;
REVOKE SELECT ON some_function FROM bob;

CCI: CCI-001499

Group ID (Vulid): V-72867
Group Title: SRG-APP-000180-DB-000115
Rule ID: SV-87519r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-001400
Rule Title: PostgreSQL must uniquely identify and authenticate non-organizational users (or processes acting on behalf of non-organizational users).

Vulnerability Discussion: Non-organizational users include all information system users other than organizational users, which includes organizational employees or individuals the organization deems to have equivalent status of employees (e.g., contractors, guest researchers, individuals from allied nations).

Non-organizational users must be uniquely identified and authenticated for all accesses other than those accesses explicitly identified and documented by the organization when related to the use of anonymous access, such as accessing a web server.

Accordingly, a risk assessment is used in determining the authentication needs of the organization.

Scalability, practicality, and security are simultaneously considered in balancing the need to ensure ease of use for access to federal information and information systems with the need to protect and adequately mitigate risk to organizational operations, organizational assets, individuals, other organizations, and the Nation.

Check Content:
PostgreSQL uniquely identifies and authenticates PostgreSQL users through the use of DBMS roles.

To list all roles in the database, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "\du"

If users are not uniquely identified as per organizational documentation, this is a finding.

Fix Text: To drop a role, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "DROP ROLE <role_to_drop>"

To create a role, as the database administrator, run the following SQL:

$ sudo su - postgres
$ psql -c "CREATE ROLE <role name> LOGIN"

For the complete list of permissions allowed by roles, see the official documentation: https://www.postgresql.org/docs/current/static/sql-createrole.html

CCI: CCI-000804

Group ID (Vulid): V-72869
Group Title: SRG-APP-000311-DB-000308

Rule ID: SV-87521r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-001700
Rule Title: PostgreSQL must associate organization-defined types of security labels having organization-defined security label values with information in storage.

Vulnerability Discussion: Without the association of security labels to information, there is no basis for PostgreSQL to make security-related access-control decisions.

Security labels are abstractions representing the basic properties or characteristics of an entity (e.g., subjects and objects) with respect to safeguarding information.

These labels are typically associated with internal data structures (e.g., tables, rows) within the database and are used to enable the implementation of access control and flow control policies, reflect special dissemination, handling or distribution instructions, or support other aspects of the information security policy.

One example includes marking data as classified or FOUO. These security labels may be assigned manually or during data processing, but, either way, it is imperative these assignments are maintained while the data is in storage. If the security labels are lost when the data is stored, there is the risk of a data compromise.

Check Content:
If security labeling is not required, this is not a finding.

First, as the database administrator (shown here as "postgres"), run the following SQL against each table that requires security labels:

$ sudo su - postgres
$ psql -c "\d+ <schema_name>.<table_name>"

If security labeling is required and the results of the SQL above do not show a policy attached to the table, this is a finding.

If security labeling is required and not implemented according to the system documentation, such as SSP, this is a finding.

If security labeling requirements have been specified, but the security labeling is not implemented or does not reliably maintain labels on information in storage, this is a finding.

Fix Text: 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 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).

RLS policies can be very different depending on their use case. For one example of using RLS for Security Labels, see supplementary content APPENDIX-D.

CCI: CCI-002262

Group ID (Vulid): V-72871
Group Title: SRG-APP-000251-DB-000160
Rule ID: SV-87523r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-001800
Rule Title: PostgreSQL must check the validity of all data inputs except those specifically identified by the organization.

Vulnerability Discussion: Invalid user input occurs when a user inserts data or characters into an application's data entry fields and the application is unprepared to process that data. This results in unanticipated application behavior, potentially leading to an application or information system compromise. Invalid user input is one of the primary methods employed when attempting to compromise an application.

With respect to database management systems, one class of threat is known as SQL Injection, or more generally, code injection. It takes advantage of the dynamic execution capabilities of various programming languages, including dialects of SQL. Potentially, the attacker can gain unauthorized access to data, including security settings, and severely corrupt or destroy the database.

Even when no such hijacking takes place, invalid input that gets recorded in the database, whether accidental or malicious, reduces the reliability and usability of the system. Available protections include data types, referential constraints, uniqueness constraints, range checking, and application-specific logic. Application-specific logic can be implemented within the database in stored procedures and triggers, where appropriate.

This calls for inspection of application source code, which will require collaboration with the application developers. It is recognized that in many cases, the database administrator (DBA) is organizationally separate from the application developers, and may have limited, if any, access to source code. Nevertheless, protections of this type are so important to the secure operation of databases that they must not be ignored. At a minimum, the DBA must attempt to obtain assurances from the development organization that this issue has been addressed, and must document what has been discovered.

Check Content:
Review PostgreSQL code (trigger procedures, functions), application code, settings, column and field definitions, and constraints to determine whether the database is protected against invalid input.

If code exists that allows invalid data to be acted upon or input into the database, this is a finding.

If column/field definitions do not exist in the database, this is a finding.

If columns/fields do not contain constraints and validity checking where required, this is a finding.

Where a column/field is noted in the system documentation as necessarily free-form, even though its name and context suggest that it should be strongly typed and constrained, the absence of these protections is not a finding.

Where a column/field is clearly identified by name, caption or context as Notes, Comments, Description, Text, etc., the absence of these protections is not a finding.

Check application code that interacts with PostgreSQL for the use of prepared statements. If prepared statements are not used, this is a finding.

Fix Text: Modify database code to properly validate data before it is put into the database or acted upon by the database.

Modify the database to contain constraints and validity checking on database columns and tables that require them for data integrity.

Use prepared statements when taking user input.

Do not allow general users direct console access to PostgreSQL.

CCI: CCI-001310

Group ID (Vulid): V-72873
Group Title: SRG-APP-000251-DB-000391
Rule ID: SV-87525r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-001900
Rule Title: PostgreSQL and associated applications must reserve the use of dynamic code execution for situations that require it.

Vulnerability Discussion: With respect to database management systems, one class of threat is known as SQL Injection, or more generally, code injection. It takes advantage of the dynamic execution capabilities of various programming languages, including dialects of SQL. In such cases, the attacker deduces the manner in which SQL statements are being processed, either from inside knowledge or by observing system behavior in response to invalid inputs. When the attacker identifies scenarios where SQL queries are being assembled by application code (which may be within the database or separate from it) and executed dynamically, the attacker is then able to craft input strings that subvert the intent of the query. Potentially, the attacker can gain unauthorized access to data, including security settings, and severely corrupt or destroy the database.

The principal protection against code injection is not to use dynamic execution except where it provides necessary functionality that cannot be utilized otherwise. Use strongly typed data items rather than general-purpose strings as input parameters to task-specific, pre-compiled stored procedures and functions (and triggers).

This calls for inspection of application source code, which will require collaboration with the application developers. It is recognized that in many cases, the database administrator (DBA) is organizationally separate from the application developers, and may have limited, if any, access to source code. Nevertheless, protections of this type are so important to the secure operation of databases that they must not be ignored. At a minimum, the DBA must attempt to obtain assurances from the development organization that this issue has been addressed, and must document what has been discovered.

Check Content:
Review PostgreSQL source code (trigger procedures, functions) and application source code, to identify cases of dynamic code execution. Any user input should be handled through prepared statements.

If dynamic code execution is employed in circumstances where the objective could practically be satisfied by static execution with strongly typed parameters, this is a finding.

Fix Text: Where dynamic code execution is employed in circumstances where the objective could practically be satisfied by static execution with strongly typed parameters, modify the code to do so.

CCI: CCI-001310

Group ID (Vulid): V-72875
Group Title: SRG-APP-000251-DB-000392
Rule ID: SV-87527r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-002000
Rule Title: PostgreSQL and associated applications, when making use of dynamic code execution, must scan input data for invalid values that may indicate a code injection attack.

Vulnerability Discussion: With respect to database management systems, one class of threat is known as SQL Injection, or more generally, code injection. It takes advantage of the dynamic execution capabilities of various programming languages, including dialects of SQL. In such cases, the attacker deduces the manner in which SQL statements are being processed, either from inside knowledge or by observing system behavior in response to invalid inputs. When the attacker identifies scenarios where SQL queries are being assembled by application code (which may be within the database or separate from it) and executed dynamically, the attacker is then able to craft input strings that subvert the intent of the query. Potentially, the attacker can

gain unauthorized access to data, including security settings, and severely corrupt or destroy the database.

The principal protection against code injection is not to use dynamic execution except where it provides necessary functionality that cannot be utilized otherwise. Use strongly typed data items rather than general-purpose strings as input parameters to task-specific, pre-compiled stored procedures and functions (and triggers).

When dynamic execution is necessary, ways to mitigate the risk include the following, which should be implemented both in the on-screen application and at the database level, in the stored procedures:
-- Allow strings as input only when necessary.
-- Rely on data typing to validate numbers, dates, etc. Do not accept invalid values. If substituting other values for them, think carefully about whether this could be subverted.
-- Limit the size of input strings to what is truly necessary.
-- If single quotes/apostrophes, double quotes, semicolons, equals signs, angle brackets, or square brackets will never be valid as input, reject them.
-- If comment markers will never be valid as input, reject them. In SQL, these are -- or /* */
-- If HTML and XML tags, entities, comments, etc., will never be valid, reject them.
-- If wildcards are present, reject them unless truly necessary. In SQL these are the underscore and the percentage sign, and the word ESCAPE is also a clue that wildcards are in use.
-- If SQL key words, such as SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, ESCAPE, UNION, GRANT, REVOKE, DENY, MODIFY will never be valid, reject them. Use case-insensitive comparisons when searching for these. Bear in mind that some of these words, particularly Grant (as a person's name), could also be valid input.
-- If there are range limits on the values that may be entered, enforce those limits.
-- Institute procedures for inspection of programs for correct use of dynamic coding, by a party other than the developer.
-- Conduct rigorous testing of program modules that use dynamic coding, searching for ways to subvert the intended use.
-- Record the inspection and testing in the system documentation.
-- Bear in mind that all this applies not only to screen input, but also to the values in an incoming message to a web service or to a stored procedure called by a software component that has not itself been hardened in these ways. Not only can the caller be subject to such vulnerabilities; it may itself be the attacker.

Check Content:
Review PostgreSQL source code (trigger procedures, functions) and application source code to identify cases of dynamic code execution.

If dynamic code execution is employed without protective measures against code injection, this is a finding.

Fix Text: Where dynamic code execution is used, modify the code to implement protections against code injection (IE: prepared statements).

CCI: CCI-001310

Group ID (Vulid): V-72877
Group Title: SRG-APP-000357-DB-000316
Rule ID: SV-87529r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-002100
Rule Title: PostgreSQL must allocate audit record storage capacity in accordance with organization-defined audit record storage requirements.

Vulnerability Discussion: In order to ensure sufficient storage capacity for the audit logs, PostgreSQL must be able to allocate audit record storage capacity. Although another requirement (SRG-APP-000515-DB-000318) mandates that audit data be off-loaded to a centralized log management system, it remains necessary to provide space on the database server to serve as a buffer against outages and capacity limits of the off-loading mechanism.

The task of allocating audit record storage capacity is usually performed during initial installation of PostgreSQL and is closely associated with the DBA and system administrator roles. The DBA or system administrator will usually coordinate the allocation of physical drive space with the application owner/installer and the application will prompt the installer to provide the capacity information, the physical location of the disk, or both.

In determining the capacity requirements, consider such factors as: total number of users; expected number of concurrent users during busy periods; number and type of events being monitored; types and amounts of data being captured; the frequency/speed with which audit records are off-loaded to the central log management system; and any limitations that exist on PostgreSQL's ability to reuse the space formerly occupied by off-loaded records.

Check Content:
Investigate whether there have been any incidents where PostgreSQL ran out of audit log space since the last time the space was allocated or other corrective measures were taken.

If there have been incidents where PostgreSQL ran out of audit log space, this is a finding.

Fix Text: Allocate sufficient audit file/table space to support peak demand.

CCI: CCI-001849

Group ID (Vulid): V-72883
Group Title: SRG-APP-000328-DB-000301
Rule ID: SV-87535r1_rule
Severity: CAT II

Rule Version (STIG-ID): PGS9-00-002200
Rule Title: PostgreSQL must enforce discretionary access control policies, as defined by the data owner, over defined subjects and objects.

Vulnerability Discussion: Discretionary Access Control (DAC) is based on the notion that individual users are "owners" of objects and therefore have discretion over who should be authorized to access the object and in which mode (e.g., read or write). Ownership is usually acquired as a consequence of creating the object or via specified ownership assignment. DAC allows the owner to determine who will have access to objects they control. An example of DAC includes user-controlled table permissions.

When discretionary access control policies are implemented, subjects are not constrained with regard to what actions they can take with information for which they have already been granted access. Thus, subjects that have been granted access to information are not prevented from passing (i.e., the subjects have the discretion to pass) the information to other subjects or objects.

A subject that is constrained in its operation by Mandatory Access Control policies is still able to operate under the less rigorous constraints of this requirement. Thus, while Mandatory Access Control imposes constraints preventing a subject from passing information to another subject operating at a different sensitivity level, this requirement permits the subject to pass the information to any subject at the same sensitivity level.

The policy is bounded by the information system boundary. Once the information is passed outside of the control of the information system, additional means may be required to ensure the constraints remain in effect. While the older, more traditional definitions of discretionary access control require identity-based access control, that limitation is not required for this use of discretionary access control.

Check Content:
Review system documentation to identify the required discretionary access control (DAC).

Review the security configuration of the database and PostgreSQL. If applicable, review the security configuration of the application(s) using the database.

If the discretionary access control defined in the documentation is not implemented in the security configuration, this is a finding.

If any database objects are found to be owned by users not authorized to own database objects, this is a finding.

To check the ownership of objects in the database, as the database administrator, run the following:

```
$ sudo su - postgres
$ psql -c "\dn *.*"
$ psql -c "\dt *.*"
$ psql -c "\ds *.*"
$ psql -c "\dv *.*"
$ psql -c "\df+ *.*"
```

If any role is given privileges to objects it should not have, this is a finding.

Fix Text: Implement the organization's DAC policy in the security configuration of the database and PostgreSQL, and, if applicable, the security configuration of the application(s) using the database.

To GRANT privileges to roles, as the database administrator (shown here as "postgres"), run statements like the following examples:

```
$ sudo su - postgres
$ psql -c "CREATE SCHEMA test"
$ psql -c "GRANT CREATE ON SCHEMA test TO bob"
$ psql -c "CREATE TABLE test.test_table(id INT)"
$ psql -c "GRANT SELECT ON TABLE test.test_table TO bob"
```

To REVOKE privileges to roles, as the database administrator (shown here as "postgres"), run statements like the following examples:

```
$ psql -c "REVOKE SELECT ON TABLE test.test_table FROM bob"
$ psql -c "REVOKE CREATE ON SCHEMA test FROM bob"
```

CCI: CCI-002165

Group ID (Vulid): V-72885
Group Title: SRG-APP-000120-DB-000061
Rule ID: SV-87537r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-002300
Rule Title: The audit information produced by PostgreSQL must be protected from unauthorized deletion.

Vulnerability Discussion: If audit data were to become compromised, then competent forensic analysis and discovery of the true source of potentially malicious system activity is impossible to achieve.

To ensure the veracity of audit data, the information system and/or the application must protect audit information from unauthorized deletion. This requirement can be achieved through multiple methods which will depend upon system architecture and design.

Some commonly employed methods include: ensuring log files enjoy the proper file system permissions utilizing file system protections; restricting access; and backing up log data to ensure log data is retained.

Applications providing a user interface to audit data will leverage user permissions and roles identifying the user accessing the data and the corresponding rights the user enjoys in order make access decisions regarding the deletion of audit data.

Audit information includes all information (e.g., audit records, audit settings, and audit reports) needed to successfully audit information system activity.

Deletion of database audit data could mask the theft of, or the unauthorized modification of, sensitive data stored in the database.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Review locations of audit logs, both internal to the database and database audit logs located at the operating system level.

Verify there are appropriate controls and permissions to protect the audit information from unauthorized modification.

stderr Logging

If the PostgreSQL server is configured to use stderr for logging, the logs will be owned by the database administrator (shown here as "postgres") with a default permissions level of 0600. The permissions can be configured in postgresql.conf.

To check the permissions for log files in postgresql.conf, as the database administrator (shown here as "postgres"), run the following command:

$ sudo su - postgres
$ grep "log_file_mode" ${PGDATA?}/postgresql.conf

If the permissions are not 0600, this is a finding.

Next, navigate to where the logs are stored. This can be found by running the following command against postgresql.conf as the database administrator (shown here as "postgres"):

$ sudo su - postgres
$ grep "log_directory" ${PGDATA?}/postgresql.conf

With the log directory identified, as the database administrator (shown here as "postgres"), list the permissions of the logs:

$ sudo su - postgres
$ ls -la ${PGDATA?}/pg_log

If logs are not owned by the database administrator (shown here as "postgres") and are not the same permissions as configured in postgresql.conf, this is a finding.

syslog Logging

If the PostgreSQL server is configured to use syslog for logging, consult the organizations syslog setting for permissions and ownership of logs.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

By default, the database administrator account is not accessible by unauthorized users. Only grant access to this account if required for operations.

stderr Logging

By default, the database administrator account is not accessible by unauthorized users. Only grant access to this account if required for operations.

With stderr logging enabled, as the database administrator (shown here as "postgres"), set the following parameter in postgresql.conf:

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
log_file_mode = 0600

syslog Logging

Check with the organization to see how syslog facilities are defined in their organization.

CCI: CCI-000164

Group ID (Vulid): V-72887
Group Title: SRG-APP-000374-DB-000322
Rule ID: SV-87539r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-002400
Rule Title: PostgreSQL must record time stamps, in audit records and application data, that can be mapped to Coordinated Universal Time (UTC,

formerly GMT).

Vulnerability Discussion: If time stamps are not consistently applied and there is no common time reference, it is difficult to perform forensic analysis.

Time stamps generated by PostgreSQL must include date and time. Time is commonly expressed in Coordinated Universal Time (UTC), a modern continuation of Greenwich Mean Time (GMT), or local time with an offset from UTC.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

When a PostgreSQL cluster is initialized using initdb, the PostgreSQL cluster will be configured to use the same time zone as the target server.

As the database administrator (shown here as "postgres"), check the current log_timezone setting by running the following SQL:

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

log_timezone
--------------
UTC
(1 row)
```

If log_timezone is not set to the desired time zone, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To change log_timezone in postgresql.conf to use a different time zone for logs, as the database administrator (shown here as "postgres"), run the following:

```
$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
log_timezone='UTC'
```

Next, restart the database:

```
# SYSTEMD SERVER ONLY
$ sudo systemctl restart postgresql-9.5

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

CCI: CCI-001890

Group ID (Vulid): V-72889
Group Title: SRG-APP-000267-DB-000163
Rule ID: SV-87541r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-002500
Rule Title: PostgreSQL must reveal detailed error messages only to the ISSO, ISSM, SA and DBA.

Vulnerability Discussion: If PostgreSQL provides too much information in error logs and administrative messages to the screen, this could lead to compromise. The structure and content of error messages need to be carefully considered by the organization and development team. The extent to which the information system is able to identify and handle error conditions is guided by organizational policy and operational requirements.

Some default PostgreSQL error messages can contain information that could aid an attacker in, among others things, identifying the database type, host address, or state of the database. Custom errors may contain sensitive customer information.

It is important that detailed error messages be visible only to those who are authorized to view them; that general users receive only generalized acknowledgment that errors have occurred; and that these generalized messages appear only when relevant to the user's task. For example, a message along the lines of, "An error has occurred. Unable to save your changes. If this problem persists, please contact your help desk" would be relevant. A message such as "Warning: your transaction generated a large number of page splits" would likely not be relevant.

Administrative users authorized to review detailed error messages typically are the ISSO, ISSM, SA, and DBA. Other individuals or roles may be specified according to organization-specific needs, with DBA approval.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Check PostgreSQL settings and custom database code to determine if detailed error messages are ever displayed to unauthorized individuals.

To check the level of detail for errors exposed to clients, as the database administrator (shown here as "postgres"), run the following:

$ sudo su - postgres
$ grep "client_min_messages"
${PGDATA?}/postgresql.conf

If client_min_messages is set to LOG or DEBUG, this is a finding.

If detailed error messages are displayed to individuals not authorized to view them, this is a finding.

stderr Logging

Logs may contain detailed information and should only be accessible by the database owner.

As the database administrator, verify the following settings of logs in the postgresql.conf file.

Note: Consult the organization's documentation on acceptable log privileges

$ sudo su - postgres
$ grep log_directory ${PGDATA?}/postgresql.conf
$ grep log_file_mode ${PGDATA?}/postgresql.conf

Next, verify the log files have the set configurations.

Note: Use location of logs from log_directory.

$ ls -l <audit_log_path>
total 32
-rw-------. 1 postgres postgres 0 Apr 8 00:00 postgresql-Fri.log
-rw-------. 1 postgres postgres 8288 Apr 11 17:36 postgresql-Mon.log
-rw-------. 1 postgres postgres 0 Apr 9 00:00 postgresql-Sat.log
-rw-------. 1 postgres postgres 0 Apr 10 00:00 postgresql-Sun.log
-rw-------. 1 postgres postgres 16212 Apr 7 17:05 postgresql-Thu.log
-rw-------. 1 postgres postgres 1130 Apr 6 17:56 postgresql-Wed.log

If logs are not owned by the database administrator or have permissions that are not 0600, this is a finding.

syslog Logging

If PostgreSQL is configured to use syslog for logging, consult the organizations location and permissions for syslog log files. If the logs are not owned by root or have permissions that are not 0600, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To set the level of detail for errors messages exposed to clients, as the database administrator (shown here as "postgres"), run the following commands:

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
client_min_messages = notice

CCI: CCI-001314

Group ID (Vulid): V-72891
Group Title: SRG-APP-000090-DB-000065
Rule ID: SV-87543r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-002600
Rule Title: PostgreSQL must allow only the ISSM (or individuals or roles appointed by the ISSM) to select which auditable events are to be audited.

Vulnerability Discussion: Without the capability to restrict which roles and individuals can select which events are audited, unauthorized personnel may be able to prevent or interfere with the auditing of critical events.

Suppression of auditing could permit an adversary to evade detection.

Misconfigured audits can degrade the system's performance by overwhelming the audit log. Misconfigured audits may also make it more difficult to establish, correlate, and investigate the events relating to an incident or identify those responsible for one.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Check PostgreSQL settings and documentation to determine whether designated personnel are able to select which auditable events are being audited.

As the database administrator (shown here as "postgres"), verify the permissions for PGDATA:

$ ls -la ${PGDATA?}

If anything in PGDATA is not owned by the database administrator, this is a finding.

Next, as the database administrator, run the following SQL:

$ sudo su - postgres
$ psql -c "\du"

Review the role permissions, if any role is listed as superuser but should not have that access, this is a finding.

Fix Text: Configure PostgreSQL's settings to allow designated personnel to select which auditable events are audited.

Using pgaudit allows administrators the flexibility to choose what they log. For an overview of the capabilities of pgaudit, see https://github.com/pgaudit/pgaudit.

See supplementary content APPENDIX-B for documentation on installing pgaudit.

See supplementary content APPENDIX-C for instructions on enabling logging. Only administrators/superuser can change PostgreSQL configurations. Access to the database administrator must be limited to designated personnel only.

To ensure that postgresql.conf is owned by the database owner:

$ chown postgres:postgres ${PGDATA?}/postgresql.conf
$ chmod 600 ${PGDATA?}/postgresql.conf

CCI: CCI-000171

Group ID (Vulid): V-72893
Group Title: SRG-APP-000360-DB-000320
Rule ID: SV-87545r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-002700
Rule Title: PostgreSQL must provide an immediate real-time alert to appropriate support staff of all audit failure events requiring real-time alerts.

Vulnerability Discussion: It is critical for the appropriate personnel to be aware if a system is at risk of failing to process audit logs as required. Without a real-time alert, security personnel may be unaware of an impending failure of the audit capability, and system operation may be adversely affected.

The appropriate support staff include, at a minimum, the ISSO and the DBA/SA.

Alerts provide organizations with urgent messages. Real-time alerts provide these messages immediately (i.e., the time from event detection to alert occurs in seconds or less).

The necessary monitoring and alerts may be implemented using features of PostgreSQL, the OS, third-party software, custom code, or a combination of these. The term "the system" is used to encompass all of these.

Check Content:
Review the system documentation to determine which audit failure events require real-time alerts.

Review the system settings and code. If the real-time alerting that is specified in the documentation is not enabled, this is a finding.

Fix Text: Configure the system to provide an immediate real-time alert to appropriate support staff when a specified audit failure occurs.

It is possible to create scripts or implement third-party tools to enable real-time alerting for audit failures in PostgreSQL.

CCI: CCI-001858

Group ID (Vulid): V-72895
Group Title: SRG-APP-000442-DB-000379
Rule ID: SV-87547r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-003000
Rule Title: PostgreSQL must maintain the confidentiality and integrity of information during reception.

Vulnerability Discussion: Information can be either unintentionally or maliciously disclosed or modified during reception, including, for example, during aggregation, at protocol transformation points, and during packing/unpacking. These unauthorized disclosures or modifications compromise the confidentiality or integrity of the information.

This requirement applies only to those applications that are either distributed or can allow access to data nonlocally. Use of this requirement will be

limited to situations where the data owner has a strict requirement for ensuring data integrity and confidentiality is maintained at every step of the data transfer and handling process.

When receiving data, PostgreSQL, associated applications, and infrastructure must leverage protection mechanisms.

PostgreSQL uses OpenSSL SSLv23_method() in fe-secure-openssl.c; while the name is misleading, this function enables only TLS encryption methods, not SSL.

See OpenSSL: https://mta.openssl.org/pipermail/openssl-dev/2015-May/001449.html

Check Content:
If the data owner does not have a strict requirement for ensuring data integrity and confidentiality is maintained at every step of the data transfer and handling process, this is not a finding.

As the database administrator (shown here as "postgres"), verify SSL is enabled in postgresql.conf by running the following SQL:

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

If SSL is off, this is a finding.

If PostgreSQL, associated applications, and infrastructure do not employ protective measures against unauthorized disclosure and modification during reception, this is a finding.

Fix Text: Implement protective measures against unauthorized disclosure and modification during reception.

To configure PostgreSQL to use SSL, see supplementary content APPENDIX-G for instructions on enabling SSL.

CCI: CCI-002422

Group ID (Vulid): V-72897
Group Title: SRG-APP-000133-DB-000200
Rule ID: SV-87549r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-003100
Rule Title: Database objects (including but not limited to tables, indexes, storage, trigger procedures, functions, links to software external to PostgreSQL, etc.) must be owned by database/DBMS principals authorized for ownership.

Vulnerability Discussion: Within the database, object ownership implies full privileges to the owned object, including the privilege to assign access to the owned objects to other subjects. Database functions and procedures can be coded using definer's rights. This allows anyone who utilizes the object to perform the actions if they were the owner. If not properly managed, this can lead to privileged actions being taken by unauthorized individuals.

Conversely, if critical tables or other objects rely on unauthorized owner accounts, these objects may be lost when an account is removed.

Check Content:
Review system documentation to identify accounts authorized to own database objects. Review accounts that own objects in the database(s).

If any database objects are found to be owned by users not authorized to own database objects, this is a finding.

To check the ownership of objects in the database, as the database administrator, run the following SQL:

```
$ sudo su - postgres
$ psql -x -c "\dn *.*"
$ psql -x -c "\dt *.*"
$ psql -x -c "\ds *.*"
$ psql -x -c "\dv *.*"
$ psql -x -c "\df+ *.*"
```

If any object is not owned by an authorized role for ownership, this is a finding.

Fix Text: Assign ownership of authorized objects to authorized object owner accounts.

Schema Owner

To create a schema owned by the user bob, run the following SQL:

```
$ sudo su - postgres
$ psql -c "CREATE SCHEMA test AUTHORIZATION bob
```

To alter the ownership of an existing object to be owned by the user bob, run the following SQL:

```
$ sudo su - postgres
$ psql -c "ALTER SCHEMA test OWNER TO bob"
```

CCI: CCI-001499

Group ID (Vulid): V-72899
Group Title: SRG-APP-000133-DB-000198
Rule ID: SV-87551r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-003200
Rule Title: The PostgreSQL software installation account must be restricted to authorized users.

Vulnerability Discussion: When dealing with change control issues, it should be noted any changes to the hardware, software, and/or firmware components of the information system and/or application can have significant effects on the overall security of the system.

If the system were to allow any user to make changes to software libraries, those changes might be implemented without undergoing the appropriate testing and approvals that are part of a robust change management process.

Accordingly, only qualified and authorized individuals must be allowed access to information system components for purposes of initiating changes, including upgrades and modifications.

DBA and other privileged administrative or application owner accounts are granted privileges that allow actions that can have a great impact on database security and operation. It is especially important to grant privileged access to only those persons who are qualified and authorized to use them.

Check Content:
Review procedures for controlling, granting access to, and tracking use of the PostgreSQL software installation account(s).

If access or use of this account is not restricted to the minimum number of personnel required or if unauthorized access to the account has been granted, this is a finding.

Fix Text: Develop, document, and implement procedures to restrict and track use of the PostgreSQL software installation account.

CCI: CCI-001499

Group ID (Vulid): V-72901
Group Title: SRG-APP-000133-DB-000199
Rule ID: SV-87553r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-003300
Rule Title: Database software, including PostgreSQL configuration files, must be stored in dedicated directories separate from the host OS and other applications.

Vulnerability Discussion: When dealing with change control issues, it should be noted, any changes to the hardware, software, and/or firmware components of the information system and/or application can potentially have significant effects on the overall security of the system.

Multiple applications can provide a cumulative negative effect. A vulnerability and subsequent exploit to one application can lead to an exploit of other applications sharing the same security context. For example, an exploit to a web server process that leads to unauthorized administrative access to host system directories can most likely lead to a compromise of all applications hosted by the same system. Database software not installed using dedicated directories both threatens and is threatened by other hosted applications. Access controls defined for one application may by default provide access to the other application's database objects or directories. Any method that provides any level of separation of security context assists in the protection between applications.

Check Content:
Review the PostgreSQL software library directory and any subdirectories.

If any non-PostgreSQL software directories exist on the disk directory, examine or investigate their use. If any of the directories are used by other applications, including third-party applications that use the PostgreSQL, this is a finding.

Only applications that are required for the functioning and administration, not use, of the PostgreSQL should be located in the same disk directory as the PostgreSQL software libraries.

If other applications are located in the same directory as PostgreSQL, this is a finding.

Fix Text: Install all applications on directories separate from the PostgreSQL software library directory. Relocate any directories or reinstall other application software that currently shares the PostgreSQL software library directory.

CCI: CCI-001499

Group ID (Vulid): V-72903
Group Title: SRG-APP-000101-DB-000044

Rule ID: SV-87555r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-003500
Rule Title: PostgreSQL must include additional, more detailed, organization-defined information in the audit records for audit events identified by type, location, or subject.

Vulnerability Discussion: Information system auditing capability is critical for accurate forensic analysis. Reconstruction of harmful events or forensic analysis is not possible if audit records do not contain enough information. To support analysis, some types of events will need information to be logged that exceeds the basic requirements of event type, time stamps, location, source, outcome, and user identity. If additional information is not available, it could negatively impact forensic investigations into user actions or other malicious events.

The organization must determine what additional information is required for complete analysis of the audited events. The additional information required is dependent on the type of information (e.g., sensitivity of the data and the environment within which it resides). At a minimum, the organization must employ either full-text recording of privileged commands or the individual identities of users of shared accounts, or both. The organization must maintain audit trails in sufficient detail to reconstruct events to determine the cause and impact of compromise.

Examples of detailed information the organization may require in audit records are full-text recording of privileged commands or the individual identities of shared account users.

Check Content:
Review the system documentation to identify what additional information the organization has determined necessary.

Check PostgreSQL settings and existing audit records to verify that all organization-defined additional, more detailed information is in the audit records for audit events identified by type, location, or subject.

If any additional information is defined and is not contained in the audit records, this is a finding.

Fix Text: Configure PostgreSQL audit settings to include all organization-defined detailed information in the audit records for audit events identified by type, location, or subject.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000135

Group ID (Vulid): V-72905
Group Title: SRG-APP-000342-DB-000302
Rule ID: SV-87557r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-003600
Rule Title: Execution of software modules (to include functions and trigger procedures) with elevated privileges must be restricted to necessary cases only.

Vulnerability Discussion: In certain situations, to provide required functionality, PostgreSQL needs to execute internal logic (stored procedures, functions, triggers, etc.) and/or external code modules with elevated privileges. However, if the privileges required for execution are at a higher level than the privileges assigned to organizational users invoking the functionality applications/programs, those users are indirectly provided with greater privileges than assigned by organizations.

Privilege elevation must be utilized only where necessary and protected from misuse.

This calls for inspection of application source code, which will require collaboration with the application developers. It is recognized that in many cases, the database administrator (DBA) is organizationally separate from the application developers, and may have limited, if any, access to source code. Nevertheless, protections of this type are so important to the secure operation of databases that they must not be ignored. At a minimum, the DBA must attempt to obtain assurances from the development organization that this issue has been addressed, and must document what has been discovered.

Check Content:
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.

To list all functions that have SECURITY DEFINER, as, the database administrator (shown here as "postgres"), run the following SQL:

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

In the query results, a prosecdef value of "t" on a row indicates that that function uses privilege elevation.

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

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

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

intended, this is a finding.

Fix Text: Determine where, when, how, and by what principals/subjects elevated privilege is needed.

To change a SECURITY DEFINER function to SECURITY INVOKER, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "ALTER FUNCTION <function_name> SECURITY INVOKER;"

CCI: CCI-002233

Group ID (Vulid): V-72907
Group Title: SRG-APP-000447-DB-000393
Rule ID: SV-87559r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-003700
Rule Title: When invalid inputs are received, PostgreSQL must behave in a predictable and documented manner that reflects organizational and system objectives.

Vulnerability Discussion: A common vulnerability is unplanned behavior when invalid inputs are received. This requirement guards against adverse or unintended system behavior caused by invalid inputs, where information system responses to the invalid input may be disruptive or cause the system to fail into an unsafe state.

The behavior will be derived from the organizational and system requirements and includes, but is not limited to, notification of the appropriate personnel, creating an audit record, and rejecting invalid input.

Check Content:
As the database administrator (shown here as "postgres"), make a small SQL syntax error in psql by running the following:

$ sudo su - postgres
$ psql -c "CREAT TABLEincorrect_syntax(id INT)"
ERROR: syntax error at or near "CREAT"

Now, as the database administrator (shown here as "postgres"), verify the syntax error was logged (change the log file name and part to suit the circumstances):

$ sudo su - postgres
$ cat ~/9.5/data/pg_log/postgresql-Wed.log
2016-03-30 16:18:10.772 EDT postgres postgres 5706bb87.90dERROR: syntax error at or near "CRT" at character 1
2016-03-30 16:18:10.772 EDT postgres postgres 5706bb87.90dSTATEMENT: CRT TABLE incorrect_syntax(id INT);

Review system documentation to determine how input errors from application to PostgreSQL are to be handled in general and if any special handling is defined for specific circumstances.

If it does not implement the documented behavior, this is a finding.

Fix Text: Enable logging.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

All errors and denials are logged if logging is enabled.

CCI: CCI-002754

Group ID (Vulid): V-72909
Group Title: SRG-APP-000356-DB-000314
Rule ID: SV-87561r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-003800
Rule Title: PostgreSQL must utilize centralized management of the content captured in audit records generated by all components of PostgreSQL.

Vulnerability Discussion: Without the ability to centrally manage the content captured in the audit records, identification, troubleshooting, and correlation of suspicious behavior would be difficult and could lead to a delayed or incomplete analysis of an ongoing attack.

The content captured in audit records must be managed from a central location (necessitating automation). Centralized management of audit records and logs provides for efficiency in maintenance and management of records, as well as the backup and archiving of those records.

PostgreSQL may write audit records to database tables, to files in the file system, to other kinds of local repository, or directly to a centralized log management system. Whatever the method used, it must be compatible with off-loading the records to the centralized system.

Check Content:
On UNIX systems, PostgreSQL can be configured to use stderr, csvlog and syslog. To send logs to a centralized location, syslog should be used.

As the database owner (shown here as "postgres"), ensure PostgreSQL uses syslog by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_destination"

As the database owner (shown here as "postgres"), check which log facility PostgreSQL is configured by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW syslog_facility"

Check with the organization to see how syslog facilities are defined in their organization.

If PostgreSQL audit records are not written directly to or systematically transferred to a centralized log management system, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

With logging enabled, as the database owner (shown here as "postgres"), configure the follow parameters in postgresql.conf:

Note: Consult the organization on how syslog facilities are defined in the syslog daemon configuration.

$ sudo su - postgres
$ vi 'log_destination' ${PGDATA?}/postgresql.conf
log_destination = 'syslog'
syslog_facility = 'LOCAL0'
syslog_ident = 'postgres'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-001844

Group ID (Vulid): V-72911
Group Title: SRG-APP-000233-DB-000124
Rule ID: SV-87563r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004000
Rule Title: PostgreSQL must isolate security functions from non-security functions.

Vulnerability Discussion: An isolation boundary provides access control and protects the integrity of the hardware, software, and firmware that perform security functions.

Security functions are the hardware, software, and/or firmware of the information system responsible for enforcing the system security policy and supporting the isolation of code and data on which the protection is based.

Developers and implementers can increase the assurance in security functions by employing well-defined security policy models; structured, disciplined, and rigorous hardware and software development techniques; and sound system/security engineering principles.

Database Management Systems typically separate security functionality from non-security functionality via separate databases or schemas. Database objects or code implementing security functionality should not be commingled with objects or code implementing application logic. When security and non-security functionality are commingled, users who have access to non-security functionality may be able to access security functionality.

Check Content:
Check PostgreSQL settings to determine whether objects or code implementing security functionality are located in a separate security domain, such as a separate database or schema created specifically for security functionality.

By default, all objects in pg_catalog and information_schema are owned by the database administrator.

To check the access controls for those schemas, as the database administrator (shown here as "postgres"), run the following commands to review the access privileges granted on the data dictionary and security tables, views, sequences, functions and trigger procedures:

$ sudo su - postgres
$ psql -x -c "\dp pg_catalog.*"
$ psql -x -c "\dp information_schema.*"

Repeat the \dp statements for any additional schemas that contain locally defined security objects.

Repeat using \df+*.* to review ownership of PostgreSQL functions:

$ sudo su - postgres
$ psql -x -c "\df+ pg_catalog.*"
$ psql -x -c "\df+ information_schema.*"

Refer to the PostgreSQL online documentation for GRANT for help in interpreting the Access Privileges column in the output from \du. Note that an entry starting with an equals sign indicates privileges granted to Public (all users). By default, most of the tables and views in the pg_catalog and information_schema schemas can be read by Public.

If any user besides the database administrator(s) is listed in access privileges and not documented, this is a finding.

If security-related database objects or code are not kept separate, this is a finding.

Fix Text: Do not locate security-related database objects with application tables or schema.

Review any site-specific applications security modules built into the database: determine what schema they are located in and take appropriate action.

Do not grant access to pg_catalog or information_schema to anyone but the database administrator(s). Access to the database administrator account(s) must not be granted to anyone without official approval.

CCI: CCI-001084

Group ID (Vulid): V-72913
Group Title: SRG-APP-000381-DB-000361
Rule ID: SV-87565r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004100
Rule Title: PostgreSQL must produce audit records of its enforcement of access restrictions associated with changes to the configuration of PostgreSQL or database(s).

Vulnerability Discussion: Without auditing the enforcement of access restrictions against changes to configuration, it would be difficult to identify attempted attacks and an audit trail would not be available for forensic investigation for after-the-fact actions.

Enforcement actions are the methods or mechanisms used to prevent unauthorized changes to configuration settings. Enforcement action methods may be as simple as denying access to a file based on the application of file permissions (access restriction). Audit items may consist of lists of actions blocked by access restrictions or changes identified after the fact.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To verify that system denies are logged when unprivileged users attempt to change database configuration, as the database administrator (shown here as "postgres"), run the following commands:

$ sudo su - postgres
$ psql

Next, create a role with no privileges, change the current role to that user and attempt to change a configuration by running the following SQL:

CREATE ROLE bob;
SET ROLE bob;
SET pgaudit.role='test';

Now check pg_log (use the latest log):

$ cat ${PGDATA?}/pg_log/postgresql-Thu.log
< 2016-01-28 17:57:34.092 UTC bob postgres: >ERROR: permission denied to set parameter "pgaudit.role"
< 2016-01-28 17:57:34.092 UTC bob postgres: >STATEMENT: SET pgaudit.role='test';

If the denial is not logged, this is a finding.

By default PostgreSQL configuration files are owned by the postgres user and cannot be edited by non-privileged users:

$ ls -la ${PGDATA?} | grep postgresql.conf
-rw-------. 1 postgres postgres 21758 Jan 22 10:27 postgresql.conf

If postgresql.conf is not owned by the database owner and does not have read and write permissions for the owner, this is a finding.

Fix Text: Enable logging.

All denials are logged by default if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions

on enabling logging.

CCI: CCI-001814

Group ID (Vulid): V-72915
Group Title: SRG-APP-000118-DB-000059
Rule ID: SV-87567r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004200
Rule Title: The audit information produced by PostgreSQL must be protected from unauthorized read access.

Vulnerability Discussion: If audit data were to become compromised, then competent forensic analysis and discovery of the true source of potentially malicious system activity is difficult, if not impossible, to achieve. In addition, access to audit records provides information an attacker could potentially use to his or her advantage.

To ensure the veracity of audit data, the information system and/or the application must protect audit information from any and all unauthorized access. This includes read, write, copy, etc.

This requirement can be achieved through multiple methods which will depend upon system architecture and design. Some commonly employed methods include ensuring log files enjoy the proper file system permissions utilizing file system protections and limiting log data location.

Additionally, applications with user interfaces to audit records should not allow for the unfettered manipulation of or access to those records via the application. If the application provides access to the audit data, the application becomes accountable for ensuring that audit information is protected from unauthorized access.

Audit information includes all information (e.g., audit records, audit settings, and audit reports) needed to successfully audit information system activity.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Review locations of audit logs, both internal to the database and database audit logs located at the operating system level.

Verify there are appropriate controls and permissions to protect the audit information from unauthorized access.

syslog Logging

If PostgreSQL is configured to use syslog for logging, consult the organizations location and permissions for syslog log files.

stderr Logging

As the database administrator (shown here as "postgres"), check the current log_file_mode configuration by running the following:

Note: Consult the organization's documentation on acceptable log privileges

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

If log_file_mode is not 600, this is a finding.

Next, check the current log_destination path by running the following SQL:

Note: This is relative to PGDATA.

```
$ psql -c "SHOW log_destination"
```

Next, verify the log files have the set configurations in the log_destination:

Note: Use location of logs from log_directory.

```
$ ls -l ${PGDATA?}/pg_log/
total 32
-rw-------. 1 postgres postgres 0 Apr 8 00:00 postgresql-Fri.log
-rw-------. 1 postgres postgres 8288 Apr 11 17:36 postgresql-Mon.log
-rw-------. 1 postgres postgres 0 Apr 9 00:00 postgresql-Sat.log
-rw-------. 1 postgres postgres 0 Apr 10 00:00 postgresql-Sun.log
-rw-------. 1 postgres postgres 16212 Apr 7 17:05 postgresql-Thu.log
-rw-------. 1 postgres postgres 1130 Apr 6 17:56 postgresql-Wed.log
```

If logs with 600 permissions do not exist in log_destination, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

syslog Logging

If PostgreSQL is configured to use syslog for logging, consult the organizations location and permissions for syslog log files.

stderr Logging

If PostgreSQL is configured to use stderr for logging, permissions of the log files can be set in postgresql.conf.

As the database administrator (shown here as "postgres"), edit the following settings of logs in the postgresql.conf file:

Note: Consult the organization's documentation on acceptable log privileges

```
$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
log_file_mode = 0600
```

Next, as the system administrator, reload the server with the new configuration:

```
# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

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

CCI: CCI-000162

Group ID (Vulid): V-72917
Group Title: SRG-APP-000454-DB-000389
Rule ID: SV-87569r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004300
Rule Title: When updates are applied to PostgreSQL software, any software components that have been replaced or made unnecessary must be removed.

Vulnerability Discussion: Previous versions of PostgreSQL components that are not removed from the information system after updates have been installed may be exploited by adversaries.

Some PostgreSQL installation tools may remove older versions of software automatically from the information system. In other cases, manual review and removal will be required. In planning installations and upgrades, organizations must include steps (automated, manual, or both) to identify and remove the outdated modules.

A transition period may be necessary when both the old and the new software are required. This should be taken into account in the planning.

Check Content:
To check software installed by packages, as the system administrator, run the following command:

```
# RHEL/CENT Systems
$ sudo rpm -qa | grep postgres
```

If multiple versions of postgres are installed but are unused, this is a finding.

Fix Text: Use package managers (RPM or apt-get) for installing PostgreSQL. Unused software is removed when updated.

CCI: CCI-002617

Group ID (Vulid): V-72919
Group Title: SRG-APP-000494-DB-000344
Rule ID: SV-87571r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004400
Rule Title: PostgreSQL must generate audit records when categorized information (e.g., classification levels/security levels) is accessed.

Vulnerability Discussion: Changes in categorized information must be tracked. Without an audit trail, unauthorized access to protected data could go undetected.

For detailed information on categorizing information, refer to FIPS Publication 199, Standards for Security Categorization of Federal Information and Information Systems, and FIPS Publication 200, Minimum Security Requirements for Federal Information and Information Systems.

Check Content:
As the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "SHOW pgaudit.log"

If pgaudit.log does not contain, "ddl, write, role", this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Using `pgaudit` the DBMS (PostgreSQL) can be configured to audit these requests. See supplementary content `APPENDIX-B` for documentation on installing `pgaudit`.

With `pgaudit` installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log = 'ddl, write, role'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72921
Group Title: SRG-APP-000492-DB-000333
Rule ID: SV-87573r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004500
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to access security objects occur.

Vulnerability Discussion: Changes to the security configuration must be tracked.

This requirement applies to situations where security data is retrieved or modified via data manipulation operations, as opposed to via specialized security functionality.

In an SQL environment, types of access include, but are not necessarily limited to:

SELECT
INSERT
UPDATE
DELETE
EXECUTE

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), setup a test schema and revoke users privileges from using it by running the following SQL:

$ sudo su - postgres
$ psql -c "CREATE SCHEMA stig_test_schema AUTHORIZATION postgres"
$ psql -c "REVOKE ALL ON SCHEMA stig_test_schema FROM public"
$ psql -c "GRANT ALL ON SCHEMA stig_test_schema TO postgres"

Next, create a test table, insert a value into that table for the following checks by running the following SQL:

$ psql -c "CREATE TABLE stig_test_schema.stig_test_table(id INT)"
$ psql -c "INSERT INTO stig_test_schema.stig_test_table(id) VALUES (0)"

CREATE
Attempt to CREATE a table in the stig_test_schema schema with a role that does not have privileges by running the following SQL:

psql -c "CREATE ROLE bob; SET ROLE bob; CREATE TABLE stig_test_schema.test_table(id INT);"
ERROR: permission denied for schema stig_test_schema

Next, as a database administrator (shown here as "postgres"), verify that the denial was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-09 09:55:19.423 EST postgres 56e0393f.186b postgres: >ERROR: permission denied for schema stig_test_schema at character 14
< 2016-03-09 09:55:19.423 EST postgres 56e0393f.186b postgres: >STATEMENT: CREATE TABLE stig_test_schema.test_table(id INT);

If the denial is not logged, this is a finding.

INSERT
As role bob, attempt to INSERT into the table created earlier, stig_test_table by running the following SQL:

$ sudo su - postgres
$ psql -c "SET ROLE bob; INSERT INTO stig_test_schema.stig_test_table(id) VALUES (0);"

Next, as a database administrator (shown here as "postgres"), verify that the denial was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-09 09:58:30.709 EST postgres 56e0393f.186b postgres: >ERROR: permission denied for schema stig_test_schema at character 13
< 2016-03-09 09:58:30.709 EST postgres 56e0393f.186b postgres: >STATEMENT: INSERT INTO stig_test_schema.stig_test_table(id) VALUES (0);

If the denial is not logged, this is a finding.

SELECT
As role bob, attempt to SELECT from the table created earlier, stig_test_table by running the following SQL:

$ sudo su - postgres
$ psql -c "SET ROLE bob; SELECT * FROM stig_test_schema.stig_test_table;"

Next, as a database administrator (shown here as "postgres"), verify that the denial was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-09 09:57:58.327 EST postgres 56e0393f.186b postgres: >ERROR: permission denied for schema stig_test_schema at character 15
< 2016-03-09 09:57:58.327 EST postgres 56e0393f.186b postgres: >STATEMENT: SELECT * FROM stig_test_schema.stig_test_table;

If the denial is not logged, this is a finding.

ALTER
As role bob, attempt to ALTER the table created earlier, stig_test_table by running the following SQL:

$ sudo su - postgres
$ psql -c "SET ROLE bob; ALTER TABLE stig_test_schema.stig_test_table ADD COLUMN name TEXT;"

Next, as a database administrator (shown here as "postgres"), verify that the denial was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-09 10:03:43.765 EST postgres 56e0393f.186b postgres: >STATEMENT: ALTER TABLE stig_test_schema.stig_test_table ADD COLUMN name TEXT;

If the denial is not logged, this is a finding.

UPDATE
As role bob, attempt to UPDATE a row created earlier, stig_test_table by running the following SQL:

$ sudo su - postgres
$ psql -c "SET ROLE bob; UPDATE stig_test_schema.stig_test_table SET id=1 WHERE id=0;"

Next, as a database administrator (shown here as "postgres"), verify that the denial was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-09 10:08:27.696 EST postgres 56e0393f.186b postgres: >ERROR: permission denied for schema stig_test_schema at character 8
< 2016-03-09 10:08:27.696 EST postgres 56e0393f.186b postgres: >STATEMENT: UPDATE stig_test_schema.stig_test_table SET id=1 WHERE id=0;

If the denial is not logged, this is a finding.

DELETE
As role bob, attempt to DELETE a row created earlier, stig_test_table by running the following SQL:

$ sudo su - postgres
$ psql -c "SET ROLE bob; DELETE FROM stig_test_schema.stig_test_table WHERE id=0;"

Next, as a database administrator (shown here as "postgres"), verify that the denial was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-09 10:09:29.607 EST postgres 56e0393f.186b postgres: >ERROR: permission denied for schema stig_test_schema at character 13
< 2016-03-09 10:09:29.607 EST postgres 56e0393f.186b postgres: >STATEMENT: DELETE FROM stig_test_schema.stig_test_table WHERE id=0;

If the denial is not logged, this is a finding.

PREPARE
As role bob, attempt to execute a prepared system using PREPARE by running the following SQL:

$ sudo su - postgres
$ psql -c "SET ROLE bob; PREPARE stig_test_plan(int) AS SELECT id FROM stig_test_schema.stig_test_table WHERE id=$1;"

Next, as a database administrator (shown here as "postgres"), verify that the denial was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-09 10:16:22.628 EST postgres 56e03e02.18e4 postgres: >ERROR: permission denied for schema stig_test_schema at character 46
< 2016-03-09 10:16:22.628 EST postgres 56e03e02.18e4 postgres: >STATEMENT: PREPARE stig_test_plan(int) AS SELECT id FROM
stig_test_schema.stig_test_table WHERE id=$1;

If the denial is not logged, this is a finding.

DROP
As role bob, attempt to DROP the table created earlier stig_test_table by running the following SQL:

$ sudo su - postgres
$ psql -c "SET ROLE bob; DROP TABLE stig_test_schema.stig_test_table;"

Next, as a database administrator (shown here as "postgres"), verify that the denial was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-09 10:18:55.255 EST postgres 56e03e02.18e4 postgres: >ERROR: permission denied for schema stig_test_schema
< 2016-03-09 10:18:55.255 EST postgres 56e03e02.18e4 postgres: >STATEMENT: DROP TABLE stig_test_schema.stig_test_table;

If the denial is not logged, this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to access security objects occur.

All denials are logged if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000172

Group ID (Vulid): V-72923
Group Title: SRG-APP-000503-DB-000351
Rule ID: SV-87575r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004600
Rule Title: PostgreSQL must generate audit records when unsuccessful logons or connection attempts occur.

Vulnerability Discussion: For completeness of forensic analysis, it is necessary to track failed attempts to log on to PostgreSQL. While positive identification may not be possible in a case of failed authentication, as much information as possible about the incident must be captured.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

In this example the user joe will log into the Postgres database unsuccessfully:

$ psql -d postgres -U joe

As the database administrator (shown here as "postgres"), check pg_log for a FATAL connection audit trail:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/postgresql-Tue.log
< 2016-02-16 16:18:13.027 EST joe 56c65135.b5f postgres: >LOG: connection authorized: user=joe database=postgres
< 2016-02-16 16:18:13.027 EST joe 56c65135.b5f postgres: >FATAL: role "joe" does not exist

If an audit record is not generated each time a user (or other principal) attempts, but fails to log on or connect to PostgreSQL (including attempts where

the user ID is invalid/unknown), this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

If logging is enabled the following configurations must be made to log unsuccessful connections, date/time, username, and session identifier.

First, as the database administrator (shown here as "postgres"), edit postgresql.conf:

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

Edit the following parameters:

```
log_connections = on
log_line_prefix = '< %m %u %c: >'
```

Where:
* %m is the time and date
* %u is the username
* %c is the session ID for the connection

Now, as the system administrator, reload the server with the new configuration:

```
# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

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

CCI: CCI-000172

Group ID (Vulid): V-72925
Group Title: SRG-APP-000505-DB-000352
Rule ID: SV-87577r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004700
Rule Title: PostgreSQL must generate audit records showing starting and ending time for user access to the database(s).

Vulnerability Discussion: For completeness of forensic analysis, it is necessary to know how long a user's (or other principal's) connection to PostgreSQL lasts. This can be achieved by recording disconnections, in addition to logons/connections, in the audit logs.

Disconnection may be initiated by the user or forced by the system (as in a timeout) or result from a system or network failure. To the greatest extent possible, all disconnections must be logged.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, log into the database with the postgres user by running the following commands:

```
$ sudo su - postgres
$ psql -U postgres
```

Next, as the database administrator, verify the log for a connection audit trail:

```
$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/postgresql-Tue.log
< 2016-02-23 20:25:39.931 EST postgres 56cfa993.7a72 postgres: >LOG: connection authorized: user=postgres database=postgres
< 2016-02-23 20:27:45.428 EST postgres 56cfa993.7a72 postgres: >LOG: AUDIT: SESSION,1,1,READ,SELECT,,,SELECT current_user;,<none>
< 2016-02-23 20:27:47.988 EST postgres 56cfa993.7a72 postgres: >LOG: disconnection: session time: 0:00:08.057 user=postgres database=postgres
host=[local]
```

If connections are not logged, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

If logging is enabled the following configurations must be made to log connections, date/time, username, and session identifier.

First, as the database administrator (shown here as "postgres"), edit postgresql.conf by running the following:

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

Edit the following parameters:

log_connections = on
log_disconnections = on
log_line_prefix = '< %m %u %c: >'

Where:
* %m is the time and date
* %u is the username
* %c is the session ID for the connection

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72927
Group Title: SRG-APP-000496-DB-000335
Rule ID: SV-87579r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004800
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to modify security objects occur.

Vulnerability Discussion: Changes in the database objects (tables, views, procedures, functions) that record and control permissions, privileges, and roles granted to users and roles must be tracked. Without an audit trail, unauthorized changes to the security subsystem could go undetected. The database could be severely compromised or rendered inoperative.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

As the database administrator (shown here as "postgres"), create a test role by running the following SQL:

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

Next, to test if audit records are generated from unsuccessful attempts at modifying security objects, run the following SQL:

$ sudo su - postgres
$ psql -c "SET ROLE bob; UPDATE pg_authid SET rolsuper = 't' WHERE rolname = 'bob';"

Next, as the database administrator (shown here as "postgres"), verify that the denials were logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-17 10:34:00.017 EDT bob 56eabf52.b62 postgres: >ERROR: permission denied for relation pg_authid
< 2016-03-17 10:34:00.017 EDT bob 56eabf52.b62 postgres: >STATEMENT: UPDATE pg_authid SET rolsuper = 't' WHERE rolname = 'bob';

If denials are not logged, this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to modify security objects occur.

Unsuccessful attempts to modifying security objects can be logged if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000172

Group ID (Vulid): V-72929
Group Title: SRG-APP-000495-DB-000326
Rule ID: SV-87581r1_rule

Severity: CAT II
Rule Version (STIG-ID): PGS9-00-004900
Rule Title: PostgreSQL must generate audit records when privileges/permissions are added.

Vulnerability Discussion: Changes in the permissions, privileges, and roles granted to users and roles must be tracked. Without an audit trail, unauthorized elevation or restriction of privileges could go undetected. Elevated privileges give users access to information and functionality that they should not have; restricted privileges wrongly deny access to authorized users.

In an SQL environment, adding permissions is typically done via the GRANT command, or, in the negative, the REVOKE command.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), create a role by running the following SQL:

Change the privileges of another user:

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

Next, GRANT then REVOKE privileges from the role:

$ psql -c "GRANT CONNECT ON DATABASE postgres TO bob"
$ psql -c "REVOKE CONNECT ON DATABASE postgres FROM bob"

postgres=# REVOKE CONNECT ON DATABASE postgres FROM bob;
REVOKE

postgres=# GRANT CONNECT ON DATABASE postgres TO bob;
GRANT

Now, as the database administrator (shown here as "postgres"), verify the events were logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-07-13 16:25:21.103 EDT postgres postgres LOG: > AUDIT: SESSION,1,1,ROLE,GRANT,,,GRANT CONNECT ON DATABASE postgres TO bob,<none>
< 2016-07-13 16:25:25.520 EDT postgres postgres LOG: > AUDIT: SESSION,1,1,ROLE,REVOKE,,,REVOKE CONNECT ON DATABASE postgres FROM bob,<none>

If the above steps cannot verify that audit records are produced when privileges/permissions/role memberships are added, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log = 'role'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72931
Group Title: SRG-APP-000502-DB-000349
Rule ID: SV-87583r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005000
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to delete categorized information (e.g., classification levels/security levels) occur.

Vulnerability Discussion: Changes in categorized information must be tracked. Without an audit trail, unauthorized access to protected data could go undetected.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

For detailed information on categorizing information, refer to FIPS Publication 199, Standards for Security Categorization of Federal Information and Information Systems, and FIPS Publication 200, Minimum Security Requirements for Federal Information and Information Systems.

Check Content:
First, as the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Next, verify that role, read, write, and ddl auditing are enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain role, read, write, and ddl, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

All errors and denials are logged if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log='ddl, role, read, write'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72933
Group Title: SRG-APP-000503-DB-000350
Rule ID: SV-87585r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005100
Rule Title: PostgreSQL must generate audit records when successful logons or connections occur.

Vulnerability Discussion: For completeness of forensic analysis, it is necessary to track who/what (a user or other principal) logs on to PostgreSQL.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), check if log_connections is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_connections"

If log_connections is off, this is a finding.

Next, verify the logs that the previous connection to the database was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>

< 2016-02-16 15:54:03.934 EST postgres postgres 56c64b8b.aeb: >LOG: connection authorized: user=postgres database=postgres

If an audit record is not generated each time a user (or other principal) logs on or connects to PostgreSQL, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

If logging is enabled the following configurations must be made to log connections, date/time, username, and session identifier.

First, as the database administrator (shown here as "postgres"), edit postgresql.conf:

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

Edit the following parameters as such:

log_connections = on
log_line_prefix = '< %m %u %d %c: >'

Where:
* %m is the time and date
* %u is the username
* %d is the database
* %c is the session ID for the connection

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72939
Group Title: SRG-APP-000501-DB-000336
Rule ID: SV-87591r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005200
Rule Title: PostgreSQL must generate audit records when security objects are deleted.

Vulnerability Discussion: The removal of security objects from the database/PostgreSQL would seriously degrade a system's information assurance posture. If such an event occurs, it must be logged.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), create a test table stig_test, enable row level security, and create a policy by running the following SQL:

$ sudo su - postgres
$ psql -c "CREATE TABLE stig_test(id INT)"
$ psql -c "ALTER TABLE stig_test ENABLE ROW LEVEL SECURITY"
$ psql -c "CREATE POLICY lock_table ON stig_test USING ('postgres' = current_user)"

Next, drop the policy and disable row level security:

$ psql -c "DROP POLICY lock_table ON stig_test"
$ psql -c "ALTER TABLE stig_test DISABLE ROW LEVEL SECURITY"

Now, as the database administrator (shown here as "postgres"), verify the security objects deletions were logged:

$ cat ${PGDATA?}/pg_log/<latest_log>
2016-03-30 14:54:18.991 EDT postgres postgres LOG: AUDIT: SESSION,11,1,DDL,DROP POLICY,,,DROP POLICY lock_table ON stig_test;,<none>
2016-03-30 14:54:42.373 EDT postgres postgres LOG: AUDIT: SESSION,12,1,DDL,ALTER TABLE,,,ALTER TABLE stig_test DISABLE ROW LEVEL SECURITY;,<none>

If audit records are not produced when security objects are dropped, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on

configuring PGDATA.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log = 'ddl'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72941
Group Title: SRG-APP-000091-DB-000325
Rule ID: SV-87593r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005300
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to retrieve privileges/permissions occur.

Vulnerability Discussion: Under some circumstances, it may be useful to monitor who/what is reading privilege/permission/role information. Therefore, it must be possible to configure auditing to do this. PostgreSQLs typically make such information available through views or functions.

This requirement addresses explicit requests for privilege/permission/role membership information. It does not refer to the implicit retrieval of privileges/permissions/role memberships that PostgreSQL continually performs to determine if any and every action on the database is permitted.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), create a role 'bob' by running the following SQL:

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

Next, attempt to retrieve information from the pg_authid table:

$ psql -c "SET ROLE bob; SELECT * FROM pg_authid"

Now, as the database administrator (shown here as "postgres"), verify the event was logged in pg_log:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-07-13 16:49:58.864 EDT postgres postgres ERROR: > permission denied for relation pg_authid
< 2016-07-13 16:49:58.864 EDT postgres postgres STATEMENT: > SELECT * FROM pg_authid;

If the above steps cannot verify that audit records are produced when PostgreSQL denies retrieval of privileges/permissions/role memberships, this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to access privileges occur.

All denials are logged if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000172

Group ID (Vulid): V-72945
Group Title: SRG-APP-000499-DB-000331
Rule ID: SV-87597r1_rule

Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005400
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to delete privileges/permissions occur.

Vulnerability Discussion: Failed attempts to change the permissions, privileges, and roles granted to users and roles must be tracked. Without an audit trail, unauthorized attempts to elevate or restrict privileges could go undetected.

In an SQL environment, deleting permissions is typically done via the REVOKE command.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), create the roles joe and bob with LOGIN by running the following SQL:

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

Next, set current role to bob and attempt to alter the role joe:

$ psql -c "SET ROLE bob; ALTER ROLE joe NOLOGIN"

Now, as the database administrator (shown here as "postgres"), verify the denials are logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-03-17 11:28:10.004 EDT bob 56eacd05.cda postgres: >ERROR: permission denied to drop role
< 2016-03-17 11:28:10.004 EDT bob 56eacd05.cda postgres: >STATEMENT: DROP ROLE joe;

If audit logs are not generated when unsuccessful attempts to delete privileges/permissions occur, this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to delete privileges occur.

All denials are logged if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000172

Group ID (Vulid): V-72947
Group Title: SRG-APP-000091-DB-000066
Rule ID: SV-87599r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005500
Rule Title: PostgreSQL must be able to generate audit records when privileges/permissions are retrieved.

Vulnerability Discussion: Under some circumstances, it may be useful to monitor who/what is reading privilege/permission/role information. Therefore, it must be possible to configure auditing to do this. PostgreSQLs typically make such information available through views or functions.

This requirement addresses explicit requests for privilege/permission/role membership information. It does not refer to the implicit retrieval of privileges/permissions/role memberships that PostgreSQL continually performs to determine if any and every action on the database is permitted.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), check if pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

If pgaudit is not found in the results, this is a finding.

Next, as the database administrator (shown here as "postgres"), list all role memberships for the database:

$ sudo su - postgres
$ psql -c "\du"

Next, verify the query was logged:

$ sudo su - postgres

$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-01-28 19:43:12.126 UTC postgres postgres: >LOG: AUDIT: SESSION,1,1,READ,SELECT,,,"SELECT r.rolname, r.rolsuper, r.rolinherit,
r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
r.rolconnlimit, r.rolvaliduntil,
ARRAY(SELECT b.rolname
FROM pg_catalog.pg_auth_members m
JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolbypassrls
FROM pg_catalog.pg_roles r
ORDER BY 1;",<none>

If audit records are not produced, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log_catalog = 'on'
pgaudit.log = 'read'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72949
Group Title: SRG-APP-000498-DB-000347
Rule ID: SV-87601r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005600
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to modify categorized information (e.g., classification levels/security levels) occur.

Vulnerability Discussion: Changes in categorized information must be tracked. Without an audit trail, unauthorized access to protected data could go undetected.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

For detailed information on categorizing information, refer to FIPS Publication 199, Standards for Security Categorization of Federal Information and Information Systems, and FIPS Publication 200, Minimum Security Requirements for Federal Information and Information Systems.

Check Content:
First, as the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Next, verify that role, read, write, and ddl auditing are enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain role, read, write, and ddl, this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to modify categories of information.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging. All denials are logged when logging is enabled.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log='ddl, role, read, write'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72951
Group Title: SRG-APP-000507-DB-000357
Rule ID: SV-87603r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005700
Rule Title: PostgreSQL must generate audit records when unsuccessful accesses to objects occur.

Vulnerability Discussion: Without tracking all or selected types of access to all or selected objects (tables, views, procedures, functions, etc.), it would be difficult to establish, correlate, and investigate the events relating to an incident or identify those responsible for one.

In an SQL environment, types of access include, but are not necessarily limited to:

SELECT
INSERT
UPDATE
DROP
EXECUTE

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), create a schema, test_schema, create a table, test_table, within test_schema, and insert a value:

$ sudo su - postgres
$ psql -c "CREATE SCHEMA test_schema"
$ psql -c "CREATE TABLE test_schema.test_table(id INT)"
$ psql -c "INSERT INTO test_schema.test_table(id) VALUES (0)"

Next, create a role 'bob' and attempt to SELECT, INSERT, UPDATE, and DROP from the test table:

$ psql -c "CREATE ROLE BOB"
$ psql -c "SELECT * FROM test_schema.test_table"
$ psql -c "INSERT INTO test_schema.test_table VALUES (0)"
$ psql -c "UPDATE test_schema.test_table SET id = 1 WHERE id = 0"
$ psql -c "DROP TABLE test_schema.test_table"
$ psql -c "DROP SCHEMA test_schema"

Now, as the database administrator (shown here as "postgres"), review PostgreSQL/database security and audit settings to verify that audit records are created for unsuccessful attempts at the specified access to the specified objects:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
2016-03-30 17:23:41.254 EDT postgres postgres ERROR: permission denied for schema test_schema at character 15
2016-03-30 17:23:41.254 EDT postgres postgres STATEMENT: SELECT * FROM test_schema.test_table;
2016-03-30 17:23:53.973 EDT postgres postgres ERROR: permission denied for schema test_schema at character 13
2016-03-30 17:23:53.973 EDT postgres postgres STATEMENT: INSERT INTO test_schema.test_table VALUES (0);
2016-03-30 17:24:32.647 EDT postgres postgres ERROR: permission denied for schema test_schema at character 8
2016-03-30 17:24:32.647 EDT postgres postgres STATEMENT: UPDATE test_schema.test_table SET id = 1 WHERE id = 0;
2016-03-30 17:24:46.197 EDT postgres postgres ERROR: permission denied for schema test_schema
2016-03-30 17:24:46.197 EDT postgres postgres STATEMENT: DROP TABLE test_schema.test_table;
2016-03-30 17:24:51.582 EDT postgres postgres ERROR: must be owner of schema test_schema

2016-03-30 17:24:51.582 EDT postgres postgres STATEMENT: DROP SCHEMA test_schema;

If any of the above steps did not create audit records for SELECT, INSERT, UPDATE, and DROP, this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to access objects occur.

All errors and denials are logged if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000172

Group ID (Vulid): V-72953
Group Title: SRG-APP-000504-DB-000354
Rule ID: SV-87605r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005800
Rule Title: PostgreSQL must generate audit records for all privileged activities or other system-level access.

Vulnerability Discussion: Without tracking privileged activity, it would be difficult to establish, correlate, and investigate the events relating to an incident or identify those responsible for one.

System documentation should include a definition of the functionality considered privileged.

A privileged function in this context is any operation that modifies the structure of the database, its built-in logic, or its security settings. This would include all Data Definition Language (DDL) statements and all security-related statements. In an SQL environment, it encompasses, but is not necessarily limited to:

CREATE
ALTER
DROP
GRANT
REVOKE

There may also be Data Manipulation Language (DML) statements that, subject to context, should be regarded as privileged. Possible examples in SQL include:

TRUNCATE TABLE;DELETE, or DELETE affecting more than n rows, for some n, or DELETE without a WHERE clause;

UPDATE or UPDATE affecting more than n rows, for some n, or UPDATE without a WHERE clause;

any SELECT, INSERT, UPDATE, or DELETE to an application-defined security table executed by other than a security principal.

Depending on the capabilities of PostgreSQL and the design of the database and associated applications, audit logging may be achieved by means of DBMS auditing features, database triggers, other mechanisms, or a combination of these.

Note: That it is particularly important to audit, and tightly control, any action that weakens the implementation of this requirement itself, since the objective is to have a complete audit trail of all administrative activity.

Check Content:
First, as the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Next, verify that role, read, write, and ddl auditing are enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain role, read, write, and ddl, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):
shared_preload_libraries = ‘pgaudit’
pgaudit.log='ddl, role, read, write'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72955
Group Title: SRG-APP-000494-DB-000345
Rule ID: SV-87607r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-005900
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to access categorized information (e.g., classification levels/security levels) occur.

Vulnerability Discussion: Changes in categorized information must be tracked. Without an audit trail, unauthorized access to protected data could go undetected.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

For detailed information on categorizing information, refer to FIPS Publication 199, Standards for Security Categorization of Federal Information and Information Systems, and FIPS Publication 200, Minimum Security Requirements for Federal Information and Information Systems.

Check Content:
First, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "SHOW pgaudit.log"

If pgaudit.log does not contain, "ddl, write, role", this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to access categories of information.

All denials are logged if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

With `pgaudit` installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log = 'ddl, write, role'

Now, as the system administrator, reload the server with the new configuration:

SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-$9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72957
Group Title: SRG-APP-000492-DB-000332
Rule ID: SV-87609r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006000
Rule Title: PostgreSQL must be able to generate audit records when security objects are accessed.

Vulnerability Discussion: Changes to the security configuration must be tracked.

This requirement applies to situations where security data is retrieved or modified via data manipulation operations, as opposed to via specialized security functionality.

In an SQL environment, types of access include, but are not necessarily limited to:

CREATE
SELECT
INSERT
UPDATE
DELETE
PREPARE
EXECUTE
ALTER
DROP

Check Content:
First, as the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Next, verify that role, read, write, and ddl auditing are enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain role, read, write, and ddl, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log='ddl, role, read, write'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72959
Group Title: SRG-APP-000499-DB-000330
Rule ID: SV-87611r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006100
Rule Title: PostgreSQL must generate audit records when privileges/permissions are deleted.

Vulnerability Discussion: Changes in the permissions, privileges, and roles granted to users and roles must be tracked. Without an audit trail, unauthorized elevation or restriction of privileges could go undetected. Elevated privileges give users access to information and functionality that they should not have; restricted privileges wrongly deny access to authorized users.

In an SQL environment, deleting permissions is typically done via the REVOKE command.

Check Content:
First, as the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Next, verify that role, read, write, and ddl auditing are enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain role, read, write, and ddl, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log = 'role'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72961
Group Title: SRG-APP-000506-DB-000353
Rule ID: SV-87613r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006200
Rule Title: PostgreSQL must generate audit records when concurrent logons/connections by the same user from different workstations occur.

Vulnerability Discussion: For completeness of forensic analysis, it is necessary to track who logs on to PostgreSQL.

Concurrent connections by the same user from multiple workstations may be valid use of the system; or such connections may be due to improper circumvention of the requirement to use the CAC for authentication; or they may indicate unauthorized account sharing; or they may be because an account has been compromised.

(If the fact of multiple, concurrent logons by a given user can be reliably reconstructed from the log entries for other events (logons/connections; voluntary and involuntary disconnections), then it is not mandatory to create additional log entries specifically for this.)

Check Content:
First, as the database administrator, verify that log_connections and log_disconnections are enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_connections"
$ psql -c "SHOW log_disconnections"

If either is off, this is a finding.

Next, verify that log_line_prefix contains sufficient information by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_line_prefix"

If log_line_prefix does not contain at least %m %u %d %c, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

First, as the database administrator (shown here as "postgres"), edit postgresql.conf:

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

Edit the following parameters as such:

log_connections = on
log_disconnections = on
log_line_prefix = '< %m %u %d %c: >'

Where:
* %m is the time and date
* %u is the username
* %d is the database
* %c is the session ID for the connection

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72963
Group Title: SRG-APP-000501-DB-000337
Rule ID: SV-87615r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006300
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to delete security objects occur.

Vulnerability Discussion: The removal of security objects from the database/PostgreSQL would seriously degrade a system's information assurance posture. If such an action is attempted, it must be logged.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Next, verify that role, read, write, and ddl auditing are enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain role, read, write, and ddl, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Configure PostgreSQL to produce audit records when unsuccessful attempts to delete security objects occur.

All errors and denials are logged if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log='ddl, role, read, write'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72965
Group Title: SRG-APP-000495-DB-000328
Rule ID: SV-87617r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006400
Rule Title: PostgreSQL must generate audit records when privileges/permissions are modified.

Vulnerability Discussion: Changes in the permissions, privileges, and roles granted to users and roles must be tracked. Without an audit trail, unauthorized elevation or restriction of privileges could go undetected. Elevated privileges give users access to information and functionality that they should not have; restricted privileges wrongly deny access to authorized users.

In an SQL environment, modifying permissions is typically done via the GRANT and REVOKE commands.

Check Content:
First, as the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Next, verify that role is enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain role, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log='role'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72969
Group Title: SRG-APP-000504-DB-000355
Rule ID: SV-87621r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006500
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to execute privileged activities or other system-level access occur.

Vulnerability Discussion: Without tracking privileged activity, it would be difficult to establish, correlate, and investigate the events relating to an incident or identify those responsible for one.

System documentation should include a definition of the functionality considered privileged.

A privileged function in this context is any operation that modifies the structure of the database, its built-in logic, or its security settings. This would include all Data Definition Language (DDL) statements and all security-related statements. In an SQL environment, it encompasses, but is not necessarily limited to:

CREATE
ALTER
DROP
GRANT
REVOKE

Note: That it is particularly important to audit, and tightly control, any action that weakens the implementation of this requirement itself, since the objective is to have a complete audit trail of all administrative activity.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

Check Content:
As the database administrator (shown here as "postgres"), create the role bob by running the following SQL:

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

Next, change the current role to bob and attempt to execute privileged activity:

```
$ psql -c "CREATE ROLE stig_test SUPERUSER"
$ psql -c "CREATE ROLE stig_test CREATEDB"
$ psql -c "CREATE ROLE stig_test CREATEROLE"
$ psql -c "CREATE ROLE stig_test CREATEUSER"
```

Now, as the database administrator (shown here as "postgres"), verify that an audit event was produced (use the latest log):

```
$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-02-23 20:16:32.396 EST postgres 56cfa74f.79eb postgres: >ERROR: must be superuser to create superusers
< 2016-02-23 20:16:32.396 EST postgres 56cfa74f.79eb postgres: >STATEMENT: CREATE ROLE stig_test SUPERUSER;
< 2016-02-23 20:16:48.725 EST postgres 56cfa74f.79eb postgres: >ERROR: permission denied to create role
< 2016-02-23 20:16:48.725 EST postgres 56cfa74f.79eb postgres: >STATEMENT: CREATE ROLE stig_test CREATEDB;
< 2016-02-23 20:16:54.365 EST postgres 56cfa74f.79eb postgres: >ERROR: permission denied to create role
< 2016-02-23 20:16:54.365 EST postgres 56cfa74f.79eb postgres: >STATEMENT: CREATE ROLE stig_test CREATEROLE;
< 2016-02-23 20:17:05.949 EST postgres 56cfa74f.79eb postgres: >ERROR: must be superuser to create superusers
< 2016-02-23 20:17:05.949 EST postgres 56cfa74f.79eb postgres: >STATEMENT: CREATE ROLE stig_test CREATEUSER;
```

If audit records are not produced, this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to execute privileged SQL.

All denials are logged by default if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000172

Group ID (Vulid): V-72971
Group Title: SRG-APP-000496-DB-000334
Rule ID: SV-87623r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006600
Rule Title: PostgreSQL must generate audit records when security objects are modified.

Vulnerability Discussion: Changes in the database objects (tables, views, procedures, functions) that record and control permissions, privileges, and roles granted to users and roles must be tracked. Without an audit trail, unauthorized changes to the security subsystem could go undetected. The database could be severely compromised or rendered inoperative.

Check Content:
First, as the database administrator, verify pgaudit is enabled by running the following SQL:

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

If the results does not contain `pgaudit`, this is a finding.

Next, verify that role, read, write, and ddl auditing are enabled:

```
$ psql -c "SHOW pgaudit.log"
```

If the output does not contain `role`, `read`, `write`, and `ddl`, this is a finding.

Next, verify that accessing the catalog is audited by running the following SQL:

```
$ psql -c "SHOW pgaudit.log_catalog"
```

If log_catalog is not `on`, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

Using `pgaudit` the DBMS (PostgreSQL) can be configured to audit these requests. See supplementary content `APPENDIX-B` for documentation on installing `pgaudit`.

With `pgaudit` installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log_catalog = 'on'
pgaudit.log='ddl, role, read, write'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72973
Group Title: SRG-APP-000498-DB-000346
Rule ID: SV-87625r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006700
Rule Title: PostgreSQL must generate audit records when categorized information (e.g., classification levels/security levels) is modified.

Vulnerability Discussion: Changes in categorized information must be tracked. Without an audit trail, unauthorized access to protected data could go undetected.

For detailed information on categorizing information, refer to FIPS Publication 199, Standards for Security Categorization of Federal Information and Information Systems, and FIPS Publication 200, Minimum Security Requirements for Federal Information and Information Systems.

Check Content:
If category tracking is not required in the database, this is not applicable.

First, as the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Next, verify that role, read, write, and ddl auditing are enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain role, read, write, and ddl, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log='ddl, role, read, write'

Now, as the system administrator, reload the server with the new configuration:

SYSTEMD SERVER ONLY

$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-72975
Group Title: SRG-APP-000495-DB-000329
Rule ID: SV-87627r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006800
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to modify privileges/permissions occur.

Vulnerability Discussion: Failed attempts to change the permissions, privileges, and roles granted to users and roles must be tracked. Without an audit trail, unauthorized attempts to elevate or restrict privileges could go undetected.

Modifying permissions is done via the GRANT and REVOKE commands.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

Check Content:
First, as the database administrator (shown here as "postgres"), create a role 'bob' and a test table by running the following SQL:

$ sudo su - postgres
$ psql -c "CREATE ROLE bob; CREATE TABLE test(id INT)"

Next, set current role to bob and attempt to modify privileges:

$ psql -c "SET ROLE bob; GRANT ALL PRIVILEGES ON test TO bob;"
$ psql -c "SET ROLE bob; REVOKE ALL PRIVILEGES ON test FROM bob"

Now, as the database administrator (shown here as "postgres"), verify the unsuccessful attempt was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
2016-07-14 18:12:23.208 EDT postgres postgres ERROR: permission denied for relation test
2016-07-14 18:12:23.208 EDT postgres postgres STATEMENT: GRANT ALL PRIVILEGES ON test TO bob;
2016-07-14 18:14:52.895 EDT postgres postgres ERROR: permission denied for relation test
2016-07-14 18:14:52.895 EDT postgres postgres STATEMENT: REVOKE ALL PRIVILEGES ON test FROM bob;

If audit logs are not generated when unsuccessful attempts to modify privileges/permissions occur, this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to modify privileges occur.

All denials are logged by default if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000172

Group ID (Vulid): V-72977
Group Title: SRG-APP-000495-DB-000327
Rule ID: SV-87629r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-006900
Rule Title: PostgreSQL must generate audit records when unsuccessful attempts to add privileges/permissions occur.

Vulnerability Discussion: Failed attempts to change the permissions, privileges, and roles granted to users and roles must be tracked. Without an audit trail, unauthorized attempts to elevate or restrict privileges could go undetected.

In an SQL environment, adding permissions is typically done via the GRANT command, or, in the negative, the REVOKE command.

To aid in diagnosis, it is necessary to keep track of failed attempts in addition to the successful ones.

Check Content:
First, as the database administrator (shown here as "postgres"), create a role 'bob' and a test table by running the following SQL:

$ sudo su - postgres
$ psql -c "CREATE ROLE bob; CREATE TABLE test(id INT)"

Next, set current role to bob and attempt to modify privileges:

$ psql -c "SET ROLE bob; GRANT ALL PRIVILEGES ON test TO bob;"

Now, as the database administrator (shown here as "postgres"), verify the unsuccessful attempt was logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>
2016-07-14 18:12:23.208 EDT postgres postgres ERROR: permission denied for relation test
2016-07-14 18:12:23.208 EDT postgres postgres STATEMENT: GRANT ALL PRIVILEGES ON test TO bob;

If audit logs are not generated when unsuccessful attempts to add privileges/permissions occur, this is a finding.

Fix Text: Configure PostgreSQL to produce audit records when unsuccessful attempts to add privileges occur.

All denials are logged by default if logging is enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000172

Group ID (Vulid): V-72979
Group Title: SRG-APP-000175-DB-000067
Rule ID: SV-87631r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-007000
Rule Title: PostgreSQL, when utilizing PKI-based authentication, must validate certificates by performing RFC 5280-compliant certification path validation.

Vulnerability Discussion: The DoD standard for authentication is DoD-approved PKI certificates.

A certificate's certification path is the path from the end entity certificate to a trusted root certification authority (CA). Certification path validation is necessary for a relying party to make an informed decision regarding acceptance of an end entity certificate. Certification path validation includes checks such as certificate issuer trust, time validity and revocation status for each certificate in the certification path. Revocation status information for CA and subject certificates in a certification path is commonly provided via certificate revocation lists (CRLs) or online certificate status protocol (OCSP) responses.

Database Management Systems that do not validate certificates by performing RFC 5280-compliant certification path validation are in danger of accepting certificates that are invalid and/or counterfeit. This could allow unauthorized access to the database.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To verify that a CRL file exists, as the database administrator (shown here as "postgres"), run the following:

$ sudo su - postgres
$ psql -c "SHOW ssl_crl_file"

If this is not set to a CRL file, this is a finding.

Next verify the existence of the CRL file by checking the directory set in postgresql.conf in the ssl_crl_file parameter from above:

Note: If no directory is specified, then the CRL file should be located in the same directory as postgresql.conf (PGDATA).

If the CRL file does not exist, this is a finding.

Next, verify that hostssl entries in pg_hba.conf have "cert" and "clientcert=1" enabled:

$ sudo su - postgres
$ grep hostssl ${PGDATA?}/postgresql.conf

If hostssl entries does not contain cert or clientcert=1, this is a finding.

If certificates are not being validated by performing RFC 5280-compliant certification path validation, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To configure PostgreSQL to use SSL, see supplementary content APPENDIX-G.

To generate a Certificate Revocation List, see the official Red Hat Documentation: https://access.redhat.com/documentation/en-US/Red_Hat_Update_Infrastructure/2.1/html/Administration_Guide/chap-Red_Hat_Update_Infrastructure-Administration_Guide-Certification_Revocation_List_CRL.html

As the database administrator (shown here as "postgres"), copy the CRL file into the data directory:

First, as the system administrator, copy the CRL file into the PostgreSQL Data Directory:

$ sudo cp root.crl ${PGDATA?}/root.crl

As the database administrator (shown here as "postgres"), set the ssl_crl_file parameter to the filename of the CRL:

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
ssl_crl_file = 'root.crl'

Next, in pg_hba.conf, require ssl authentication:

$ sudo su - postgres
$ vi ${PGDATA?}/pg_hba.conf
hostssl <database> <user> <address> cert clientcert=1

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000185

Group ID (Vulid): V-73123
Group Title: SRG-APP-000097-DB-000041
Rule ID: SV-87775r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-007100
Rule Title: PostgreSQL must produce audit records containing sufficient information to establish where the events occurred.

Vulnerability Discussion: Information system auditing capability is critical for accurate forensic analysis. Without establishing where events occurred, it is impossible to establish, correlate, and investigate the events relating to an incident.

In order to compile an accurate risk assessment and provide forensic analysis, it is essential for security personnel to know where events occurred, such as application components, modules, session identifiers, filenames, host names, and functionality.

Associating information about where the event occurred within the application provides a means of investigating an attack; recognizing resource utilization or capacity thresholds; or identifying an improperly configured application.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), check the current log_line_prefix setting by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_line_prefix"

If log_line_prefix does not contain %m %u %d %s, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To check that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

First edit the postgresql.conf file as the database administrator (shown here as "postgres"):

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

Extra parameters can be added to the setting log_line_prefix to log application related information:

%a = application name
%u = user name
%d = database name
%r = remote host and port
%p = process ID
%m = timestamp with milliseconds
%i = command tag
%s = session startup

%e = SQL state

For example:

log_line_prefix = '< %m %a %u %d %r %p %i %e %s>'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000132

Group ID (Vulid): V-72981
Group Title: SRG-APP-000441-DB-000378
Rule ID: SV-87633r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-007200
Rule Title: PostgreSQL must maintain the confidentiality and integrity of information during preparation for transmission.

Vulnerability Discussion: Information can be either unintentionally or maliciously disclosed or modified during preparation for transmission, including, for example, during aggregation, at protocol transformation points, and during packing/unpacking. These unauthorized disclosures or modifications compromise the confidentiality or integrity of the information.

Use of this requirement will be limited to situations where the data owner has a strict requirement for ensuring data integrity and confidentiality is maintained at every step of the data transfer and handling process.

When transmitting data, PostgreSQL, associated applications, and infrastructure must leverage transmission protection mechanisms.

PostgreSQL uses OpenSSL SSLv23_method() in fe-secure-openssl.c, while the name is misleading, this function enables only TLS encryption methods, not SSL.

See OpenSSL: https://mta.openssl.org/pipermail/openssl-dev/2015-May/001449.html

Check Content:
If the data owner does not have a strict requirement for ensuring data integrity and confidentiality is maintained at every step of the data transfer and handling process, this is not a finding.

As the database administrator (shown here as "postgres"), verify SSL is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW ssl"

If SSL is not enabled, this is a finding.

If PostgreSQL does not employ protective measures against unauthorized disclosure and modification during preparation for transmission, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Implement protective measures against unauthorized disclosure and modification during preparation for transmission.

To configure PostgreSQL to use SSL, as a database administrator (shown here as "postgres"), edit postgresql.conf:

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

Add the following parameter:

ssl = on

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

For more information on configuring PostgreSQL to use SSL, see supplementary content APPENDIX-G.

CCI: CCI-002420

Group ID (Vulid): V-72983
Group Title: SRG-APP-000089-DB-000064
Rule ID: SV-87635r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-007400
Rule Title: PostgreSQL must provide audit record generation capability for DoD-defined auditable events within all DBMS/database components.

Vulnerability Discussion: Without the capability to generate audit records, it would be difficult to establish, correlate, and investigate the events relating to an incident or identify those responsible for one.

Audit records can be generated from various components within PostgreSQL (e.g., process, module). Certain specific application functionalities may be audited as well. The list of audited events is the set of events for which audits are to be generated. This set of events is typically a subset of the list of all events for which the system is capable of generating audit records.

DoD has defined the list of events for which PostgreSQL will provide an audit record generation capability as the following:

(i) Successful and unsuccessful attempts to access, modify, or delete privileges, security objects, security levels, or categories of information (e.g., classification levels);
(ii) Access actions, such as successful and unsuccessful logon attempts, privileged activities, or other system-level access, starting and ending time for user access to the system, concurrent logons from different workstations, successful and unsuccessful accesses to objects, all program initiations, and all direct access to the information system; and
(iii) All account creation, modification, disabling, and termination actions.

Organizations may define additional events requiring continuous or ad hoc auditing.

Check Content:
Check PostgreSQL auditing to determine whether organization-defined auditable events are being audited by the system.

If organization-defined auditable events are not being audited, this is a finding.

Fix Text: Configure PostgreSQL to generate audit records for at least the DoD minimum set of events.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

CCI: CCI-000169

Group ID (Vulid): V-72985
Group Title: SRG-APP-000375-DB-000323
Rule ID: SV-87637r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-007700
Rule Title: PostgreSQL must generate time stamps, for audit records and application data, with a minimum granularity of one second.

Vulnerability Discussion: Without sufficient granularity of time stamps, it is not possible to adequately determine the chronological order of records.

Time stamps generated by PostgreSQL must include date and time. Granularity of time measurements refers to the precision available in time stamp values. Granularity coarser than one second is not sufficient for audit trail purposes. Time stamp values are typically presented with three or more decimal places of seconds; however, the actual granularity may be coarser than the apparent precision. For example, PostgreSQL will always return at least millisecond timestamps but it can be truncated using EXTRACT functions: SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

First, as the database administrator (shown here as "postgres"), verify the current log_line_prefix setting by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_line_prefix"

If log_line_prefix does not contain %m, this is a finding.

Next check the logs to verify time stamps are being logged:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_log/<latest_log>

< 2016-02-23 12:53:33.947 EDT postgres postgres 570bd68d.3912 >LOG: connection authorized: user=postgres database=postgres
< 2016-02-23 12:53:41.576 EDT postgres postgres 570bd68d.3912 >LOG: AUDIT: SESSION,1,1,DDL,CREATE TABLE,,,CREATE TABLE test_srg(id INT);,<none>
< 2016-02-23 12:53:44.372 EDT postgres postgres 570bd68d.3912 >LOG: disconnection: session time: 0:00:10.426 user=postgres database=postgres host=[local]

If time stamps are not being logged, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

PostgreSQL will not log anything if logging is not enabled. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

If logging is enabled the following configurations must be made to log events with time stamps:

First, as the database administrator (shown here as "postgres"), edit postgresql.conf:

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

Add %m to log_line_prefix to enable time stamps with milliseconds:

log_line_prefix = '< %m >'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-001889

Group ID (Vulid): V-72987
Group Title: SRG-APP-000100-DB-000201
Rule ID: SV-87639r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-007800
Rule Title: PostgreSQL must produce audit records containing sufficient information to establish the identity of any user/subject or process associated with the event.

Vulnerability Discussion: Information system auditing capability is critical for accurate forensic analysis. Without information that establishes the identity of the subjects (i.e., users or processes acting on behalf of users) associated with the events, security personnel cannot determine responsibility for the potentially harmful event.

Identifiers (if authenticated or otherwise known) include, but are not limited to, user database tables, primary key values, user names, or process identifiers.

1) Linux's sudo and su feature enables a user (with sufficient OS privileges) to emulate another user, and it is the identity of the emulated user that is seen by PostgreSQL and logged in the audit trail. Therefore, care must be taken (outside of Postgresql) to restrict sudo/su to the minimum set of users necessary.

2) PostgreSQL's SET ROLE feature enables a user (with sufficient PostgreSQL privileges) to emulate another user running statements under the permission set of the emulated user. In this case, it is the emulating user's identity, and not that of the emulated user, that gets logged in the audit trail. While this is definitely better than the other way around, ideally, both identities would be recorded.

Check Content:
Check PostgreSQL settings and existing audit records to verify a user name associated with the event is being captured and stored with the audit records. If audit records exist without specific user information, this is a finding.

First, as the database administrator (shown here as "postgres"), verify the current setting of log_line_prefix by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_line_prefix"

If log_line_prefix does not contain %m, %u, %d, %p, %r, %a, this is a finding.

Fix Text: Logging must be enabled in order to capture the identity of any user/subject or process associated with an event. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

To enable username, database name, process ID, remote host/port and application name in logging, as the database administrator (shown here as "postgres"), edit the following in postgresql.conf:

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
log_line_prefix = '< %m %u %d %p %r %a >'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-001487

Group ID (Vulid): V-72989
Group Title: SRG-APP-000514-DB-000381
Rule ID: SV-87641r1_rule
Severity: CAT I
Rule Version (STIG-ID): PGS9-00-008000
Rule Title: PostgreSQL must implement NIST FIPS 140-2 validated cryptographic modules to generate and validate cryptographic hashes.

Vulnerability Discussion: Use of weak or untested encryption algorithms undermines the purposes of utilizing encryption to protect data. The 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.

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.

Check Content:
First, as the system administrator, run the following to see if FIPS is enabled:

$ cat /proc/sys/crypto/fips_enabled

If fips_enabled is not 1, this is a finding.

Fix Text: Configure OpenSSL to be FIPS compliant.

PostgreSQL uses OpenSSL for cryptographic modules. To configure OpenSSL to be FIPS 140-2 compliant, see the official RHEL Documentation:
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/sect-Security_Guide-Federal_Standards_And_Regulations-Federal_Information_Processing_Standard.html

For more information on configuring PostgreSQL to use SSL, see supplementary content APPENDIX-G.

CCI: CCI-002450

Group ID (Vulid): V-72991
Group Title: SRG-APP-000416-DB-000380
Rule ID: SV-87643r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-008100
Rule Title: PostgreSQL must use NSA-approved cryptography to protect classified information in accordance with the data owners requirements.

Vulnerability Discussion: Use of weak or untested encryption algorithms undermines the purposes of utilizing encryption to protect data. The 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.

NSA-approved cryptography for classified networks is hardware based. This requirement addresses the compatibility of PostgreSQL with the encryption devices.

Check Content:
If PostgreSQL is deployed in an unclassified environment, this is not applicable (NA).

If PostgreSQL is not using NSA-approved cryptography to protect classified information in accordance with applicable federal laws, Executive Orders, directives, policies, regulations, and standards, this is a finding.

To check if PostgreSQL is configured to use SSL, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "SHOW ssl"

If SSL is off, this is a finding.

Consult network administration staff to determine whether the server is protected by NSA-approved encrypting devices. If not, this a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To configure PostgreSQL to use SSL, as a database administrator (shown here as "postgres"), edit postgresql.conf:

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

Add the following parameter:

ssl = on

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

For more information on configuring PostgreSQL to use SSL, see supplementary content APPENDIX-G.

Deploy NSA-approved encrypting devices to protect the server on the network.

CCI: CCI-002450

Group ID (Vulid): V-72993
Group Title: SRG-APP-000514-DB-000383
Rule ID: SV-87645r1_rule
Severity: CAT I
Rule Version (STIG-ID): PGS9-00-008200
Rule Title: PostgreSQL must implement NIST FIPS 140-2 validated cryptographic modules to protect unclassified information requiring confidentiality and cryptographic protection, in accordance with the data owners requirements.

Vulnerability Discussion: Use of weak or untested encryption algorithms undermines the purposes of utilizing encryption to protect data. The 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.

Check Content:
First, as the system administrator, run the following to see if FIPS is enabled:

$ cat /proc/sys/crypto/fips_enabled

If fips_enabled is not 1, this is a finding.

Fix Text: Configure OpenSSL to be FIPS compliant.

PostgreSQL uses OpenSSL for cryptographic modules. To configure OpenSSL to be FIPS 140-2 compliant, see the official RHEL Documentation:
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/sect-Security_Guide-Federal_Standards_And_Regulations-Federal_Information_Processing_Standard.html

For more information on configuring PostgreSQL to use SSL, see supplementary content APPENDIX-G.

CCI: CCI-002450

Group ID (Vulid): V-72995
Group Title: SRG-APP-000231-DB-000154
Rule ID: SV-87647r1_rule

Severity: CAT II
Rule Version (STIG-ID): PGS9-00-008300
Rule Title: PostgreSQL must protect the confidentiality and integrity of all information at rest.

Vulnerability Discussion: This control is intended to address the confidentiality and integrity of information at rest in non-mobile devices and covers user information and system information. 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. Applications and application users generate information throughout the course of their application use.

User data generated, as well as application-specific configuration data, needs to be protected. Organizations may choose to employ different mechanisms to achieve confidentiality and integrity protections, as appropriate.

If the confidentiality and integrity of application data is not protected, the data will be open to compromise and unauthorized modification.

Check Content:
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 (shown here as "postgres"), run the following command:

$ sudo su - postgres
$ psql -c "SELECT * FROM pg_available_extensions where name='pgcrypto'"

If data in the database requires encryption and pgcrypto is not available, this is a finding.

If disk or filesystem requires encryption, ask the system owner, DBA, and SA to demonstrate the use of disk-level encryption. If this is required and is not found, this is a finding.

If controls do not exist or are not enabled, this is a finding.

Fix Text: Apply appropriate controls to protect the confidentiality and integrity of data at rest in the database.

The pgcrypto module provides cryptographic functions for PostgreSQL. See supplementary content APPENDIX-E for documentation on installing pgcrypto.

With pgcrypto installed, it is possible to insert encrypted data into the database:

INSERT INTO accounts(username, password) VALUES ('bob', crypt('a_secure_password', gen_salt('xdes')));

CCI: CCI-001199

Group ID (Vulid): V-72997
Group Title: SRG-APP-000378-DB-000365
Rule ID: SV-87649r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-008400
Rule Title: PostgreSQL must prohibit user installation of logic modules (functions, trigger procedures, views, etc.) without explicit privileged status.

Vulnerability Discussion: Allowing regular users to install software, without explicit privileges, creates the risk that untested or potentially malicious software will be installed on the system. Explicit privileges (escalated or administrative privileges) provide the regular user with explicit capabilities and control that exceed the rights of a regular user.

PostgreSQL functionality and the nature and requirements of databases will vary; so while users are not permitted to install unapproved software, there may be instances where the organization allows the user to install approved software packages such as from an approved software repository. The requirements for production servers will be more restrictive than those used for development and research.

PostgreSQL must enforce software installation by users based upon what types of software installations are permitted (e.g., updates and security patches to existing software) and what types of installations are prohibited (e.g., software whose pedigree with regard to being potentially malicious is unknown or suspect) by the organization).

In the case of a database management system, this requirement covers stored procedures, functions, triggers, views, etc.

Check Content:
If PostgreSQL supports only software development, experimentation and/or developer-level testing (that is, excluding production systems, integration testing, stress testing, and user acceptance testing), this is not a finding.

Review PostgreSQL and database security settings with respect to non-administrative users' ability to create, alter, or replace logic modules, to include but not necessarily only stored procedures, functions, triggers, and views.

To list the privileges for all tables and schemas, as the database administrator (shown here as "postgres"), run the following:

$ sudo su - postgres
$ psql -c "\dp"
$ psql -c "\dn+"

The privileges are as follows:

rolename=xxxx -- privileges granted to a role
=xxxx -- privileges granted to PUBLIC

r -- SELECT ("read")
w -- UPDATE ("write")
a -- INSERT ("append")
d -- DELETE
D -- TRUNCATE
x -- REFERENCES
t -- TRIGGER
X -- EXECUTE
U -- USAGE
C -- CREATE
c -- CONNECT
T -- TEMPORARY
arwdDxt -- ALL PRIVILEGES (for tables, varies for other objects)
* -- grant option for preceding privilege

/yyyy -- role that granted this privilege

If any such permissions exist and are not documented and approved, this is a finding.

Fix Text: Document and obtain approval for any non-administrative users who require the ability to create, alter or replace logic modules.

Implement the approved permissions. Revoke any unapproved permissions.

CCI: CCI-001812

Group ID (Vulid): V-72999
Group Title: SRG-APP-000211-DB-000122
Rule ID: SV-87651r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-008500
Rule Title: PostgreSQL must separate user functionality (including user interface services) from database management functionality.

Vulnerability Discussion: Information system management functionality includes functions necessary to administer databases, network components, workstations, or servers and typically requires privileged user access.

The separation of user functionality from information system management functionality is either physical or logical and is accomplished by using different computers, different central processing units, different instances of the operating system, different network addresses, combinations of these methods, or other methods, as appropriate.

An example of this type of separation is observed in web administrative interfaces that use separate authentication methods for users of any other information system resources.

This may include isolating the administrative interface on a different domain and with additional access controls.

If administrative functionality or information regarding PostgreSQL management is presented on an interface available for users, information on DBMS settings may be inadvertently made available to the user.

Check Content:
Check PostgreSQL settings and vendor documentation to verify that administrative functionality is separate from user functionality.

As the database administrator (shown here as "postgres"), list all roles and permissions for the database:

$ sudo su - postgres
$ psql -c "\du"

If any non-administrative role has the attribute "Superuser", "Create role", "Create DB" or "Bypass RLS", this is a finding.

If administrator and general user functionality are not separated either physically or logically, this is a finding.

Fix Text: Configure PostgreSQL to separate database administration and general user functionality.

Do not grant superuser, create role, create db or bypass rls role attributes to users that do not require it.

To remove privileges, see the following example:

ALTER ROLE <username> NOSUPERUSER NOCREATEDB NOCREATEROLE NOBYPASSRLS;

CCI: CCI-001082

Group ID (Vulid): V-73001
Group Title: SRG-APP-000092-DB-000208
Rule ID: SV-87653r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-008600
Rule Title: PostgreSQL must initiate session auditing upon startup.

Vulnerability Discussion: Session auditing is for use when a user's activities are under investigation. To be sure of capturing all activity during those periods when session auditing is in use, it needs to be in operation for the whole time PostgreSQL is running.

Check Content:
As the database administrator (shown here as "postgres"), check the current settings by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

If pgaudit is not in the current setting, this is a finding.

As the database administrator (shown here as "postgres"), check the current settings by running the following SQL:

$ psql -c "SHOW logging_destination"

If stderr or syslog are not in the current setting, this is a finding.

Fix Text: Configure PostgreSQL to enable auditing.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

For session logging we suggest using pgaudit. For instructions on how to setup pgaudit, see supplementary content APPENDIX-B.

CCI: CCI-001464

Group ID (Vulid): V-73003
Group Title: SRG-APP-000428-DB-000386
Rule ID: SV-87655r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-008700
Rule Title: PostgreSQL must implement cryptographic mechanisms to prevent unauthorized modification of organization-defined information at rest (to include, at a minimum, PII and classified information) on organization-defined information system components.

Vulnerability Discussion: PostgreSQLs handling data requiring "data at rest" protections must employ cryptographic mechanisms to prevent unauthorized disclosure and modification of the information at rest. These cryptographic mechanisms may be native to PostgreSQL or implemented via additional software or operating system/file system settings, as appropriate to the situation.

Selection of a cryptographic mechanism is based on the need to protect the integrity of organizational information. The strength of the mechanism is commensurate with the security category and/or classification of the information. Organizations have the flexibility to either encrypt all information on storage devices (i.e., full disk encryption) or encrypt specific data structures (e.g., files, records, or fields).

The decision whether and what to encrypt rests with the data owner and is also influenced by the physical measures taken to secure the equipment and media on which the information resides.

Check Content:
Review the system documentation to determine whether the organization has defined the information at rest that is to be protected from modification, which must include, at a minimum, PII and classified information.

If no information is identified as requiring such protection, this is not a finding.

Review the configuration of PostgreSQL, operating system/file system, and additional software as relevant.

If any of the information defined as requiring cryptographic protection from modification is not encrypted in a manner that provides the required level of protection, this is a finding.

One possible way to encrypt data within PostgreSQL is to use pgcrypto extension.

To check if pgcrypto is installed on PostgreSQL, as a database administrator (shown here as "postgres"), run the following command:

$ sudo su - postgres
$ psql -c "SELECT * FROM pg_available_extensions where name='pgcrypto'"

If data in the database requires encryption and pgcrypto is not available, this is a finding.

If disk or filesystem requires encryption, ask the system owner, DBA, and SA to demonstrate filesystem or disk level encryption.

If this is required and is not found, this is a finding.

Fix Text: Configure PostgreSQL, operating system/file system, and additional software as relevant, to provide the required level of cryptographic protection.

The pgcrypto module provides cryptographic functions for PostgreSQL. See supplementary content APPENDIX-E for documentation on installing pgcrypto.

With pgcrypto installed, it's possible to insert encrypted data into the database:

INSERT INTO accounts(username, password) VALUES ('bob', crypt('a_secure_password', gen_salt('md5')));

CCI: CCI-002475

Group ID (Vulid): V-73005
Group Title: SRG-APP-000098-DB-000042
Rule ID: SV-87657r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-008800
Rule Title: PostgreSQL must produce audit records containing sufficient information to establish the sources (origins) of the events.

Vulnerability Discussion: Information system auditing capability is critical for accurate forensic analysis. Without establishing the source of the event, it is impossible to establish, correlate, and investigate the events relating to an incident.

In order to compile an accurate risk assessment and provide forensic analysis, it is essential for security personnel to know where events occurred, such as application components, modules, session identifiers, filenames, host names, and functionality.

In addition to logging where events occur within the application, the application must also produce audit records that identify the application itself as the source of the event.

Associating information about the source of the event within the application provides a means of investigating an attack; recognizing resource utilization or capacity thresholds; or identifying an improperly configured application.

Check Content:
Check PostgreSQL settings and existing audit records to verify information specific to the source (origin) of the event is being captured and stored with audit records.

As the database administrator (usually postgres, check the current log_line_prefix and "log_hostname" setting by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_line_prefix"
$ psql -c "SHOW log_hostname"

For a complete list of extra information that can be added to log_line_prefix, see the official documentation: https://www.postgresql.org/docs/current/static/runtime-config-logging.html#GUC-LOG-LINE-PREFIX

If the current settings do not provide enough information regarding the source of the event, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

If logging is enabled the following configurations can be made to log the source of an event.

First, as the database administrator, edit postgresql.conf:

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

Log Line Prefix

Extra parameters can be added to the setting log_line_prefix to log source of event:

%a = application name
%u = user name
%d = database name
%r = remote host and port
%p = process ID
%m = timestamp with milliseconds

For example:
log_line_prefix = '< %m %a %u %d %r %p %m >'

Log Hostname

By default only IP address is logged. To also log the hostname the following parameter can also be set in postgresql.conf:

log_hostname = on

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000133

Group ID (Vulid): V-73007
Group Title: SRG-APP-000141-DB-000091
Rule ID: SV-87659r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-008900
Rule Title: Unused database components, PostgreSQL software, and database objects must be removed.

Vulnerability Discussion: Information systems are capable of providing a wide variety of functions and services. Some of the functions and services, provided by default, may not be necessary to support essential organizational operations (e.g., key missions, functions).

It is detrimental for software products to provide, or install by default, functionality exceeding requirements or mission objectives.

PostgreSQLs must adhere to the principles of least functionality by providing only essential capabilities.

Check Content:
To get a list of all extensions installed, use the following commands:

$ sudo su - postgres
$ psql -c "select * from pg_extension where extname != 'plpgsql';""

If any extensions exist that are not approved, this is a finding.

Fix Text: To remove extensions, use the following commands:

$ sudo su - postgres
$ psql -c "DROP EXTENSION <extension_name>"

Note: it is recommended that plpgsql not be removed.

CCI: CCI-000381

Group ID (Vulid): V-73009
Group Title: SRG-APP-000141-DB-000093
Rule ID: SV-87661r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-009100
Rule Title: Access to external executables must be disabled or restricted.

Vulnerability Discussion: Information systems are capable of providing a wide variety of functions and services. Some of the functions and services, provided by default, may not be necessary to support essential organizational operations (e.g., key missions, functions).

It is detrimental for applications to provide, or install by default, functionality exceeding requirements or mission objectives.

Applications must adhere to the principles of least functionality by providing only essential capabilities.

PostgreSQLs may spawn additional external processes to execute procedures that are defined in PostgreSQL but stored in external host files (external procedures). The spawned process used to execute the external procedure may operate within a different OS security context than PostgreSQL and provide unauthorized access to the host system.

Check Content:
PostgreSQL's Copy command can interact with the underlying OS. Only superuser has access to this command.

First, as the database administrator (shown here as "postgres"), run the following SQL to list all roles and their privileges:

$ sudo su - postgres
$ psql -x -c "\du"

If any role has "superuser" that should not, this is a finding.

It is possible for an extension to contain code that could access external executables via SQL. To list all installed extensions, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -x -c "SELECT * FROM pg_available_extensions WHERE installed_version IS NOT NULL"

If any extensions are installed that are not approved, this is a finding.

Fix Text: To remove superuser from a role, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "ALTER ROLE <role-name> WITH NOSUPERUSER"

To remove extensions from PostgreSQL, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "DROP EXTENSION extension_name"

CCI: CCI-000381

Group ID (Vulid): V-73011
Group Title: SRG-APP-000141-DB-000092
Rule ID: SV-87663r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-009200
Rule Title: Unused database components which are integrated in PostgreSQL and cannot be uninstalled must be disabled.

Vulnerability Discussion: Information systems are capable of providing a wide variety of functions and services. Some of the functions and services, provided by default, may not be necessary to support essential organizational operations (e.g., key missions, functions).

It is detrimental for software products to provide, or install by default, functionality exceeding requirements or mission objectives.

PostgreSQLs must adhere to the principles of least functionality by providing only essential capabilities.

Unused, unnecessary PostgreSQL components increase the attack vector for PostgreSQL by introducing additional targets for attack. By minimizing the services and applications installed on the system, the number of potential vulnerabilities is reduced. Components of the system that are unused and cannot be uninstalled must be disabled. The techniques available for disabling components will vary by DBMS product, OS and the nature of the component and may include DBMS configuration settings, OS service settings, OS file access security, and DBMS user/role permissions.

Check Content:
To list all installed packages, as the system administrator, run the following:

# RHEL/CENT Systems
$ sudo yum list installed | grep postgres

# Debian Systems
$ dpkg --get-selections | grep postgres

If any packages are installed that are not required, this is a finding.

Fix Text: To remove any unneeded executables, as the system administrator, run the following:

# RHEL/CENT Systems
$ sudo yum erase <package_name>

# Debian Systems
$ sudo apt-get remove <package_name>

CCI: CCI-000381

Group ID (Vulid): V-73013
Group Title: SRG-APP-000313-DB-000309
Rule ID: SV-87665r1_rule

Severity: CAT II
Rule Version (STIG-ID): PGS9-00-009400
Rule Title: PostgreSQL must associate organization-defined types of security labels having organization-defined security label values with information in process.

Vulnerability Discussion: Without the association of security labels to information, there is no basis for PostgreSQL to make security-related access-control decisions.

Security labels are abstractions representing the basic properties or characteristics of an entity (e.g., subjects and objects) with respect to safeguarding information.

These labels are typically associated with internal data structures (e.g., tables, rows) within the database and are used to enable the implementation of access control and flow control policies, reflect special dissemination, handling or distribution instructions, or support other aspects of the information security policy.

One example includes marking data as classified or FOUO. These security labels may be assigned manually or during data processing, but, either way, it is imperative these assignments are maintained while the data is in storage. If the security labels are lost when the data is stored, there is the risk of a data compromise.

The mechanism used to support security labeling may be the sepgsql feature of PostgreSQL, a third-party product, or custom application code.

Check Content:
If security labeling is not required, this is not a finding.

First, as the database administrator (shown here as "postgres"), run the following SQL against each table that requires security labels:

$ sudo su - postgres
$ psql -c "\d+ <schema_name>.<table_name>"

If security labeling requirements have been specified, but the security labeling is not implemented or does not reliably maintain labels on information in process, this is a finding.

Fix Text: 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 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).

RLS policies can be very different depending on their use case. For one example of using RLS for Security Labels, see supplementary content APPENDIX-D.

CCI: CCI-002263

Group ID (Vulid): V-73015
Group Title: SRG-APP-000171-DB-000074
Rule ID: SV-87667r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-009500
Rule Title: If passwords are used for authentication, PostgreSQL must store only hashed, salted representations of passwords.

Vulnerability Discussion: The DoD standard for authentication is DoD-approved PKI certificates.

Authentication based on User ID and Password may be used only when it is not possible to employ a PKI certificate, and requires AO approval.

In such cases, database passwords stored in clear text, using reversible encryption, or using unsalted hashes would be vulnerable to unauthorized disclosure. Database passwords must always be in the form of one-way, salted hashes when stored internally or externally to PostgreSQL.

Check Content:
To check if password encryption is enabled, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "SHOW password_encryption"

If password_encryption is not on, this is a finding.

Next, to identify if any passwords have been stored without being hashed and salted, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -x -c "SELECT * FROM pg_shadow"

If any password is in plaintext, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To enable password_encryption, as the database administrator, edit postgresql.conf:

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
password_encryption = on

Institute a policy of not using the "WITH UNENCRYPTED PASSWORD" option with the CREATE ROLE/USER and ALTER ROLE/USER commands. (This option overrides the setting of the password_encryption configuration parameter.)

As the system administrator, restart the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl restart postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 restart

CCI: CCI-000196

Group ID (Vulid): V-73017
Group Title: SRG-APP-000380-DB-000360
Rule ID: SV-87669r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-009600
Rule Title: PostgreSQL must enforce access restrictions associated with changes to the configuration of PostgreSQL or database(s).

Vulnerability Discussion: Failure to provide logical access restrictions associated with changes to configuration may have significant effects on the overall security of the system.

When dealing with access restrictions pertaining to change control, it should be noted that any changes to the hardware, software, and/or firmware components of the information system can potentially have significant effects on the overall security of the system.

Accordingly, only qualified and authorized individuals should be allowed to obtain access to system components for the purposes of initiating changes, including upgrades and modifications.

Check Content:
To list all the permissions of individual roles, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "\du

If any role has SUPERUSER that should not, this is a finding.

Next, list all the permissions of databases and schemas by running the following SQL:

$ sudo su - postgres
$ psql -c "\l"
$ psql -c "\dn+"

If any database or schema has update ("W") or create ("C") privileges and should not, this is a finding.

Fix Text: Configure PostgreSQL to enforce access restrictions associated with changes to the configuration of PostgreSQL or database(s).

Use ALTER ROLE to remove accesses from roles:

$ psql -c "ALTER ROLE <role_name> NOSUPERUSER"

Use REVOKE to remove privileges from databases and schemas:

$ psql -c "REVOKE ALL PRIVILEGES ON <table> FROM <role_name>;

CCI: CCI-001813

Group ID (Vulid): V-73019
Group Title: SRG-APP-000080-DB-000063
Rule ID: SV-87671r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-009700
Rule Title: PostgreSQL must protect against a user falsely repudiating having performed organization-defined actions.

Vulnerability Discussion: Non-repudiation of actions taken is required in order to maintain data integrity. Examples of particular actions taken by

individuals include creating information, sending a message, approving information (e.g., indicating concurrence or signing a contract), and receiving a message.

Non-repudiation protects against later claims by a user of not having created, modified, or deleted a particular data item or collection of data in the database.

In designing a database, the organization must define the types of data and the user actions that must be protected from repudiation. The implementation must then include building audit features into the application data tables, and configuring PostgreSQL' audit tools to capture the necessary audit trail. Design and implementation also must ensure that applications pass individual user identification to PostgreSQL, even where the application connects to PostgreSQL with a standard, shared account.

Check Content:
First, as the database administrator, review the current log_line_prefix settings by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_line_prefix"

If log_line_prefix does not contain at least '< %m %a %u %d %r %p %m >', this is a finding.

Next, review the current shared_preload_libraries' settings by running the following SQL:

$ psql -c "SHOW shared_preload_libraries"

If shared_preload_libraries does not contain "pgaudit", this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Configure the database to supply additional auditing information to protect against a user falsely repudiating having performed organization-defined actions.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

Modify the configuration of audit logs to include details identifying the individual user:

First, as the database administrator (shown here as "postgres"), edit postgresql.conf:

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

Extra parameters can be added to the setting log_line_prefix to identify the user:

log_line_prefix = '< %m %a %u %d %r %p %m >'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

Use accounts assigned to individual users. Where the application connects to PostgreSQL using a standard, shared account, ensure that it also captures the individual user identification and passes it to PostgreSQL.

CCI: CCI-000166

Group ID (Vulid): V-73021
Group Title: SRG-APP-000093-DB-000052
Rule ID: SV-87673r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-009800
Rule Title: PostgreSQL must provide the capability for authorized users to capture, record, and log all content related to a user session.

Vulnerability Discussion: Without the capability to capture, record, and log all content related to a user session, investigations into suspicious user activity would be hampered.

Typically, this PostgreSQL capability would be used in conjunction with comparable monitoring of a user's online session, involving other software components such as operating systems, web servers and front-end user applications. The current requirement, however, deals specifically with PostgreSQL.

Check Content:

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

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

If shared_preload_libraries does not contain pgaudit, this is a finding.

Next, to verify connections and disconnections are logged, run the following SQL:

$ psql -c "SHOW log_connections"
$ psql -c "SHOW log_disconnections"

If log_connections and log_disconnections are off, this is a finding.

Now, to verify that pgaudit is configured to log, run the following SQL:

$ psql -c "SHOW pgaudit.log"

If pgaudit.log does not contain ddl, role, read, write, this is a finding.

Fix Text: Configure the database capture, record, and log all content related to a user session.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

With logging enabled, as the database administrator (shown here as "postgres"), enable log_connections and log_disconnections:

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
log_connections = on
log_disconnections = on

Using pgaudit PostgreSQL can be configured to audit activity. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed, as a database administrator (shown here as "postgres"), enable which objects required for auditing a user's session:

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
pgaudit.log = 'write, ddl, role, read, function';
pgaudit.log_relation = on;

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-001462

Group ID (Vulid): V-73023
Group Title: SRG-APP-000359-DB-000319
Rule ID: SV-87675r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-009900
Rule Title: The system must provide a warning to appropriate support staff when allocated audit record storage volume reaches 75% of maximum audit record storage capacity.

Vulnerability Discussion: Organizations are required to use a central log management system, so, under normal conditions, the audit space allocated to PostgreSQL on its own server will not be an issue. However, space will still be required on PostgreSQL server for audit records in transit, and, under abnormal conditions, this could fill up. Since a requirement exists to halt processing upon audit failure, a service outage would result.

If support personnel are not notified immediately upon storage volume utilization reaching 75%, they are unable to plan for storage capacity expansion.

The appropriate support staff include, at a minimum, the ISSO and the DBA/SA.

Check Content:
Review system configuration.

If no script/tool is monitoring the partition for the PostgreSQL log directories, this is a finding.

If appropriate support staff are not notified immediately upon storage volume utilization reaching 75%, this is a finding.

Fix Text: Configure the system to notify appropriate support staff immediately upon storage volume utilization reaching 75%.

PostgreSQL does not monitor storage, however, it is possible to monitor storage with a script.

Example Monitoring Script

```
#!/bin/bash

PGDATA=/var/lib/psql/9.5/data
CURRENT=$(df ${PGDATA?} | grep / | awk '{ print $5}' | sed 's/%//g')
THRESHOLD=75

if [ "$CURRENT" -gt "$THRESHOLD" ] ; then
mail -s 'Disk Space Alert' mail@support.com << EOF
The data directory volume is almost full. Used: $CURRENT
%EOF
fi
```

Schedule this script in cron to run around the clock.

CCI: CCI-001855

Group ID (Vulid): V-73025
Group Title: SRG-APP-000353-DB-000324
Rule ID: SV-87677r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-010000
Rule Title: PostgreSQL must provide the means for individuals in authorized roles to change the auditing to be performed on all application components, based on all selectable event criteria within organization-defined time thresholds.

Vulnerability Discussion: If authorized individuals do not have the ability to modify auditing parameters in response to a changing threat environment, the organization may not be able to effectively respond, and important forensic information may be lost.

This requirement enables organizations to extend or limit auditing as necessary to meet organizational requirements. Auditing that is limited to conserve information system resources may be extended to address certain threat situations. In addition, auditing may be limited to a specific set of events to facilitate audit reduction, analysis, and reporting. Organizations can establish time thresholds in which audit actions are changed, for example, near real time, within minutes, or within hours.

Check Content:
First, as the database administrator, check if pgaudit is present in shared_preload_libraries:

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

If pgaudit is not present in the result from the query, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

For audit logging we suggest using pgaudit. For instructions on how to setup pgaudit, see supplementary content APPENDIX-B.

As a superuser (postgres), any pgaudit parameter can be changed in postgresql.conf. Configurations can only be changed by a superuser.

Example: Change Auditing To Log Any ROLE Statements

Note: This will override any setting already configured.

Alter the configuration to do role-based logging:

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

Add the following parameters (or edit existing parameters):

```
pgaudit.log = 'role'
```

Now, as the system administrator, reload the server with the new configuration:

```
# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

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

Example: Set An Auditing Role And Grant Privileges

An audit role can be configured and granted privileges to specific tables and columns that need logging.

Create Test Table

$ sudo su - postgres
$ psql -c "CREATE TABLE public.stig_audit_example(id INT, name TEXT, password TEXT);"

Define Auditing Role

As PostgreSQL superuser (such as postgres), add the following to postgresql.conf or any included configuration files.

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

Add the following parameters (or edit existing parameters):

pgaudit.role = 'auditor'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

Next in PostgreSQL create a new role:

postgres=# CREATE ROLE auditor;
postgres=# GRANT select(password) ON public.stig_audit_example TO auditor;

Note: This role is created with NOLOGIN privileges by default.

Now any SELECT on the column password will be logged:

$ sudo su - postgres
$ psql -c "SELECT password FROM public.stig_audit_example;"
$ cat ${PGDATA?}/pg_log/<latest_log>
< 2016-01-28 16:46:09.038 UTC bob postgres: >LOG: AUDIT: OBJECT,6,1,READ,SELECT,TABLE,public.stig_audit_example,SELECT password FROM stig_audit_example;,<none>

Change Configurations During A Specific Timeframe

Deploy PostgreSQL that allows audit configuration changes to take effect within the timeframe required by the application owner and without involving actions or events that the application owner rules unacceptable.

Crontab can be used to do this.

For a specific audit role:

Grant specific audit privileges to an auditing role at 5 PM every day of the week, month, year at the 0 minute mark.
0 5 * * * postgres /usr/bin/psql -c "GRANT select(password) ON public.stig_audit_example TO auditor;"
Revoke specific audit privileges to an auditing role at 5 PM every day of the week, month, year at the 0 minute mark.
0 17 * * * postgres /usr/bin/psql -c "REVOKE select(password) ON public.stig_audit_example FROM auditor;"

CCI: CCI-001914

Group ID (Vulid): V-73027
Group Title: SRG-APP-000389-DB-000372
Rule ID: SV-87679r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-010100
Rule Title: PostgreSQL must require users to reauthenticate when organization-defined circumstances or situations require reauthentication.

Vulnerability Discussion: The DoD standard for authentication of an interactive user is the presentation of a Common Access Card (CAC) or other physical token bearing a valid, current, DoD-issued Public Key Infrastructure (PKI) certificate, coupled with a Personal Identification Number (PIN) to be entered by the user at the beginning of each session and whenever reauthentication is required.

Without reauthentication, users may access resources or perform tasks for which they do not have authorization.

When applications provide the capability to change security roles or escalate the functional capability of the application, it is critical the user re-authenticate.

In addition to the reauthentication requirements associated with session locks, organizations may require reauthentication of individuals and/or devices in other situations, including (but not limited to) the following circumstances:

(i) When authenticators change;
(ii) When roles change;
(iii) When security categorized information systems change;
(iv) When the execution of privileged functions occurs;
(v) After a fixed period of time; or
(vi) Periodically.

Within the DoD, the minimum circumstances requiring reauthentication are privilege escalation and role changes.

Check Content:
Determine all situations where a user must re-authenticate. Check if the mechanisms that handle such situations use the following SQL:

To make a single user re-authenticate, the following must be present:

SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE user='<username>'

To make all users re-authenticate, run the following:

SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE user LIKE '%'

If the provided SQL does not force re-authentication, this is a finding.

Fix Text: Modify and/or configure PostgreSQL and related applications and tools so that users are always required to reauthenticate when changing role or escalating privileges.

To make a single user re-authenticate, the following must be present:

SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE user='<username>'

To make all users re-authenticate, the following must be present:

SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE user LIKE '%'

CCI: CCI-002038

Group ID (Vulid): V-73029
Group Title: SRG-APP-000176-DB-000068
Rule ID: SV-87681r1_rule
Severity: CAT I
Rule Version (STIG-ID): PGS9-00-010200
Rule Title: PostgreSQL must enforce authorized access to all PKI private keys stored/utilized by PostgreSQL.

Vulnerability Discussion: The DoD standard for authentication is DoD-approved PKI certificates. PKI certificate-based authentication is performed by requiring the certificate holder to cryptographically prove possession of the corresponding private key.

If the private key is stolen, an attacker can use the private key(s) to impersonate the certificate holder. In cases where PostgreSQL-stored private keys are used to authenticate PostgreSQL to the system's clients, loss of the corresponding private keys would allow an attacker to successfully perform undetected man-in-the-middle attacks against PostgreSQL system and its clients.

Both the holder of a digital certificate and the issuing authority must take careful measures to protect the corresponding private key. Private keys should always be generated and protected in FIPS 140-2 validated cryptographic modules.

All access to the private key(s) of PostgreSQL must be restricted to authorized and authenticated users. If unauthorized users have access to one or more of PostgreSQL's private keys, an attacker could gain access to the key(s) and use them to impersonate the database on the network or otherwise perform unauthorized actions.

Check Content:
First, as the database administrator (shown here as "postgres"), verify the following settings:

Note: If no specific directory given before the filename, the files are stored in PGDATA.

$ sudo su - postgres
$ psql -c "SHOW ssl_ca_file"
$ psql -c "SHOW ssl_cert_file"
$ psql -c "SHOW ssl_crl_file"
$ psql -c "SHOW ssl_key_file"

If the directory these files are stored in is not protected, this is a finding.

Fix Text: Store all PostgreSQL PKI private keys in a FIPS 140-2 validated cryptographic module. Ensure access to PostgreSQL PKI private keys is restricted to only authenticated and authorized users.

PostgreSQL private key(s) can be stored in $PGDATA directory, which is only accessible by the database owner (usually postgres, DBA) user. Do not allow access to this system account to unauthorized users.

To put the keys in a different directory, as the database administrator (shown here as "postgres"), set the following settings to a protected directory:

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
ssl_ca_file = "/some/protected/directory/root.crt"
ssl_crl_file = "/some/protected/directory/root.crl"
ssl_cert_file = "/some/protected/directory/server.crt"
ssl_key_file = "/some/protected/directory/server.key"

Now, as the system administrator, restart the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl restartpostgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 restart

For more information on configuring PostgreSQL to use SSL, see supplementary content APPENDIX-G.

CCI: CCI-000186

Group ID (Vulid): V-73031
Group Title: SRG-APP-000427-DB-000385
Rule ID: SV-87683r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-010300
Rule Title: PostgreSQL must only accept end entity certificates issued by DoD PKI or DoD-approved PKI Certification Authorities (CAs) for the establishment of all encrypted sessions.

Vulnerability Discussion: Only DoD-approved external PKIs have been evaluated to ensure that they have security controls and identity vetting procedures in place which are sufficient for DoD systems to rely on the identity asserted in the certificate. PKIs lacking sufficient security controls and identity vetting procedures risk being compromised and issuing certificates that enable adversaries to impersonate legitimate users.

The authoritative list of DoD-approved PKIs is published at http://iase.disa.mil/pki-pke/interoperability.

This requirement focuses on communications protection for PostgreSQL session rather than for the network packet.

Check Content:
As the database administrator (shown here as "postgres"), verify the following setting in postgresql.conf:

$ sudo su - postgres
$ psql -c "SHOW ssl_ca_file"
$ psql -c "SHOW ssl_cert_file"

If the database is not configured to used approved certificates, this is a finding.

Fix Text: Revoke trust in any certificates not issued by a DoD-approved certificate authority.

Configure PostgreSQL to accept only DoD and DoD-approved PKI end-entity certificates.

To configure PostgreSQL to accept approved CA's, see the official PostgreSQL documentation: http://www.postgresql.org/docs/current/static/ssl-tcp.html

For more information on configuring PostgreSQL to use SSL, see supplementary content APPENDIX-G.

CCI: CCI-002470

Group ID (Vulid): V-73033
Group Title: SRG-APP-000095-DB-000039
Rule ID: SV-87685r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-010400
Rule Title: PostgreSQL must produce audit records containing sufficient information to establish what type of events occurred.

Vulnerability Discussion: Information system auditing capability is critical for accurate forensic analysis. Without establishing what type of event occurred, it would be difficult to establish, correlate, and investigate the events relating to an incident or identify those responsible for one.

Audit record content that may be necessary to satisfy the requirement of this policy includes, for example, time stamps, user/process identifiers, event descriptions, success/fail indications, filenames involved, and access control or flow control rules invoked.

Associating event types with detected events in the application and audit logs provides a means of investigating an attack; recognizing resource utilization or capacity thresholds; or identifying an improperly configured application.

Database software is capable of a range of actions on data stored within the database. It is important, for accurate forensic analysis, to know exactly what actions were performed. This requires specific information regarding the event type an audit record is referring to. If event type information is not recorded and stored with the audit record, the record itself is of very limited use.

Check Content:
As the database administrator (shown here as "postgres"), verify the current log_line_prefix setting in postgresql.conf:

$ sudo su - postgres
$ psql -c "SHOW log_line_prefix"

Verify that the current settings are appropriate for the organization.

The following is what is possible for logged information:

%a = application name
%u = user name
%d = database name
%r = remote host and port
%h = remote host
%p = process ID
%t = timestamp without milliseconds
%m = timestamp with milliseconds
%i = command tag
%e = SQL state
%c = session ID
%l = session line number
%s = session start timestamp
%v = virtual transaction ID
%x = transaction ID (0 if none)
%q = stop here in non-session
processes

If the audit record does not log events required by the organization, this is a finding.

Next, verify the current settings of log_connections and log_disconnections by running the following SQL:

$ psql -c "SHOW log_connections"
$ psql -c "SHOW log_disconnections"

If both settings are off, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

If logging is enabled the following configurations must be made to log connections, date/time, username and session identifier.

First, edit the postgresql.conf file as a privileged user:

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

Edit the following parameters based on the organization's needs (minimum requirements are as follows):

log_connections = on
log_disconnections = on
log_line_prefix = '< %m %u %d %c: >'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000130

Group ID (Vulid): V-73035
Group Title: SRG-APP-000429-DB-000387
Rule ID: SV-87687r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-010500
Rule Title: PostgreSQL must implement cryptographic mechanisms preventing the unauthorized disclosure of organization-defined information at rest on organization-defined information system components.

Vulnerability Discussion: PostgreSQLs handling data requiring "data at rest" protections must employ cryptographic mechanisms to prevent unauthorized disclosure and modification of the information at rest. These cryptographic mechanisms may be native to PostgreSQL or implemented via additional software or operating system/file system settings, as appropriate to the situation.

Selection of a cryptographic mechanism is based on the need to protect the integrity of organizational information. The strength of the mechanism is commensurate with the security category and/or classification of the information. Organizations have the flexibility to either encrypt all information on storage devices (i.e., full disk encryption) or encrypt specific data structures (e.g., files, records, or fields).

The decision whether and what to encrypt rests with the data owner and is also influenced by the physical measures taken to secure the equipment and media on which the information resides.

Check Content:
To check if pgcrypto is installed on PostgreSQL, as a database administrator (shown here as "postgres"), run the following command:

$ sudo su - postgres
$ psql -c "SELECT * FROM pg_available_extensions where name='pgcrypto'"

If data in the database requires encryption and pgcrypto is not available, this is a finding.

If a disk or filesystem requires encryption, ask the system owner, DBA, and SA to demonstrate the use of filesystem and/or disk-level encryption. If this is required and is not found, this is a finding.

Fix Text: Configure PostgreSQL, operating system/file system, and additional software as relevant, to provide the required level of cryptographic protection for information requiring cryptographic protection against disclosure.

Secure the premises, equipment, and media to provide the required level of physical protection.

The pgcrypto module provides cryptographic functions for PostgreSQL. See supplementary content APPENDIX-E for documentation on installing pgcrypto.

With pgcrypto installed, it is possible to insert encrypted data into the database:

INSERT INTO accounts(username, password) VALUES ('bob', crypt('a_secure_password', gen_salt('md5')));

CCI: CCI-002476

Group ID (Vulid): V-73037
Group Title: SRG-APP-000220-DB-000149
Rule ID: SV-87689r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-010600
Rule Title: PostgreSQL must invalidate session identifiers upon user logout or other session termination.

Vulnerability Discussion: Captured sessions can be reused in "replay" attacks. This requirement limits the ability of adversaries to capture and continue to employ previously valid session IDs.

This requirement focuses on communications protection for PostgreSQL session rather than for the network packet. The intent of this control is to establish grounds for confidence at each end of a communications session in the ongoing identity of the other party and in the validity of the information being transmitted.

Session IDs are tokens generated by PostgreSQLs to uniquely identify a user's (or process's) session. DBMSs will make access decisions and execute logic based on the session ID.

Unique session IDs help to reduce predictability of said identifiers. Unique session IDs address man-in-the-middle attacks, including session hijacking or insertion of false information into a session. If the attacker is unable to identify or guess the session information related to pending application traffic, they will have more difficulty in hijacking the session or otherwise manipulating valid sessions.

When a user logs out, or when any other session termination event occurs, PostgreSQL must terminate the user session(s) to minimize the potential for sessions to be hijacked.

Check Content:
As the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "SHOW tcp_keepalives_idle"

$ psql -c "SHOW tcp_keepalives_interval"
$ psql -c "SHOW tcp_keepalives_count"
$ psql -c "SHOW statement_timeout"

If these settings are not set, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

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

$ sudo su - postgres
$ vi $PGDATA/postgresql.conf

Set the following parameters to organizational requirements:

statement_timeout = 10000 #milliseconds
tcp_keepalives_idle = 10 # seconds
tcp_keepalives_interval = 10 # seconds
tcp_keepalives_count = 10

Now, as the system administrator, restart the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl restart postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 restart

CCI: CCI-001185

Group ID (Vulid): V-73039
Group Title: SRG-APP-000121-DB-000202
Rule ID: SV-87691r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-010700
Rule Title: PostgreSQL must protect its audit features from unauthorized access.

Vulnerability Discussion: Protecting audit data also includes identifying and protecting the tools used to view and manipulate log data.

Depending upon the log format and application, system and application log tools may provide the only means to manipulate and manage application and system log data. It is, therefore, imperative that access to audit tools be controlled and protected from unauthorized access.

Applications providing tools to interface with audit data will leverage user permissions and roles identifying the user accessing the tools and the corresponding rights the user enjoys in order make access decisions regarding the access to audit tools.

Audit tools include, but are not limited to, OS-provided audit tools, vendor-provided audit tools, and open source audit tools needed to successfully view and manipulate audit information system activity and records.

If an attacker were to gain access to audit tools, he could analyze audit logs for system weaknesses or weaknesses in the auditing itself. An attacker could also manipulate logs to hide evidence of malicious activity.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA. Only the database owner and superuser can alter configuration of PostgreSQL.

Make sure the pg_log directory are owned by postgres user and group:

$ sudo su - postgres
$ ls -la ${PGDATA?}/pg_log

If pg_log is not owned by the database owner, this is a finding.

Make sure the data directory are owned by postgres user and group.

$ sudo su - postgres
$ ls -la ${PGDATA?}

If PGDATA is not owned by the database owner, this is a finding.

Make sure pgaudit installation is owned by root:

$ sudo su - postgres
$ ls -la /usr/pgsql-9.5/share/contrib/pgaudit

If pgaudit installation is not owned by root, this is a finding.

Next, as the database administrator (shown here as "postgres"), run the following SQL to list all roles and their privileges:

$ sudo su - postgres
$ psql -x -c "\du"

If any role has "superuser" that should not, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

If pg_log or data directory are not owned by postgres user and group, configure them as follows:

$ sudo chown -R postgres:postgres ${PGDATA?}

If the pgaudit installation is not owned by root user and group, configure it as follows:

$ sudo chown -R root:root /usr/pgsql-9.5/share/contrib/pgaudit.

To remove superuser from a role, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "ALTER ROLE <role-name> WITH NOSUPERUSER"

CCI: CCI-001493

Group ID (Vulid): V-73041
Group Title: SRG-APP-000096-DB-000040
Rule ID: SV-87693r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-011100
Rule Title: PostgreSQL must produce audit records containing time stamps to establish when the events occurred.

Vulnerability Discussion: Information system auditing capability is critical for accurate forensic analysis. Without establishing when events occurred, it is impossible to establish, correlate, and investigate the events relating to an incident.

In order to compile an accurate risk assessment and provide forensic analysis, it is essential for security personnel to know the date and time when events occurred.

Associating the date and time with detected events in the application and audit logs provides a means of investigating an attack; recognizing resource utilization or capacity thresholds; or identifying an improperly configured application.

Database software is capable of a range of actions on data stored within the database. It is important, for accurate forensic analysis, to know exactly when specific actions were performed. This requires the date and time an audit record is referring to. If date and time information is not recorded and stored with the audit record, the record itself is of very limited use.

Check Content:
As the database administrator (usually postgres, run the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_line_prefix"

If the query result does not contain "%m", this is a finding.

Fix Text: Logging must be enabled in order to capture timestamps. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

If logging is enabled the following configurations must be made to log events with timestamps:

First, as the database administrator (shown here as "postgres"), edit postgresql.conf:

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

Add %m to log_line_prefix to enable timestamps with milliseconds:

log_line_prefix = '< %m >'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000131

Group ID (Vulid): V-73043
Group Title: SRG-APP-000123-DB-000204
Rule ID: SV-87695r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-011200
Rule Title: PostgreSQL must protect its audit features from unauthorized removal.

Vulnerability Discussion: Protecting audit data also includes identifying and protecting the tools used to view and manipulate log data. Therefore, protecting audit tools is necessary to prevent unauthorized operation on audit data.

Applications providing tools to interface with audit data will leverage user permissions and roles identifying the user accessing the tools and the corresponding rights the user enjoys in order make access decisions regarding the deletion of audit tools.

Audit tools include, but are not limited to, vendor-provided and open source audit tools needed to successfully view and manipulate audit information system activity and records. Audit tools include custom queries and report generators.

Check Content:
As the database administrator (shown here as "postgres"), verify the permissions of PGDATA:

$ sudo su - postgres
$ ls -la ${PGDATA?}

If PGDATA is not owned by postgres:postgres or if files can be accessed by others, this is a finding.

As the system administrator, verify the permissions of pgsql shared objects and compiled binaries:

$ ls -la /usr/pgsql-9.5/bin/
$ ls -la /usr/pgsql-9.5/share
$ ls -la /usr/pgsql-9.5/include

If any of these are not owned by root:root, this is a finding.

Fix Text: As the system administrator, change the permissions of PGDATA:

$ sudo chown -R postgres:postgres ${PGDATA?}
$ sudo chmod 700 ${PGDATA?}

As the system administrator, change the permissions of pgsql:

$ sudo chown -R root:root /usr/pgsql-9.5/share/contrib/pgaudit

CCI: CCI-001495

Group ID (Vulid): V-73045
Group Title: SRG-APP-000515-DB-000318
Rule ID: SV-87697r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-011300
Rule Title: PostgreSQL must off-load audit data to a separate log management facility; this must be continuous and in near real time for systems with a network connection to the storage facility and weekly or more often for stand-alone systems.

Vulnerability Discussion: Information stored in one location is vulnerable to accidental or incidental deletion or alteration.

Off-loading is a common process in information systems with limited audit storage capacity.

PostgreSQL may write audit records to database tables, to files in the file system, to other kinds of local repository, or directly to a centralized log management system. Whatever the method used, it must be compatible with off-loading the records to the centralized system.

Check Content:
First, as the database administrator (shown here as "postgres"), ensure PostgreSQL uses syslog by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_destination"

If log_destination is not syslog, this is a finding.

Next, as the database administrator, check which log facility is configured by running the following SQL:

$ psql -c "SHOW syslog_facility"

Check with the organization to see how syslog facilities are defined in their organization.

If the wrong facility is configured, this is a finding.

If PostgreSQL does not have a continuous network connection to the centralized log management system, and PostgreSQL audit records are not transferred to the centralized log management system weekly or more often, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Configure PostgreSQL or deploy and configure software tools to transfer audit records to a centralized log management system, continuously and in near-real time where a continuous network connection to the log management system exists, or at least weekly in the absence of such a connection.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

With logging enabled, as the database administrator (shown here as "postgres"), configure the follow parameters in postgresql.conf (the example uses the default values - tailor for environment):

Note: Consult the organization on how syslog facilities are defined in the syslog daemon configuration.

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
log_destination = 'syslog'
syslog_facility = 'LOCAL0'
syslog_ident = 'postgres'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-001851

Group ID (Vulid): V-73047
Group Title: SRG-APP-000224-DB-000384
Rule ID: SV-87699r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-011400
Rule Title: PostgreSQL must maintain the authenticity of communications sessions by guarding against man-in-the-middle attacks that guess at Session ID values.

Vulnerability Discussion: One class of man-in-the-middle, or session hijacking, attack involves the adversary guessing at valid session identifiers based on patterns in identifiers already known.

The preferred technique for thwarting guesses at Session IDs is the generation of unique session identifiers using a FIPS 140-2 approved random number generator.

However, it is recognized that available PostgreSQL products do not all implement the preferred technique yet may have other protections against session hijacking. Therefore, other techniques are acceptable, provided they are demonstrated to be effective.

Check Content:
To check if PostgreSQL is configured to use ssl, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "SHOW ssl"

If this is not set to on, this is a finding.

Fix Text: To configure PostgreSQL to use SSL, as a database owner (shown here as "postgres"), edit postgresql.conf:

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

Add the following parameter:

ssl = on

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

For more information on configuring PostgreSQL to use SSL, see supplementary content APPENDIX-G.

For further SSL configurations, see the official documentation: https://www.postgresql.org/docs/current/static/ssl-tcp.html

CCI: CCI-001188

Group ID (Vulid): V-73049
Group Title: SRG-APP-000148-DB-000103
Rule ID: SV-87701r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-011500
Rule Title: PostgreSQL must uniquely identify and authenticate organizational users (or processes acting on behalf of organizational users).

Vulnerability Discussion: To assure accountability and prevent unauthenticated access, organizational users must be identified and authenticated to prevent potential misuse and compromise of the system.

Organizational users include organizational employees or individuals the organization deems to have equivalent status of employees (e.g., contractors). Organizational users (and any processes acting on behalf of users) must be uniquely identified and authenticated for all accesses, except the following:

(i) Accesses explicitly identified and documented by the organization. Organizations document specific user actions that can be performed on the information system without identification or authentication; and
(ii) Accesses that occur through authorized use of group authenticators without individual authentication. Organizations may require unique identification of individuals using shared accounts, for detailed accountability of individual activity.

Check Content:
Review PostgreSQL settings to determine whether organizational users are uniquely identified and authenticated when logging on/connecting to the system.

To list all roles in the database, as the database administrator (shown here as "postgres"), run the following SQL:

$ sudo su - postgres
$ psql -c "\du"

If organizational users are not uniquely identified and authenticated, this is a finding.

Next, as the database administrator (shown here as "postgres"), verify the current pg_hba.conf authentication settings:

$ sudo su - postgres
$ cat ${PGDATA?}/pg_hba.conf

If every role does not have unique authentication requirements, this is a finding.

If accounts are determined to be shared, determine if individuals are first individually authenticated. If individuals are not individually authenticated before using the shared account, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Configure PostgreSQL settings to uniquely identify and authenticate all organizational users who log on/connect to the system.

To create roles, use the following SQL:

CREATE ROLE <role_name> [OPTIONS]

For more information on CREATE ROLE, see the official documentation: https://www.postgresql.org/docs/current/static/sql-createrole.html

For each role created, the database administrator can specify database authentication by editing pg_hba.conf:

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

An example pg_hba entry looks like this:

TYPE DATABASE USER ADDRESS METHOD
host test_db bob 192.168.0.0/16 md5

For more information on pg_hba.conf, see the official documentation: https://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html

CCI: CCI-000764

Group ID (Vulid): V-73051
Group Title: SRG-APP-000295-DB-000305
Rule ID: SV-87703r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-011600
Rule Title: PostgreSQL must automatically terminate a user session after organization-defined conditions or trigger events requiring session disconnect.

Vulnerability Discussion: This addresses the termination of user-initiated logical sessions in contrast to the termination of network connections that are associated with communications sessions (i.e., network disconnect). A logical session (for local, network, and remote access) is initiated whenever a user (or process acting on behalf of a user) accesses an organizational information system. Such user sessions can be terminated (and thus terminate user access) without terminating network sessions.

Session termination ends all processes associated with a user's logical session except those batch processes/jobs that are specifically created by the user (i.e., session owner) to continue after the session is terminated.

Conditions or trigger events requiring automatic session termination can include, for example, organization-defined periods of user inactivity, targeted responses to certain types of incidents, and time-of-day restrictions on information system use.

This capability is typically reserved for specific cases where the system owner, data owner, or organization requires additional assurance.

Check Content:
Review system documentation to obtain the organization's definition of circumstances requiring automatic session termination. If the documentation explicitly states that such termination is not required or is prohibited, this is not a finding.

If the documentation requires automatic session termination, but PostgreSQL is not configured accordingly, this is a finding.

Fix Text: Configure PostgreSQL to automatically terminate a user session after organization-defined conditions or trigger events requiring session termination.

Examples follow.

Change a role to nologin and disconnect the user

ALTER ROLE '<username>' NOLOGIN;
SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE username='<username>';

Disconnecting users during a specific time range
See supplementary content APPENDIX-A for a bash script for this example.

The script found in APPENDIX-A using the -l command can disable all users with rolcanlogin=t from logging in. The script keeps track of who it disables in a .restore_login file. After the specified time is over, the same script can be run with the -r command to restore all login connections.

This script would be added to a cron job:

lock at 5 am every day of the week, month, year at the 0 minute mark.
0 5 * * * postgres /var/lib/pgsql/no_login.sh -d postgres -l
restore at 5 pm every day of the week, month, year at the 0 minute mark.
0 17 * * * postgres /var/lib/pgsql/no_login.sh -d postgres -r

CCI: CCI-002361

Group ID (Vulid): V-73053
Group Title: SRG-APP-000340-DB-000304
Rule ID: SV-87705r1_rule
Severity: CAT I
Rule Version (STIG-ID): PGS9-00-011700
Rule Title: PostgreSQL must prevent non-privileged users from executing privileged functions, to include disabling, circumventing, or altering implemented security safeguards/countermeasures.

Vulnerability Discussion: Preventing non-privileged users from executing privileged functions mitigates the risk that unauthorized individuals or processes may gain unnecessary access to information or privileges.

System documentation should include a definition of the functionality considered privileged.

Depending on circumstances, privileged functions can include, for example, establishing accounts, performing system integrity checks, or administering cryptographic key management activities. Non-privileged users are individuals that do not possess appropriate authorizations. Circumventing intrusion detection and prevention mechanisms or malicious code protection mechanisms are examples of privileged functions that require protection from

non-privileged users.

A privileged function in PostgreSQL/database context is any operation that modifies the structure of the database, its built-in logic, or its security settings. This would include all Data Definition Language (DDL) statements and all security-related statements. In an SQL environment, it encompasses, but is not necessarily limited to:

CREATE
ALTER
DROP
GRANT
REVOKE

There may also be Data Manipulation Language (DML) statements that, subject to context, should be regarded as privileged. Possible examples include:

TRUNCATE TABLE;
DELETE, or
DELETE affecting more than n rows, for some n, or
DELETE without a WHERE clause;

UPDATE or
UPDATE affecting more than n rows, for some n, or
UPDATE without a WHERE clause;

any SELECT, INSERT, UPDATE, or DELETE to an application-defined security table executed by other than a security principal.

Depending on the capabilities of PostgreSQL and the design of the database and associated applications, the prevention of unauthorized use of privileged functions may be achieved by means of DBMS security features, database triggers, other mechanisms, or a combination of these.

However, the use of procedural languages within PostgreSQL, such as pl/R and pl/Python, introduce security risk. Any user on the PostgreSQL who is granted access to pl/R or pl/Python is able to run UDFs to escalate privileges and perform unintended functions. Procedural languages such as pl/Perl and pl/Java have "untrusted" mode of operation, which do not allow a non-privileged PostgreSQL user to escalate privileges or perform actions as a database administrator.

Check Content:
Review the system documentation to obtain the definition of the PostgreSQL functionality considered privileged in the context of the system in question.

Review the PostgreSQL security configuration and/or other means used to protect privileged functionality from unauthorized use.

If the configuration does not protect all of the actions defined as privileged, this is a finding.

If PostgreSQL instance uses procedural languages, such as pl/Python or pl/R, without AO authorization, this is a finding.

Fix Text: Configure PostgreSQL security to protect all privileged functionality.

If pl/R and pl/Python are used, document their intended use, document users that have access to pl/R and pl/Python, as well as their business use case, such as data-analytics or data-mining. Because of the risks associated with using pl/R and pl/Python, their use must have AO risk acceptance.

To remove unwanted extensions, use:

DROP EXTENSION <extension_name>

To remove unwanted privileges from a role, use the REVOKE command.

See the PostgreSQL documentation for more details: http://www.postgresql.org/docs/current/static/sql-revoke.html

CCI: CCI-002235

Group ID (Vulid): V-73055
Group Title: SRG-APP-000177-DB-000069
Rule ID: SV-87707r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-011800
Rule Title: PostgreSQL must map the PKI-authenticated identity to an associated user account.

Vulnerability Discussion: The DoD standard for authentication is DoD-approved PKI certificates. Once a PKI certificate has been validated, it must be mapped to PostgreSQL user account for the authenticated identity to be meaningful to PostgreSQL and useful for authorization decisions.

Check Content:
The cn (Common Name) attribute of the certificate will be compared to the requested database user name, and if they match the login will be allowed.

To check the cn of the certificate, using openssl, do the following:

$ openssl x509 -noout -subject -in client_cert

If the cn does not match the users listed in PostgreSQL and no user mapping is used, this is a finding.

User name mapping can be used to allow cn to be different from the database user name. If User Name Maps are used, run the following as the database administrator (shown here as "postgres"), to get a list of maps used for authentication:

$ sudo su - postgres
$ grep "map" ${PGDATA?}/pg_hba.conf

With the names of the maps used, check those maps against the user name mappings in pg_ident.conf:

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

If user accounts are not being mapped to authenticated identities, this is a finding.

If the cn and the username mapping do not match, this is a finding.

Fix Text: Configure PostgreSQL to map authenticated identities directly to PostgreSQL user accounts.

For information on configuring PostgreSQL to use SSL, see supplementary content APPENDIX-G.

CCI: CCI-000187

Group ID (Vulid): V-73057
Group Title: SRG-APP-000243-DB-000128
Rule ID: SV-87709r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-011900
Rule Title: Database contents must be protected from unauthorized and unintended information transfer by enforcement of a data-transfer policy.

Vulnerability Discussion: Applications, including PostgreSQLs, must prevent unauthorized and unintended information transfer via shared system resources.

Data used for the development and testing of applications often involves copying data from production. It is important that specific procedures exist for this process, to include the conditions under which such transfer may take place, where the copies may reside, and the rules for ensuring sensitive data are not exposed.

Copies of sensitive data must not be misplaced or left in a temporary location without the proper controls.

Check Content:
Review the procedures for the refreshing of development/test data from production.

Review any scripts or code that exists for the movement of production data to development/test systems, or to any other location or for any other purpose.

Verify that copies of production data are not left in unprotected locations.

If the code that exists for data movement does not comply with the organization-defined data transfer policy and/or fails to remove any copies of production data from unprotected locations, this is a finding.

Fix Text: Modify any code used for moving data from production to development/test systems to comply with the organization-defined data transfer policy, and to ensure copies of production data are not left in unsecured locations.

CCI: CCI-001090

Group ID (Vulid): V-73059
Group Title: SRG-APP-000243-DB-000374
Rule ID: SV-87711r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-012000
Rule Title: Access to database files must be limited to relevant processes and to authorized, administrative users.

Vulnerability Discussion: Applications, including PostgreSQLs, must prevent unauthorized and unintended information transfer via shared system resources. Permitting only DBMS processes and authorized, administrative users to have access to the files where the database resides helps ensure that those files are not shared inappropriately and are not open to backdoor access and manipulation.

Check Content:
Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Review the permissions granted to users by the operating system/file system on the database files, database log files and database backup files.

To verify that all files are owned by the database administrator and have the correct permissions, run the following as the database administrator (shown here as "postgres"):

$ sudo su - postgres
$ ls -lR ${PGDATA?}

If any files are not owned by the database administrator or allow anyone but the database administrator to read/write/execute, this is a finding.

If any user/role who is not an authorized system administrator with a need-to-know or database administrator with a need-to-know, or a system account for running PostgreSQL processes, is permitted to read/view any of these files, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

Configure the permissions granted by the operating system/file system on the database files, database log files, and database backup files so that only relevant system accounts and authorized system administrators and database administrators with a need to know are permitted to read/view these files.

Any files (for example: extra configuration files) created in PGDATA must be owned by the database administrator, with only owner permissions to read, write, and execute.

CCI: CCI-001090

Group ID (Vulid): V-73061
Group Title: SRG-APP-000122-DB-000203
Rule ID: SV-87713r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-012200
Rule Title: PostgreSQL must protect its audit configuration from unauthorized modification.

Vulnerability Discussion: Protecting audit data also includes identifying and protecting the tools used to view and manipulate log data. Therefore, protecting audit tools is necessary to prevent unauthorized operation on audit data.

Applications providing tools to interface with audit data will leverage user permissions and roles identifying the user accessing the tools and the corresponding rights the user enjoys in order make access decisions regarding the modification of audit tools.

Audit tools include, but are not limited to, vendor-provided and open source audit tools needed to successfully view and manipulate audit information system activity and records. Audit tools include custom queries and report generators.

Check Content:
All configurations for auditing and logging can be found in the postgresql.conf configuration file. By default, this file is owned by the database administrator account.

To check that the permissions of the postgresql.conf are owned by the database administrator with permissions of 0600, run the following as the database administrator (shown here as "postgres"):

$ sudo su - postgres
$ ls -la ${PGDATA?}

If postgresql.conf is not owned by the database administrator or does not have 0600 permissions, this is a finding.

stderr Logging

To check that logs are created with 0600 permissions, check the postgresql.conf file for the following setting:

$ sudo su - postgres
$ psql -c "SHOW log_file_mode"

If permissions are not 0600, this is a finding.

syslog Logging

If PostgreSQL is configured to use syslog, verify that the logs are owned by root and have 0600 permissions. If they are not, this is a finding.

Fix Text: Apply or modify access controls and permissions (both within PostgreSQL and in the file system/operating system) to tools used to view or modify audit log data. Tools must be configurable by authorized personnel only.

$ sudo su - postgres
$ vi ${PGDATA?}/postgresql.conf
log_file_mode = 0600

Next, as the database administrator (shown here as "postgres"), change the ownership and permissions of configuration files in PGDATA:

$ sudo su - postgres

$ chown postgres:postgres ${PGDATA?}/*.conf
$ chmod 0600 ${PGDATA?}/*.conf

CCI: CCI-001494

Group ID (Vulid): V-73063
Group Title: SRG-APP-000179-DB-000114
Rule ID: SV-87715r1_rule
Severity: CAT I
Rule Version (STIG-ID): PGS9-00-012300
Rule Title: PostgreSQL must use NIST FIPS 140-2 validated cryptographic modules for cryptographic operations.

Vulnerability Discussion: Use of weak or not validated cryptographic algorithms undermines the purposes of utilizing encryption and digital signatures to protect data. Weak algorithms can be easily broken and not validated cryptographic modules may not implement algorithms correctly. Unapproved cryptographic modules or algorithms should not be relied on for authentication, confidentiality or integrity. Weak cryptography could allow an attacker to gain access to and modify data stored in the database as well as the administration settings of the DBMS.

Applications, including DBMSs, utilizing cryptography are required to use approved NIST FIPS 140-2 validated cryptographic modules that meet the requirements of applicable federal laws, Executive Orders, directives, policies, regulations, standards, and guidance.

The security functions validated as part of FIPS 140-2 for cryptographic modules are described in FIPS 140-2 Annex A.

NSA Type-X (where X=1, 2, 3, 4) products are NSA-certified, hardware-based encryption modules.

Check Content:
As the system administrator, run the following:

$ openssl version

If "fips" is not included in the openssl version, this is a finding.

Fix Text: Configure OpenSSL to meet FIPS Compliance using the following documentation in section 9.1:

http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1758.pdf

For more information on configuring PostgreSQL to use SSL, see supplementary content APPENDIX-G.

CCI: CCI-000803

Group ID (Vulid): V-73065
Group Title: SRG-APP-000502-DB-000348
Rule ID: SV-87717r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-012500
Rule Title: Audit records must be generated when categorized information (e.g., classification levels/security levels) is deleted.

Vulnerability Discussion: Changes in categorized information must be tracked. Without an audit trail, unauthorized access to protected data could go undetected.

For detailed information on categorizing information, refer to FIPS Publication 199, Standards for Security Categorization of Federal Information and Information Systems, and FIPS Publication 200, Minimum Security Requirements for Federal Information and Information Systems.

Check Content:
As the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Verify that role, read, write and ddl auditing are enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain role, read, write, and ddl, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations can be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log='ddl, role, read, write'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-73067
Group Title: SRG-APP-000507-DB-000356
Rule ID: SV-87719r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-012600
Rule Title: PostgreSQL must generate audit records when successful accesses to objects occur.

Vulnerability Discussion: Without tracking all or selected types of access to all or selected objects (tables, views, procedures, functions, etc.), it would be difficult to establish, correlate, and investigate the events relating to an incident, or identify those responsible for one.

In an SQL environment, types of access include, but are not necessarily limited to:

SELECT
INSERT
UPDATE
DELETE
EXECUTE

Check Content:
As the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Verify that role, read, write, and ddl auditing are enabled:

$ psql -c "SHOW pgaudit.log"

If the output does not contain read and write, this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA. To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

If logging is enabled the following configurations must be made to log unsuccessful connections, date/time, username and session identifier.

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

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

Edit the following parameters:

log_connections = on
log_line_prefix = '< %m %u %c: >'
pgaudit.log = 'read, write'

Where:
* %m is the time and date
* %u is the username
* %c is the session ID for the connection

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-73069
Group Title: SRG-APP-000508-DB-000358
Rule ID: SV-87721r1_rule
Severity: CAT II
Rule Version (STIG-ID): PGS9-00-012700
Rule Title: PostgreSQL must generate audit records for all direct access to the database(s).

Vulnerability Discussion: In this context, direct access is any query, command, or call to the DBMS that comes from any source other than the application(s) that it supports. Examples would be the command line or a database management utility program. The intent is to capture all activity from administrative and non-standard sources.

Check Content:
As the database administrator, verify pgaudit is enabled by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW shared_preload_libraries"

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

Verify that connections and disconnections are being logged by running the following SQL:

$ sudo su - postgres
$ psql -c "SHOW log_connections"
$ psql -c "SHOW log_disconnections"

If the output does not contain "on", this is a finding.

Fix Text: Note: The following instructions use the PGDATA environment variable. See supplementary content APPENDIX-F for instructions on configuring PGDATA.

To ensure that logging is enabled, review supplementary content APPENDIX-C for instructions on enabling logging.

Using pgaudit PostgreSQL can be configured to audit these requests. See supplementary content APPENDIX-B for documentation on installing pgaudit.

With pgaudit installed the following configurations should be made:

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

Add the following parameters (or edit existing parameters):

pgaudit.log='ddl, role, read, write'
log_connections='on'
log_disconnections='on'

Now, as the system administrator, reload the server with the new configuration:

# SYSTEMD SERVER ONLY
$ sudo systemctl reload postgresql-9.5

# INITD SERVER ONLY
$ sudo service postgresql-9.5 reload

CCI: CCI-000172

Group ID (Vulid): V-73071
Group Title: SRG-APP-000179-DB-000114
Rule ID: SV-87723r1_rule
Severity: CAT I
Rule Version (STIG-ID): PGS9-00-012800

Rule Title: The DBMS must be configured on a platform that has a NIST certified FIPS 140-2 installation of OpenSSL.

Vulnerability Discussion: Postgres uses OpenSSL for the underlying encryption layer. 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.

Check Content:
If the deployment incorporates a custom build of the operating system and Postgres guaranteeing the use of FIPS 140-2- compliant OpenSSL, this is not a finding.

If the Postgres Plus Advanced Server is not installed on Red Hat Enterprise Linux (RHEL), this is a finding.

If FIPS encryption is not enabled, this is a finding.

Fix Text: Install Postgres with FIPS-compliant cryptography enabled on RHEL; or by other means ensure that FIPS 140-2 certified OpenSSL libraries are used by the DBMS.

CCI: CCI-000803

UNCLASSIFIED