	HARDENING POSTGRESQL	V.1

CONTROL DE VERSIONES

Elaborado por:	Ricardo M Andrade M	**No. de Versión:**	1
Revisado por:		**Fecha de revisión:**	
Aprobado por:		**Fecha de aprobación:**	

Historia de Modificaciones

No. de Versión	Fecha de Versión	Autor	Revisado por	Aprobado por	Descripción
1	Febrero 15, 2023	Ricardo Mario Andrade Muñiz			

1. DESCRIPCIÓN GENERAL

Objetivo

Presentar los puntos primordiales para ejecutar un proceso de gestión de seguridad y auditoria de los servicios del motor de Base de Datos PostgreSQL, establecidos por SETI, basados en documentos generados por el Fabricante y en la experiencia de SETI S.A.S.

Definiciones

Observaciones

Este documento está está elaborado de tal forma que puede ser aplicado desde la versión 9 a la versión 14 del motor de base de datos PostgreSQL.

Tener en cuenta:

- Los Hardening por demanda realizados por parte del cliente pueden generar oportunidades de mejora puesto que los servicios están en constante cambio o evolución.
- Cualquier punto tratado en este documento podría representar un Riesgo el cual demandará un análisis más profundo del tema específico.
- La ejecución de esta actividad de Hardening puede llevar a la planeación y ejecución de actividades de PAYM.

Las tareas presentadas en este documento también aplican cuando se desea ajustar y mejorar los lineamientos vigentes conforme lo demanden las buenas prácticas del fabricante o la experiencia de SETI.

La Plantillas Default mencionada en este documento, se relacionan a continuación:
A03-PS030106 Estándar de Seguridad BD Postgress,
CIS_PostgreSQL_9_Benchmark_v1.0.0.pdf, CIS_PostgreSQL_10_Benchmark_v1.0.0.pdf,
CIS_PostgreSQL_11_Benchmark_v1.0.0.pdf, CIS_PostgreSQL_12_Benchmark_v1.0.0.pdf,
CIS_PostgreSQL_13_Benchmark_v1.0.0.pdf, CIS_PostgreSQL_14_Benchmark_v1.0.0.pdf y
CIS_PostgreSQL_15_Benchmark_v1.0.0.pdf.

El documento asociado llamado "Lista de Chequeo adicional.xlsx" se utiliza para plasmar información a la cual no se tenga acceso directo por parte de SETI o que estén fuera de nuestro alcance, pero que igual, permiten confrontar y llevar un registro del estado de aseguramiento de los servicios de Bases de datos en sus diferentes niveles.

CALIFICACION GENERAL DEL HARDENING

Teniendo en cuenta los puntos tratados en este documento , su nivel de criticidad y las evidencias obtenidas , resultado del proceso ejecutado, se presenta a continuación una tabla con los hallazgos que permitirá calificar el nivel de cumplimiento de este Hardening.

		NO CUMPLIMIENTO DE VULNERABILIDADES		
CAPA		**ALTA**	**MEDIA**	**BAJA**
BASES DE DATOS	**VULNERABILIDADES**	**##%**	**##%**	**##%**
	Sistema Operativo			
	Log de Instancia			
	Archivos de Configuración			
	TableSpace			
	Roles, Usuarios y Permisos			
	Consideraciones especiales			
Total ítems Evaluados:	**##**	**#**	**#**	**#**

1. VULNERABILIDADES A NIVEL DE SISTEMA OPERATIVO

TAREA	CRITICIDAD

1. Instalación y Parches:
Importante comentar que a nivel de PostgreSQL no se manejan parches sino cambios de versión, por ejemplo: estar en la versión 10.1.7 y pasa a la versión 10.1.8, lo que conlleva la instalación completa del motor. Además de verificar que se cuente con las últimas versiones liberadas por el Fabricante, es necesario determinar que los paquetes se hayan obtenido de repositorios autorizados y confiables, según manual del fabricante.
 	
NOTA: Si se utilizan repositorios no autorizados o confiables se corre el riesgo de utilizar productos manipulados por terceros, con altas fallas de seguridad o rastreo, incluso, sin ningún tipo de documentación de soporte valida.
 	
AUDITORIA:
A. Determine que los repositorios listados a continuación son válidos y autorizados, para el caso de PostgreSQL el repositorio oficial es: **yum.postgresql.org** o **apt.postgresql.org**:
`dnf provides '*libpq.so'`
`dnf repolist all`
`dnf repolist all | egrep 'enabled$'`
B. Asegure que las versiones de los paquetes principales coincidan todas con la versión actual del producto:
`dnf info $(rpm -qa|grep postgres) | egrep '^Name|^Version|^From'`
 	
EVIDENCIAS:
	Alta

2. Registro de servicio de PostgreSQL en SYSTEMD: En caso de que el servidor sufra algún cambio de estado, es necesario que los servicios de PostgreSQL estén configurados para actuar en consecuencia, para evitar que los tiempos de indisponibilidad se extiendan más de lo necesario, así mismo, garantizando que su cambio de estado se corrija de forma automática. **NOTA:** El no hacer uso de automatizaciones mediante SYSTEMD no es posible asegurar que los servicios de PostgreSQL vuelvan a estar activos rápidamente en caso de cambios de estado en el servidor.	**Media**

TAREA	CRITICIDAD

AUDITORIA:
A. Establecer el Target a consultar:
`systemctl get-default`
B. Listar las dependencias sobre el Target encontrado:
`systemctl list-dependencies multi-user.target | grep -i postgres`
 	
EVIDENCIAS:	

3. <u>Inicio correcto de DATA CLUSTER, Permisos en Carpetas y Binarios:</u>
Todo servicio de PostgreSQL que este correctamente implementado a nivel de propietario y permisos debe cumplir los siguientes criterios:
➢ Los archivos y carpetas ubicados en el directorio de DATA, deben ser de propiedad del usuario propietario de la instalación del motor de PostgreSQL. (Generalmente es el usuario: postgres)
➢ Sobre el directorio de DATA ningún usuario o grupo distinto al propietario debe poseer permisos de ninguna índole.
➢ El directorio DATA no puede ser de propiedad del usuario **root** del sistema operativo.
➢ El servicio o procesos de PostgreSQL no debe ser invocado o activado por el usuario **root** o por cualquier otro distinto al propietario de los Binarios de instalación.
 	
NOTA: El no contar con estas pautas, permitirá la inestabilidad tanto de seguridad como del servicio mismo de la instancia y de los Datos contenidos en la misma, incluso, llegar al punto de fallar en el momento de iniciarla.
 	
AUDITORIA:
A. Verificar que los mínimos procesos de PostgreSQL estén en pleno funcionamiento:
`ps -fea|grep postg`
`(mínimamente: postgres -D, logger process, writer process, autovacuum, stats collector process)`
B. Determinar el propietario, grupo y permisos de los archivos de la instancia:
`ls -la ~postgres/13 (13 corresponde al número de la versión existente)`
	Alta

TAREA	CRITICIDAD

Ejemplos de permisos correctos en archivos y carpetas:
`drwx------. postgres postgres 4096 Oct 4 14:01 data`
`-rw-------. postgres postgres 923 Oct 4 14:01 initdb.log`
C. Verificar los permisos y atributos de la Carpeta o Directorio de DATA:
`/usr/pgsql-13/bin/postgresql-13-check-db-dir ~postgres/13/data`
`echo $?`
■ Si el resultado es cero (`0`), se indica que todo está correctamente establecido.
 	
EVIDENCIAS:	

4. <u>UMASK – Permisos de Archivos:</u>
Todos los archivos y Carpetas son creados siguiendo y criterio base de permisos derivados del usuario que los está creando, sin embargo, lo ideal es restringirlos para que sean accedidos por quien exclusivamente debe hacerlo. Para ambientes de Linux el permiso o UMASK por defecto es 0022 y para motores PostgreSQL se recomienda que el usuario propietario de los Binarios e instancia cuente con un UMASK de 077.
 	
NOTA: Esta asignación de UMASK 077 permite restringir el acceso a carpetas y archivos a cualquier usuario exceptuando al propietario de la instancia y sus binarios, además, evita utilizar el comando chmod para ajustar permisos, el cual, es permisivo a errores.
 	
AUDITORIA:
C. Conectado con el usuario propietario, ejecutar lo siguiente para determinar el UMASK actual:
`umask`
D. Determinar a nivel de archivos de Perfil (Profile) el UMASK establecido:
`grep "umask" .bash_profile / grep "umask" .profile / grep "umask" .bashrc`
 	
EVIDENCIAS:
	Alto

5. <u>Uso del grupo PG WHEEL para permisos de tipo "SUPERUSER":</u> En los sistemas operativos Linux sobre el cual se cuente con el aprovisionamiento de una instancia PostgreSQL,	**Alto**

TAREA	CRITICIDAD

es posible hacer uso del grupo llamado PG_WHEEL. Asociando cualquier cuenta de usuario a este grupo, inmediatamente se le asignan el permiso de SuperUser sobre la instancia de PostgreSQL y puede tener acceso a la misma con este nivel de permisos al hacer uso del comando SUDO por sistema operativo.
 	
NOTA: No es esencial que este grupo exista, de existir, solo usuarios que realmente deban actuar como administradores del servicio en general de la instancia de PostgreSQL, requieren estar asociados al mismo, de lo contrario, se activa un alto riesgo de seguridad.
 	
AUDITORIA:
A. Verificar la existencia del grupo PG_WHEEL:
`getent group pg_wheel`
B. Listar los miembros de este grupo:
`awk -F':' '/pg_wheel/{print $4}' /etc/group`
 	
EVIDENCIAS:	

6. Conexión Directa con Usuario "postgres" a Nivel de SO: Este usuario es comúnmente conocido por ser el propietario del servicio e instalación de los motores PostgreSQL, por lo que es buscado para hacer intrusión directa a las bases de datos con permisos de Super Administrador. Se hace necesario establecer las siguientes pautas para controlar su uso, teniendo en cuenta que solo los consultores pertenecientes al grupo DBA deben contar con acceso al Motor de forma directa mediante el Sistema Operativo: ➢ Crear usuarios nombrados ya sea para SETI o para cada consultor perteneciente al grupo DBA. ➢ Otorgar permisos a los usuarios anteriores para hacer SUDO hacia el usuario "postgres". ➢ Activar, mediante el usuario root, el registro o seguimiento de todas las actividades ejecutadas tanto por los usuarios nombrados como por el usuario "postgres". ➢ Establecer procesos de auditoría de la información de rastreo o seguimiento (directamente por el área encargada en el cliente) de las labores ejecutadas por el usuario "postgres", así, como los intentos correctos y fallidos de SUDO a este usuario o de conexión al Sistema Operativo, para monitorear el uso de	**Alta**

TAREA	CRITICIDAD

usuarios y permisos, además, de visualizar eventos de acceso indebido.
 	
NOTA: El no asegurar el acceso al usuario "postgres" pone en grave riesgo la información y vida útil de los datos almacenados en las Bases de Datos de la Instancia de PostgreSQL.
AUDITORIA:
A. Visualizar la lista de usuarios con permisos de SUDO:
`getent group sudo`
B. Determinar si el usuario con el cual se ha conectado tiene permisos de SUDO:
`sudo -l`
C. Verificar la asignación de SUDO y permisos para un usuario especifico:
`sudo -l -U user_name`
 	
EVIDENCIAS:	

2. VULNERABILIDADES A NIVEL DEL ARCHIVO LOG DE LA INSTANCIA

TAREA	CRITICIDAD
1. Método de Registro del LOG de la Instancia: En muchas oportunidades en las implementaciones de instancias de PostgreSQL se deja de lado la correcta configuración del Log de errores y seguimiento de la instancia, perdiendo de vista cualquier suceso o evento que se presente sobre esta, además, de las evidencias de conexiones de usuarios, entre otras. Existen diferentes métodos de registro y destino del Log de eventos donde el más utilizado es STDERR. Este método indica que los eventos a nivel de instancia de PostgreSQL tendrán como destino un archivo exclusivo, el cual, también estará bajo el control de la instancia. Se recomienda que el Log de la instancia No comparta ubicación o discos con aquellos utilizados por las bases de datos dado que esto puede afectar el I/O requerido de forma constante por el procesamiento a nivel de BD.	**Media**

TAREA	CRITICIDAD
NOTA: Se hace necesario llevar un correcto control de los ajustes a nivel de Log, dado que entre más labores se establezcan o activen, más escritura demandará y más información se tendrá que estar verificando o auditando por parte del área encargada. **AUDITORIA:** A. Verificar como está configurado el Log y su destino: `show log_destination;` **EVIDENCIAS:**	
2. Proceso de LOGGING COLLECTOR Automático: Este es un proceso Background que se encarga de capturar los mensajes del Log (STDERR) y registrarlos al archivo de Log físico de la instancia. **NOTA:** Cuando en el parámetro de configuración "log_destination" se establecer la opción "stderr" o "csvlog" es imperativo que este parámetro "logging_collector" este habilitado para que pueda ser ejecutado y aparecer activo en los procesos Background de la instancia. **AUDITORIA:** A. Verificar como está configurado el Log y su destino: `show logging_collector;` **EVIDENCIAS:**	**Alta**
3. Directorio de Destino para el Registro de LOG: A nivel de instancia, el parámetro que hace referencia a esta ruta es "log_directory", indicando el PATH sobre el cual se escribirán los archivos de Log cuando el parámetro "log_destination" esta seteado en "stderr" o "csvlog", siempre, se recomienda que sea establecido como una ruta absoluta. Al no contar con una ruta absoluta, sino, referencias o rutas relativas, esto puede afectar el registro adecuado del Log en la actualidad o no ser valido en procesos de Upgrade.	**Alta**

TAREA	CRITICIDAD
NOTA: En caso de no asignar una ruta al parámetro "log_directory", la instancia la asume como "/" la cual, hace referencia a la ruta raíz del sistema. En este caso, cuando la instancia intente escribir en dicha ruta "/" típicamente se presentará un error de escritura por falta de permisos, pues esa ruta le debe pertenecer exclusivamente al usuario root. En algunos casos, este parámetro tiene asignado el valor "log", el cual, es relativo a la carpeta de Log, la cual, está inmersa en la carpeta de DATA de la instancia: `/var/lib/pgsql/<version_de_instancia>/data/log.` Esto termina siendo una mala práctica pues se afecta el I/O requerido por los Binarios de la instancia o las Bases de datos. No es recomendable utilizar mecanismos como "syslog" o "csvlog" para la captura de información en los Log de PostgreSQL dado que mucha información no es capturada correctamente o completamente por dichos procesos. **AUDITORIA:** A. Verificar configuración del directorio destino para el Log: `show log_directory;` **EVIDENCIAS:**	
4. <u>Permisos Correctos Sobre el Archivo de LOG:</u> Los permisos establecidos a los archivos de Log de la instancia pueden ser administrados y visualizados con el parámetro "log_file_mode". Comúnmente los archivos de Log contienen información sensible, por lo que si se tienen permisos desbordados o innecesarios, inadvertidamente se expone esta información a usuarios diferentes al administrador de la instancia. **NOTA:** En caso de requerir que un Aplicativo de terceros tenga acceso a estos archivos de Log, se hace necesario contar con la debida autorización por parte del director de TI u oficial de seguridad, con la aceptación de dicho riesgo. Se recomienda que el permiso sea: 0600, con lo cual, solo el usuario "postgres" debe tener permisos de lecto/escritura sobre los archivos de Log. **AUDITORIA:**	**Alta**

TAREA	CRITICIDAD
A. Verificar los permisos asignados a los archivos de Log: `show log_file_mode;` **EVIDENCIAS:**	
5. <u>Control del Tamaño, Nombre y Rotación del Archivo de LOG:</u> Una buena práctica requerida para el manejo de los Logs de una instancia es parametrizarlo de tal forma que la información contenida en los mismos sea referente a un corto periodo de tiempo, así, como realizar una rotación constante de estos para no contar con archivos de gran tamaño o complejos de manipular. Este control lo ejercemos con los parámetros: "log_truncate_on_rotation", "log_rotation_age" y "log_rotation_size". Lo anterior, va de la mano con el parámetro "log_filename", el cual, se requiere para dar formato al nombre del archivo de Log. Uniendo las configuraciones recomendadas para estos parámetros, permite establecer un control en: el nombre y los ciclos de rotación del Log. A continuación las recomendaciones para cada parámetro: ➢ log_filename: nombramiento para cada día del año: postgresql-%Y%m%d.log ➢ log_truncate_on_rotation: debe estar activo (on). ➢ log_rotation_age: establecido para periodos cortos de tiempo, para cada hora (1h) en ambientes altamente transaccionales, o para cada día (1d) valor por defecto. ➢ log_rotation_size: establecerlo en cero (0) si se hace uso del parametro "log_rotation_age", de lo contrario, establecerlo en (100M) para ambientes altamente transaccionales, o para un máximo de (500M) **NOTA:** En caso de no establecer el nombre de forma correcta, la instancia no podría ser iniciada o en su defecto estaría sobrescribiendo la información en el Log. Así mismo, si los 3 parámetros restantes no son bien configurados, el Log podría estar haciendo uso de un único archivo o sobrescribiendo la información un número limitado de archivos, por lo que se perdería el histórico del comportamiento del servicio en general. **AUDITORIA:** A. Verificar estado de la rotación del Log: `show log_truncate_on_rotation;` B. Verificar cada cuanto se debe rotar el Log:	**Alta**

TAREA	CRITICIDAD
show log_rotation_age; C. Determinar el nombramiento asignado al Log: show log_filename; D. Establecer si está activo (diferente de cero (0)) y su tamaño asignado: show log_rotation_size; **EVIDENCIAS:**	
6. Manejo Correcto de los Parámetros DEBUG: Desde la versión 9 se cuenta con varios parámetros de configuración alusivos a la opción de Debug utilizada mayormente por los desarrolladores para evaluar la ejecución de procesos o sentencias, detectar errores o eventos y proceder a solucionarlos. En todo ambiente productivo, se deben establecer de la siguiente forma: ➢ debug_print_parse: Deshabilitado/Off. ➢ debug_print_rewritten: Deshabilitado/Off. ➢ debug_print_plan: Deshabilitado/Off. El Habilitar estos 3 parámetros provocaría el registro de información confidencial en el Log que, de lo contrario, se omitiría en función de la configuración de los demás ajustes según los lineamientos en este documento. ➢ debug_pretty_print: Habilitado/On. Es parámetro si es viable mantenerlo habilitado, pues se encarga de capturar la información generada de los primeros 3 pero de forma concreta y segura, además, de hacer dicha información mucho más fácil de interpretar o leer. **NOTA:** El manejo inadecuado de estos parámetros, puede provocar afectaciones en el I/O, así, como crecimiento anormal del Log, incluso, capturando información sensible lo cual implica un riesgo de seguridad. **AUDITORIA:** A. Determinar el estado actual de los parámetros: show debug_print_parse; show debug_print_rewritten;	**Alta**

TAREA	CRITICIDAD
show debug_print_plan; show debug_pretty_print; **EVIDENCIAS:**	
7. Captura de Sesiones: Existen 2 parámetros de configuración que nos permiten capturar todo aquel intento de conexión/desconexión desde y hacia la instancia, además del tiempo de duración de las conexiones. **NOTA:** Este es el único mecanismo que se tiene, de forma preestablecida en la instancia, para capturar información de rastreo de conexiones y su tiempo de duración; PostgreSQL no almacena la información de Inicio o Fin de conexiones internamente para ser visualizado posteriormente, solo es posible con la activación de estos 2 parámetros. **AUDITORIA:** A. Determinar el estado actual de los parámetros: show log_connections; show log_disconnections; **EVIDENCIAS:**	**Media**
8. Captura Detallada en el LOG: La cantidad de detalle que se puede capturar en el Log es establecida por el parámetro "log_error_verbosity", el cual, acepta 3 valores: ➢ TERSE: este excluye la captura de información son respecto a: Hint, Detail, Query e información de contexto para los errores. ➢ DEFAULT: Valor por defecto indicando una mínima cantidad de información a ser registrada. ➢ VERBOSE: Incluye todo lo anterior, además, Sqlstate, código de errores, nombre de función, nombre de archivo de código fuente y numero de línea que genero el error. **NOTA:** No existe un valor erróneo para este parámetro, ya que depende de la cantidad básica o amplia de	**Media**

TAREA	CRITICIDAD
información detallada para errores, procesos, eventos y demás que la organización requiera capturar. **AUDITORIA:** A. Determinar el estado actual de los parámetros: `show log_error_verbosity;` **EVIDENCIAS:**	
9. Parámetro "log_line_prefix": Es utilizado para capturar información adicional en el Log la cual, se antepone a cada registro de información que deba ser ingresado en dicho Log. Esto es necesario para cuando se desea incluir información pertinente que permita realizar un seguimiento más adecuado de las conexiones de los usuarios o procesos en ejecución. **NOTA:** el valor por defecto de este parámetro es "`%m [%p]`", el cual solo permite capturar la fecha hora con milisegundos y el ID del proceso. En eventos de seguimientos o auditoria, incluso de análisis del comportamiento del servicio se hace muy complejo investigar de forma detallada cualquier evento, pues no se cuenta con la información suficiente. Esto no comprender una falla, pero sí, una falta de información completa a nivel administrativo del servicio. Los valores mínimos recomendados son: "`%m [%p]: [%l-1] db=%d, user=%u, app=%a, client=%h, State=%e, sstart=%s`" **AUDITORIA:** A. Verificar el valor actual asignado al parámetro: `show log_line_prefix;` **EVIDENCIAS:**	**Media**
10. Parámetro "*log_statement*": Es utilizado para establecer el tipo de sentencias SQL deben ser capturadas y registradas en el Log. Este parámetro recibe los siguientes valores:	**Media**

TAREA	CRITICIDAD
➢ None: Apagado. ➢ DDL: Registra las sentencias de tipo declaración de definición de datos. ➢ MOD: además de registrar DDL, también incluye las sentencias de modificación de datos, Insert, update y Delete, Truncate, Copy From. ➢ ALL: realiza el registro de absolutamente todas las sentencias (DDL + MOD). **NOTA:** el valor por defecto de este parámetro es "none", por lo cual, no se realiza la captura de ninguna sentencia. Esto representa una falla dado que no permite contar con la información necesaria para realizar el monitoreo y seguimiento adecuado del procesamiento sobre la instancia. Se recomienda que este parámetro cuente con el valor "DDL", puesto que la opción MOD y ALL generaran un alto impacto en el I/O sobre el recurso físico en el cual resida el Log. **AUDITORIA:** B. Verificar el valor actual asignado al parámetro: `show log_statement;` **EVIDENCIAS:**	

3. VULNERABILIDADES A NIVEL DE ARCHIVOS DE CONFIGURACIÓN CORE

TAREA	CRITICIDAD
1. Acceso a Archivos de Configuración "postgresql.conf" y "pg_hba.conf": El archivo postgresql.conf es el Core de configuraciones a nivel global de instancia, el cual, debe estar mínimamente protegido a nivel de permisos en el sistema operativo, restringiendo al máximo el acceso y visualización de este. Esto mismo ocurre con el archivo pg_hba.conf sobre el cual, se establecer todas las configuraciones de conectividad de los usuarios hacia el servicio o instancia de PostgreSQL. Puntos a tener en cuenta:	**Alta**

TAREA	CRITICIDAD

➢ El usuario y grupo propietario de este archivo deben ser el mismo propietario de la instalación el producto, para este caso, comúnmente es el usuario "postgres:postgres".
➢ Los permisos asociados a este archivo deberían ser:
o Lecto/escritura para el propietario (postgres)
o Lectura para el grupo propietario (postgres)
o Ninguno para el resto de los usuarios y/o grupos.
➢ Estos archivos debe ser respaldados con regularidad.
➢ Se debe contar con un historial de cambios sobre cada uno de estos archivos, ya sea, sobre cada uno a manera de comentarios o en su defecto, un archivo externo con la fecha e información del cambio asociado a cada archivo.
 	
NOTA: Al acceder a estos 2 archivos por parte de usuarios indebidos, dejaría totalmente vulnerable el servicio o instancia, así mismo, podría permitir acceso total y directo a toda la instancia y sus bases de datos.
 	
AUDITORIA:
A. Verificar las rutas actuales de los archivos de configuración:
`show config_file;`
`show hba_file;`
B. Establecer nivel de permisos de cada archivo:
`ls -la ruta_nombre_archivo_postgresql.conf`
`Ejemplo: (ls -la /etc/postgresql/13/main/postgresql.conf)`
`ls -la ruta_nombre_archivo_pg_hba.conf`
`Ejemplo: (ls -la /etc/postgresql/13/main/pg_hba.conf)`
 	
EVIDENCIAS:	

2. Configuración de Archivo "postgresql.conf": Al ser este un archivo Core de configuración de la instancia, es importante establecer puntos mínimos a tener en cuenta para determinar que se ajusta a lineamientos mínimos de seguridad y/o configuración del servicio.	**Alta**

TAREA	CRITICIDAD

➢ **listen_addresses**: si sobre el archivo pg_hba.conf se establecen las configuraciones correctas de conectividad, es viable asignar el valor "*" en este parámetro.
➢ **max_connections**: establece la cantidad de sesiones que se pueden establecer contra la instancia, importante evaluar la documentación del fabricante para establecer un valor correcto, dependiendo de los recursos físicos con los que se cuente.
➢ **superuser_reserved_connections**: muy importante establecer la cantidad de conexiones reservadas para el super administrador, con el fin, de lograr conexiones y entrar a resolver cualquier eventualidad, sin que todas sean consumidas por las aplicaciones o usuarios del servicio.
➢ **password_encryption**: este parámetro debe ir de la mano con lo establecido en el archivo pg_hba.conf, los valores posibles son: md5 o scram-sha-256.
➢ **Autovacuum**: Teniendo en cuenta que SETI implementa y monitorea procesos de mantenimiento para los ambientes de Bases de datos bajo administración, los cuales, son constantemente analizados en bsuca de ajustes necesarios, se recomienda que este parámetro este apagado (off), así, como los subsecuentes parámetros pertenecientes a este rubro en este archivo de configuración.
 	
Adicional a estos parámetros, es importante verificar los evaluados en este documento, los cuales, están registrados en su mayoría en el archivo postgresql.conf.
 	
NOTA: El no ajustar parámetros en este archivo no implica reglas adicionales por defecto, simplemente se obvian configuraciones necesarias o requeridas para el correcto funcionamiento del servicio o instancia.
 	
AUDITORIA:
A. Verificar los valores actuales para estos parámetros de configuración mediante psql:
`show listen_addresses;`
`show max_connections;`
`show superuser_reserved_connections;`
`show password_encryption;`
`show Autovacuum;`
B. Verificar los valores actuales para estos parámetros de configuración desde el sistema operativo (cat):
`cat ruta_nombre_archivo_postgresql.conf`
`Ejemplo: (cat /etc/postgresql/13/main/postgresql.conf)`	

TAREA	CRITICIDAD
EVIDENCIAS:	

3. Configuración de Archivo "pg_hba.conf":
Al ser este un archivo Core de configuración de la instancia, es importante establecer puntos mínimos a tener en cuenta para determinar que se ajusta a lineamientos mínimos de seguridad y/o configuración del servicio.
 	
Se hace necesario establecer que no se estén haciendo uso de métodos de autenticación poco seguros como: "trust" o "password", incluso, que no se establezcan nombres de Bases de datos, usuario e IP específica de conexión para cada caso, y en cambio, que se utilicen valores de configuración como "all" para el nombre de la base de datos o usuario de conexión.
 	
NOTA: El no ajustar parámetros en este archivo no implica reglas adicionales por defecto, simplemente se obvian configuraciones necesarias o requeridas para el correcto funcionamiento del servicio o instancia.
 	
AUDITORIA:
A. Verificar los valores actuales para estos parámetros de configuración desde el sistema operativo (grep):
`grep "trust" ruta_nombre_archivo_pg_hba.conf`
`grep "all" ruta_nombre_archivo_pg_hba.conf`
`grep "passwords" ruta_nombre_archivo_pg_hba.conf`
`Ejemplo: (grep "trust" /etc/postgresql/13/main/pg_hba.conf)`
 	
EVIDENCIAS:
	Alta

4. Uso de Conectividad en Archivo "pg_hba.conf": Para el motor de base de datos PostgreSQL el aseguramiento de conectividad se establece en 3 niveles: sobre el archivo pg_hba.conf, archivo postgresql.conf y a nivel de usuarios y roles sobre la instancia. Se hace necesario establecer, inicialmente, el primer nivel de seguridad en la estructura de conectividad mediante el archivo pg_hba.conf, la cual, está dividida en 2 niveles Unix Socket y TCP/IP.	**Alta**

TAREA	CRITICIDAD

NOTA: La omisión o mal ajuste de parámetros de configuración en el archivo pg_hba.conf deja abierta la posibilidad de violaciones de seguridad mediante diferentes mecanismos, por lo que los únicos niveles disponibles de aseguramiento es a nivel de: usuarios/roles/privilegios y acceso al servicio mediante vpn o firewall.

AUDITORIA:

Configuración a Nivel de Unix Socket:

Podría decirse que un inicio de sesión de host remoto, a través de ssh, es el medio más seguro para acceder y administrar el servidor PostgreSQL. Conexión con el cliente psql, a través de UNIX DOMAIN SOCKETS, utilizando el método de autenticación PEER siendo el mecanismo más seguro disponible para conexiones locales. Donde se proporciona una cuenta de usuario de base de datos con el mismo nombre del usuario UNIX, incluso, las cuentas de usuario ordinarias pueden acceder al clúster de una manera igualmente segura mientras estas sean locales y que tanto el usuario de SO sea igual al creado en la Instancia.

Un ejemplo de como debería lucir esta configuración en el archivo pg_hba.conf:

```
# Permite conección local de Usuario postgres vía UNIX socket
# TYPE    DATABASE   USER       ADDRESS   METHOD
  local   all        postgres             peer

# Permite conección local de todos los Usuario locales vía UNIX socket
# TYPE    DATABASE   USER       ADDRESS   METHOD
  local   all        all                  peer
# Permite conección local de todos los usuarios locales, siempre y cuando, sean miembros del
Rol 'localusers'
# TYPE    DATABASE   USER       ADDRESS   METHOD
  local   all        +localusers          peer
```

Configuración a nivel de TCP/IP:

Existe una gran cantidad de métodos de autenticación viables de ser utilizados para aquellas conexiones basadas en TCP/IP, como son: `reject`, `md5`, `scram-sha-256`, `gss`, `sspi`, `pam`, `ldap`, `radius` y `cert`.

Importante tener presente que NO es recomendable utilizar para este tipo de conexiones remotas los métodos: "`trust`", "`password`" e "`ident`". El método más recomendado es "**`scram-sha-256`**", pues No es vulnerable a

TAREA	CRITICIDAD

ataques de tipo repetición de paquetes o "`packet replay attacks`", como Si lo es el método "`md5`".

Los métodos de conexión: "`gss`, `sspi`, `pam`, `ldap`, `radius` y `cert`" aunque son más seguros que "`md5`" son totalmente dependientes de la disponibilidad de procesos o servicios externos de autenticación y su uso está bajo el criterio del área de seguridad y conectividad del cliente.

```
# Permite conexión remota de Usuario espcifico con contraseña encriptada
# TYPE    DATABASE   USER             ADDRESS        METHOD
  Host    all        nombre_usuario   172.10.12.26   scram-sha-256

# Permite conexión remota de todos los Usuario con contraseña encriptada
# TYPE    DATABASE   USER    ADDRESS         METHOD
  Host    all        all     172.10.12.26    scram-sha-256

# Permite conexión remota de todos los usuarios, siempre y cuando, sean miembros del Rol
'localusers' utilizando contraseña encriptada
# TYPE    DATABASE   USER          ADDRESS         METHOD
  Host    all        +localusers   172.10.12.26    scram-sha-256
```

En la columna "DATABASE" se puede establecer uno o varios nombres de Bases de datos, la opción "all" indica que se permite para todas las bases de datos en la instancia, siempre y cuando, el usuario tenga permisos.

En la columna "ADDRESS" se pueden utilizar las siguientes especificaciones:

- `127.0.0.1/32`: Para indicar que la conexión es local.
- `192.168.1.0-192.168.1.255`: este ejemplo permitirá conexiones entre el rango de IP establecido desde 1.0 hasta 1.255
- `172.10.12.26`: Para indicar que la conexión es exclusiva desde esta IP.
- `192.168.10.12/24`: también puede utilizar este método para establecer un rango.

EVIDENCIAS:

TAREA	CRITICIDAD

4. VULNERABILIDADES A NIVEL DE TABLESPACES

TAREA	CRITICIDAD
1. <u>TableSpace Temporales "temp tablespace":</u> En todo ambiente productivo se imperativo separar física y lógicamente el procesamiento (I/O) que demandan las operaciones sobre los objetos y Data de las bases de datos, del procesamiento volátil o aquel que permanece durante la vida útil de una sesión, los cuales, residen en el TableSpace Temporal. El archivo físico asociado al TableSpace temporal, debe ser totalmente independiente al Disco/LUN/FileSystem asociado al DATADIR. **NOTA:** El no generar una separación del procesamiento Temporal de aquel cotidiano en el servicio o instancia de PostgreSQL, impide realizar una identificación más acertada y rápida de cualquier evento o proceso temporal que este propiciado la afectación del servicio, incluso, que cada tipo de procesamiento se ejecute en su recuso físico exclusivo. **AUDITORIA:** B. Detectar configuración actual del TableSpace temporal: `show temp_tablespaces;` **EVIDENCIAS:**	**Alta**
2. <u>TableSpace para Data e Índices:</u> En el común de las bases de datos nos podemos encontrar con Tablas e índices que son mucho más demandantes, ya sea en espacio o en procesamiento, que el resto de estos objetos. Incluso, nos podemos encontrar con Tablas e Índices de tipo Históricos que son muy poco utilizados. Es entonces que debemos hacer	**Media**

TAREA	CRITICIDAD

uso de la configuración de diferentes TableSpace para separar los objetos más demandantes de aquellos que presentan un procesamiento regular y de aquellos de carácter histórico.
 	
NOTA: Esto representa un impacto en el comportamiento/performance general de la instancia y las bases de datos que residen en esta, impidiendo determinar exactamente qué base de datos o que proceso, son los que generan el mayor impacto en un servicio o instancia.
 	
AUDITORIA:
A. Verificar listado de TableSpace y su ubicación física:
`\db+`
B. Verificar listado de TableSpace y que bases de datos tienen objetos y cuáles de estos:
`SELECT t.spcname AS "Tablespace", array_to_string( ARRAY((SELECT datname FROM pg_database WHERE oid IN ( SELECT pg_tablespace_databases(t.oid) AS datoid ) ORDER BY 1)),',') AS "Database(s)" FROM pg_tablespace t WHERE t.spcname != 'pg_global' ORDER BY 1;`
 	
EVIDENCIAS:	

5. VULNERABILIDADES A NIVEL DE ROLES, USUARIOS Y PRIVILEGIOS

TAREA	CRITICIDAD
1. <u>Privilegios Administrativos:</u> Concerniente a PostgreSQL únicamente los Super Usuarios (DBA) deberían tener asignados permisos de administración elevados, por consiguiente, los usuarios no administradores y/o de aplicación no deberían tener asignados permisos como: Superuser, Create role, Create DB, Replication, Bypass RLS. **NOTA:** Asignar estos permisos a un usuario No Super Usuario (DBA) representa un alto fallo de seguridad puesto que se pone en riesgo incluso, la existencia o disponibilidad de la Data y del servicio mismo.	**Alto**

TAREA	CRITICIDAD

AUDITORIA:
A. listar permisos asignados al usuario postgres (Super Usuario de la instancia):
`\du postgres`
B. Listar todos los usuarios/roles y permisos asignados:
`select * from pg_user order by usename;`
`\du *`
 	
EVIDENCIAS:	

2. <u>Funciones con Permisos Excesivos:</u>
En un entorno productivo pueden existir objetos de programación como: funciones, procedimientos almacenados, triggers, etc. Con permisos elevados que les permiten ejecutar tareas administrativas por parte de usuarios con pocos privilegios, sin embargo, esto hace que dichos usuarios hereden altos privilegios administrativos de forma indirecta.
 	
NOTA: Idealmente, estos objetos de programación deberías estar vetados para no utilizar estos privilegios administrativos pues se puede llegar a incurrir en un riesgo de seguridad, en caso de no ser posible se debe levantar un acta de riesgo documentando el objeto que presenta este nivel de seguridad, estableciendo el porqué de esta necesidad y el uso que se hace del mismo, para finalmente contar con el visto bueno del responsable del servicio por parte del cliente.
 	
AUDITORIA:
A. listar los objetos de programación con altos privilegios (donde prosecdef = 't'):
`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;`
B. Listar los usuarios con permisos de ejecución sobre un objeto de programación:
`SELECT proname, proacl FROM pg_proc WHERE proname = 'nombre_objeto_programación';`
	Medio

TAREA	CRITICIDAD
EVIDENCIAS:	
3. <u>Uso de Roles para Asignación de Privilegios:</u> Con el fin de llevar un control adecuado y centralizado de los permisos asociados a los usuarios, es pertinente hacer uso de Roles, incluso, para aquellas actividades como: monitoreo general, monitoreo de estadísticas de uso de objetos, lecto/escritura por procesos desde la instancia, entre otros, la instancia cuenta con Roles preexistentes con los permisos específicos para realizar estas actividades. Se recomienda siempre empezar de menor a mayor cuando se trata de permiso, pues es un riesgo asignar permisos no necesarios a un usuario. **NOTA:** Al asignar privilegios directamente a los usuarios se puede perder fácilmente su control, incluso, no visualizar permisos que un usuario no debería tener, lo anterior, por la misma forma en que PostgreSQL muestra la estructura de permisos en pantalla. **AUDITORIA:** A. Verificar el listado de Roles actuales en la instancia: `select rolname from pg_roles;` B. Verificar el listado de Roles actuales en la instancia con el privilegio de SuperUser: `select rolname from pg_roles where rolsuper is true;` C. Listar todos los usuarios y sus roles asignados: `SELECT r.rolname,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 FROM pg_catalog.pg_roles r WHERE r.rolname NOT IN ('pg_signal_backend', 'rds_iam', 'rds_replication','rds_superuser','rdsadmin','rdsrepladmin') ORDER BY 1;` D. Listar todos los usuarios/roles y permisos asignados: `\du+` **EVIDENCIAS:**	**Media**
4. <u>Roles con Permisos de Conexión a la Instancia:</u> Teniendo en cuenta la sentencia que se utiliza en PostgreSQL para crear tanto usuarios como Roles es muy	**Alta**

TAREA	CRITICIDAD

común que se presenten errores al momento de crear los Roles y se les otorguen los permisos de conexión a la instancia o incluso, que no se les revoque después de creado.
 	
NOTA: Al permitir que un Role genere conexión a la instancia, no se está estableciendo un correcto aseguramiento del servicio y así dando pie a la generación de un alto riesgo de seguridad. Se debe garantizar que ningún Role tenga permisos de Conexión, en caso de ser requerido, se debe contar con un acta de aceptación del riesgo por parte del responsable justificando el porqué se requiere.
 	
AUDITORIA:
A. listar los Roles que tengan permitido conectarse a la instancia:
`SELECT rolname FROM pg_roles WHERE rolcanlogin;`
`SELECT rolname FROM pg_authid WHERE rolcanlogin;`
 	
EVIDENCIAS:	

6. VULNERABILIDADES A NIVEL DE CONSIDERACIONES ESPECIALES

TAREA	CRITICIDAD
1. <u>Procesos de Replicación, Alta Disponibilidad o Continuidad del Servicio:</u> Todo ambiente productivo debería contar con un mecanismo que permita ser recuperado de forma rápida y eficiente con la mínima perdida de información o en su defecto con mecanismos de continuidad del servicio. **NOTA:** De no contar con mecanismos de continuidad del servicio, alta disponibilidad o Replicación, los tiempos para recuperar el servicio podrían ser muy altos, incluso, dar pie a perdida de datos o de la totalidad de la base de datos. **AUDITORIA:**	**Alta**

TAREA	CRITICIDAD
7. Establecer qué tipo de servicio o proceso se tiene implementado en el ambiente productivo: - Replicación (Si/No): - Alta Disponibilidad (Si/No): - Continuidad del Servicio (Replica hacia otro DataCenter o Nube) (Si/No): **EVIDENCIAS:**	
2. Usuario y Permisos Específicos para Servicio de Replicación: Cuando se implementan servicios o procesos de Replicación se debe establecer un usuario asignado a este propósito, el cual, debe contar con los permisos específicos que le permita ejecutar exclusivamente la actividad propuesta. En algunas oportunidades se establece como usuario de Replicación, el mismo usuario propietario de la instancia para este caso "postgres", lo que representa una falla. Al hablar directamente de PostgreSQL, con tamos con un Rol especial para este propósito llamado "rolreplication" **NOTA:** Si no establecemos un usuario especifico con los permisos exclusivos para el proceso de Replicación, estamos propiciado la activación de un Riesgo de Seguridad. Si el proceso o servicio de Replicación requiere que el usuario asignado cuente con permisos administrativos o superiores, se debe establecer un acta de riesgo con el visto bueno del responsable indicando el porqué de esta necesidad. **AUDITORIA:** 8. Listar los usuarios con el rol "rolreplication" asignado: `select rolname from pg_roles where rolreplication is true;` **EVIDENCIAS:**	**Alta**
3. Archivado o Write Ahead Log (WAL): El Write Ahead Log o archivado de registros de escritura anticipada (WAL), es el proceso de envío de transacciones desde el host PRIMARIO a uno o más hosts STANDBY o Replicas o para ser archivados en un dispositivo de almacenamiento remoto para su uso posterior, como por ejemplo, recuperar la Base de datos en cualquier momento en el tiempo. Hay varias utilidades que pueden copiar WAL que incluyen, entre otros, cp, scp, sftp y rynsc. Básicamente, el servidor sigue un conjunto de parámetros de tiempo de ejecución que define	**Media**

TAREA	CRITICIDAD

cuándo se debe copiar el WAL usando alguna de las utilidades antes mencionadas. Uno de los métodos mas recomendados es RSYNC, para poder utilizarlo es necesario que el servicio de SSH este habilitado tanto en el Origen como en el Destino, además, confirmar que la llave privada o publica halla sido generada sobre el Home del Super Usuario de cada ambiente.
NOTA: El no establecer o configurar de manera correcta la transferencia de los Log o transacciones entre el origen (Primario) y el destino (Replica/Standby).
 	
AUDITORIA:
A. Listar los usuarios con el rol "rolreplication" asignado:
`Show archive_mode;`
`Show archive_command;`
B. Visualizar la llave o Key del SSH en Linux:
`cat ~/.ssh/id_rsa.pub`
`cat ~/.ssh/id_ecdsa.pub`
`cat ~/.ssh/ id_ed25519.pub`
`ls -al ~/.ssh`
 	
EVIDENCIAS:	

4. Rutas Físicas para Archivos de Configuración: Teniendo en cuenta que los intrusos buscan los archivos de configuración en rutas básicas conocidas como por ejemplo PGDATA o la ruta sobre la cual residen la instalación de la instancia, es importante, reubicarlos para salvaguardarlos y controlar un poco mas el acceso a los mismos. Los archivos básicos de configuración son: `postgresql.conf`, `pg_hba.conf` y `pg_ident`. **NOTA:** De no reubicarlos, sería muy fácil encontrarlos a nivel de sistema operativo, lo cual, representa una falla o riesgo. **AUDITORIA:** A. Verificar la ruta actual de los archivos de configuración básicos de la instancia:	**Media**

TAREA	CRITICIDAD

`select name, setting from pg_settings where name ~ '.*_file$';`
B. Verificar la existencia de "INCLUDES" activos y validar sus permisos (solo el usuario postgres, propietario de la instancia) y los usuarios exclusivamente autorizados deben tener acceso a los mismos:
`grep ^include $PGDATA/postgresql.{auto.,}conf`
EVIDENCIAS:	

5. <u>Utilidad de PostgreSQL para Backup y Restore:</u> Aunque en la actualidad existen múltiples herramientas de terceros que posibilitan y gestionan los Backup de las bases de datos, la recomendación, siempre va a ser utilizar aquellos recursos propios del producto o desarrollados por el mismo fabricante o referenciada/autorizada por este. Esta herramienta es gratis bajo licencia MIT, por lo que puede ser utilizada sin ningún tipo de restricción de licenciamiento para uso personal o comercial. **NOTA:** Esto no representa una falla, simplemente se suministra la información de su disponibilidad y posibilidad de uso para instancias PostgreSQL, por ser desarrollada específicamente para este producto. **AUDITORIA:** A. Verificar si la utilidad está instalada (ejecutar desde el sistema operativo): `pgbackrest` **EVIDENCIAS:**	**Baja**