WHITEPAPER

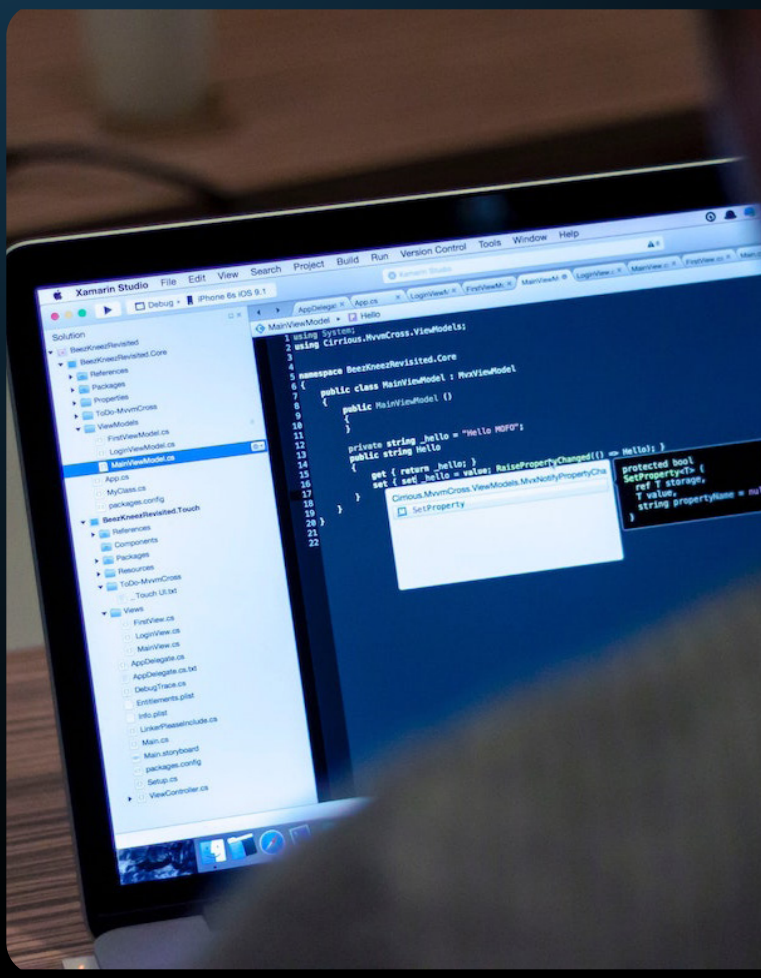
Bewährte Sicherheitspraktiken für PostgreSQL

Inhalt

1

Zusammenfassung

Dieses Whitepaper stellt einen Rahmen und eine Reihe von Empfehlungen zur Sicherung und zum Schutz einer PostgreSQL-Datenbank vor. Wir diskutieren ein mehrschichtiges Sicherheitsmodell, das sich mit physischer Sicherheit, Netzwerksicherheit, Host-Zugriffskontrolle, Datenbankzugriffsverwaltung und Datensicherheit befasst. Obwohl alle diese Aspekte gleich wichtig sind, konzentrieren wir uns auf PostgreSQL-spezifische Aspekte der Sicherung einer Datenbank und der darin enthaltenen Daten. Für unsere Ausführungen über die spezifischen Sicherheitsaspekte einer Datenbank und die in der Datenbank verwalteten Daten verwenden wir den AAA-Ansatz (Authentifizierung, Autorisierung und Auditierung), der für Computer- und Netzwerksicherheit üblich ist.



Der größte Teil der Empfehlungen in diesem Whitepaper gelten für PostgreSQL (die Community-Edition) und für EDB Postgres Advanced Server (Advanced Server), die kommerzielle Enterprise-Class Version von PostgreSQL von EnterpriseDB® (EDB) mit umfangreicher Funktionalität. Advanced Server bietet zusätzliche relevante Schutzfunktionalitäten, wie z. B. Passwort-Profile, Auditierung, Datenschwärzung und Schutz vor SQL Server Injektions-Angriffen (SQL Server Injection Protection), die in PostgreSQL nicht in der gleichen Form enthalten sind.

Die aktualisierte Version des Dokumentes beinhaltet die Sicherheitserweiterungen, die mit Postgres 12 sowie EDB Postgres Advanced Server 12 eingeführt sind

2

Einleitung

Wir können uns Sicherheit als Ebenen vorstellen und raten zu einer Strategie, die für jede Aufgabe oder Rolle den geringsten notwendigen Zugang gewährt und unnötigen Zugang so früh wie möglich blockiert.

- Als Erstes sollte der physische Zugang zum Host gesichert werden
- Dann sollte der Zugang zu Ihrem Unternehmensnetzwerk im Allgemeinen beschränkt werden
- Dann sollte der Zugriff auf den Datenbank-Cluster
- Dann sollte der Zugriff auf die Datenbankanwendung eingeschränkt werden
- Dann sollte der Zugang zu den Daten darin eingeschränkt werden
- Dann sollten die Daten darin gesichert werden

Allgemeine Empfehlungen

- Sorgen Sie dafür, dass Ihr Betriebssystem und Ihre Datenbank immer auf dem neuesten Stand sind. Die Support- Subscriptions von EDB bieten rechtzeitige Benachrichtigungen über Sicherheitsupdates und die richtigen Patches für Postgres. Für die Überwachung von Betriebssystem-Upgrades gibt es eine Vielzahl von Tools, die in Paketmanagementsysteme wie yum/dnf oder apt integriert werden können.

- Schützen Sie die Datenbank-Prozesse (insbesondere Postmaster-Prozesse) von unbeabsichtigten Zugriffen aus dem Internet. Falls die Datenbank aus dem Internet erreichbar sein sollte, müssen die strenge Firewall Regeln konfiguriert werden ; falls das nicht möglich ist, stellen Sie eine schreibgeschützte Standby-Datenbank auf dem Port zur Verfügung, statt eines Read-Write-Masters. Netzwerk-Port-Forwarding mit Auditierung aller Verbindungen stellt eine wirksame Alternative dar.

- Isolieren Sie den Datenbank-Port von anderem Netzwerkverkehr durch Subnetting oder andere Verfahren.

- Gewähren Sie Nutzern nur den Mindestzugriff, densie für ihre Arbeit benötigen, nicht mehr; reservieren Sie die Verwendung von Superuser-Konten für Aufgaben oder Rollen, wo das unbedingt erforderlich ist.

- Beschränken Sie den Zugriff auf Konfigurationsdateien (postgresql. conf und pg_hba.conf) und Protokolldateien (pg_log) auf Administratoren.

- Verbieten Sie die Anmeldung am Hostsystem mit Datenbank-Superuser-Rollen (postgres auf PostgreSQL, enterprisedb auf EDB Postgres Advanced Server). Aktivieren Sie den Superuser-Zugriff nur bei Bedarf und unter außergewöhnlichen Umständen.

- Stellen Sie jedem Nutzer eigene Anmeldekonten zur Verfügung; Anmeldekonten, die mit anderen geteilt werden, sind keine empfehlenswerte Praxis und erschweren die Auditierung. Verwenden Sie alternativ die edb_audit_tag-Fähigkeit (nur in EDB Postgres Advanced Server verfügbar), so dass Anwendungen weitere Audit-Informationen zu Sitzungen hinzufügen können, die aus Verbindungen auf Anwendungsebene stammen.

- Verlassen Sie sich nicht allein auf Ihre Front-End-Anwendung, um unbefugten Zugriff auf Ihre Datenbank zu verhindern; integrieren Sie Datenbanksicherheit mit Authentifizierungsund Autorisierungsmodellen auf Unternehmensebene, wie LDAP/AD oder Kerberos.

- Bewahren Sie Backups auf und legen Sie einen erprobten Wiederherstellungsplan an. Unabhängig davon, wie gut Sie Ihr System sichern, ist es immer noch möglich, dass jemand in Ihr System eindringt und Ihre Daten löscht oder verändert. Stellen Sie sicher, dass auch Ihre Backups sicher aufbewahrt werden, um unbefugten Zugriff zu verhindern.

Es ist vielleicht hilfreich, Sicherheit im Sinne des AAA-Modells zu betrachten, das für Netzwerk- und Computersicherheit entwickelt wurde. AAA steht für Authentifizierung, Autorisierung und Auditierung.

- **Authentifizierung: Überprüfung, ob der Nutzer die Person ist, für die er sich ausgibt.**
- **Autorisierung: Überprüfung, ob dem Nutzer der Zugriff gestattet ist.**
- **Auditierung (oder Accounting): Aufzeichnung aller Datenbankaktivitäten, einschließlich des Nutzernamens und der Uhrzeit in den Protokolldateien.**

Nicht alle Funktionen lassen sich sauber in diese Kategorien einordnen, aber das AAA-Modell bietet einen nützlichen Rahmen für diese Diskussion.

3

Verwendung der PostgreSQL-Sicherheitsfunktionen im AAA-System.

In den folgenden Abschnitten finden Sie detaillierte Abrisse, wie Sie PostgreSQL-Sicherheitsfunktionen in das AAA-System integrieren können.

3.1 Authentifizierung

Die Datei pg_hba.conf (PostgreSQL host-based access) beschränkt den Zugriff basierend auf dem Nutzernamen, der Datenbank und der Quell-IP (wenn der Nutzer sich über TCP/IP verbindet). In dieser Datei werden auch Authentifizierungsverfahren zugewiesen. Welches Authentifizierungsverfahren (oder welche Verfahren) Sie wählen, hängt von Ihrem Anwendungsfall ab.

Kerberos/GSSAPI — PostgreSQL unterstützt GSSAPI mit Kerberos-Authentifizierung gemäß RFC 1964. GSSAPI bietet automatische Authentifizierung (Single Sign-On) für Systeme, die das unterstützen. Die Authentifizierung selbst ist sicher, aber Daten, die über die Datenbankverbindung gesendet werden, sind unverschlüsselt, es sei denn, es wird GSS- oder SSL-Verschlüsselung angewendet.

SSPI — Verwenden Sie SSPI, wenn Sie ein Windows-System haben und Single-Sign-On (SSO)-Authentifizierung implementieren möchten.

LDAP sollte nur verwendet werden, wenn Kerberos (das sowohl SSPI als auch GSSAPI enthält) nicht in Frage kommt. LDAP ist nicht ganz so sicher wie Kerberos, da die Passwörter an den LDAP-Server weitergeleitet werden,und es ist leicht es unsicher einzurichten.

LDAP and RADIUS — LDAP und RADIUS sind nützlich, wenn Sie eine große Anzahl von Nutzern haben und Passwörter von einem zentralen Ort aus verwalten müssen. Diese Zentralisierung hat den Vorteil, dass Ihre Datei pg_hba.conf klein und somit überschaubarer bleibt. Auch gibt sie Ihren Nutzern ein „einheitliches Passwort-Erlebnis“ in Ihrer gesamten Infrastruktur. Sowohl LDAP als auch RADIUS erfordern eine solide Infrastruktur, da Sie für den Zugriff auf Ihre Datenbank auf den Dienst und die Konnektivität zu diesem Dienst angewiesen sind.

RADIUS sollte nicht verwendet werden, da es über eine schwache Verschlüsselung verfügt und md5-Hashing für die Zugangsdaten verwendet. Cert —TLS-Zertifikat-Authentifizierung (manchmal auch als SSL bezeichnet) kann zur Verschlüsselung des Datenverkehrs in der Verbindung und zur Authentifizierung verwendet werden. Zertifikate werden häufig in Machine-to-Machine-Kommunikation verwendet.

md5 — md5 speichert Informationen zu Nutzername und Passwort in der Datenbank, was eine geeignete Alternative sein kann, wenn Sie eine kleine Anzahl von Nutzern haben. Scram ist gegenüber md5 stark bevorzugt, da die Passwörter sicher gehasht werden.

Scram — Wenn Sie eine kleine Anzahl von vertrauenswürdigen Nutzern haben, können Sie scram-sha-256 zur Authentifizierung nutzen. Scram ist gegenüber md5 stark bevorzugt, da die Passwörter sicher gehasht werden.

Reject — Ablehnen: Verwenden Sie diese Methode, um bestimmte Nutzer, Verbindungen zu bestimmten Datenbanken und/oder bestimmte Quell-IPs abzulehnen.

Trust — Vertrauen: Die Trust-Authentifizierung sollte, wenn überhaupt, nur in Ausnahmefällen verwendet werden, da sie es einem passenden Client ermöglicht, sich ohne weitere Authentifizierung mit dem Server zu verbinden.

Es ist unbedingt erforderlich, dass Sie die Auswirkungen der einzelnen Authentifizierungsverfahren vollständig verstehen. In der PostgreSQL-Dokumentation finden Sie eine detailliertere Studie dieser und anderer Authentifizierungsverfahren.

Wie wir es schon in der Einleitung erwähnt haben, sollte der Zugriff auf die Datei pg_ hba.conf auf Administratoren beschränkt werden. Versuchen Sie, diese Datei so klein wie möglich zu halten; größere, kompliziertere Dateien sind schwieriger zu pflegen und enthalten mit größerer Wahrscheinlichkeit falsche oder veraltete Einträge. Überprüfen Sie diese Datei regelmäßig auf unnötige Einträge.

3.2 Password Profiles

Ab Version 9.5 unterstützt Advanced Server Oracle-kompatible Passwort-Profile bei Verwendung von MD5- oder SCRAM-Authentifizierung. Ein Passwort-Profil ist ein benannter Satz von Passwortattributen, die es einem DBA ermöglichen, eine Gruppe von Rollen, die vergleichbare Authentifizierungsanforderungen teilen, einfach zu verwalten. Jedes Profil kann mit einem oder mehreren Nutzern verknüpft werden. Wenn sich ein Nutzer mit dem Server verbindet, erzwingt der Server das Profil, das mit der Anmelderolle verknüpft ist. Weitere Informationen finden Sie im EDB's Database Compatibility for Oracle® Developer's Guide [Entwicklerhandbuch zur Datenbankkompatibilität für Oracle®] in Abschnitt 2.3 „Profile Management [Profilverwaltung]“, das hier verfügbar ist.

Profiles can be used to:

- die Anzahl der zulässigen fehlgeschlagenen Anmeldeversuche zu spezifizieren.
- ein Konto aufgrund von übermäßig vielen fehlgeschlagenen Anmeldeversuchen zu sperren.
- ein Ablaufdatum für ein Passwort zu setzen.
- eine Karenzzeit nach Ablauf eines Passworts zu definieren.
- Regeln für die Komplexität von Passwörtern zu definieren.
- Regeln, die die Wiederverwendung von Passwörtern einschränken, zu definieren.

3.3 Authorization

Wenn der Nutzer ordnungsgemäß authentifiziert wurde, müssen Sie Berechtigungen zum Anzeigen von Daten und zur Ausführung von Arbeiten in der Datenbank erteilen. Gewähren Sie, wie zuvor empfohlen, nur die Rechte, die ein Nutzer für die Ausführung einer Aufgabe benötigt, und verbieten Sie (Gruppen-)Anmeldedaten, die mit anderen geteilt werden. Verwalten Sie Nutzer und Gruppen in PostgreSQL über Rollenzuweisungen. Eine Rolle kann sich auf einen einzelnen Nutzer oder eine Gruppe von Nutzern beziehen. In Postgres werden Rollen auf der Cluster- (Datenbankserver-) Ebene erstellt. Das bedeutet, dass Rollen auf alle Datenbanken angewendet werden, die auf dem Cluster/Datenbankserver definiert sind; es ist daher wichtig, die Rollenberechtigungen entsprechend einzuschränken. Berechtigungen können auf Datenbankobjekte (Tabellen, Ansichten, Funktionen usw.), auf Zeilen innerhalb von Tabellen und auf Schwärzungsrichtlinien angewendet werden.

3.3.1 – Zugriff auf Datenbankobjekte

Die zugewiesenen Privilegien und Vorbehalte werden in der Dokumentation zu PostgreSQL CREATE ROLE beschrieben:

- Entziehen Sie allen Nutzern die CREATE-Privilegien und gewähren Sie sie nur vertrauenswürdigen Nutzern zurück.
- Erlauben Sie keine Verwendung von Funktionen oder Triggern, die in nicht vertrauenswürdigen prozeduralen Sprachen geschrieben sind.
- SECURITY DEFINER-Funktionen [Sicherheitsdefinitionsfunktionen] erlauben Nutzern die kontrollierte Ausführung von Funktionen auf einer höheren Berechtigungsebene, aber eine unachtsam geschriebene Funktion kann die Sicherheit unbeabsichtigter Weise verringern. Lesen Sie die Dokumentation (Abschnitt Writing Security Definer Functions Safely of CREATE FUNCTION [Sicheres Schreiben von Sicherheitsdefinitionsfunktionen der CREATE FUNCTION]) für weitere Einzelheiten.
- Datenbankobjekte sollten von einer sicheren Rolle verwaltet werden, idealerweise einer Rolle mit erheblich eingeschränktem Zugriff auf die Datenbank (z. B. nur von einem Unix Domain Socket aus), und nicht von einer Rolle, mit der sich ein Anwendungsnutzer verbinden kann. Dadurch wird die Möglichkeit minimiert, dass ein Angreifer Objekte verändern oder verwerfen kann. Während dies aus Sicherheitssicht bevorzugt wird, kann das bei Anwendungs-Frameworks, die das Schema selbst verwalten, problematisch sein - solche Funktionalitäten sollten mit Vorsicht implementiert werden.

Seien Sie sich bewusst, dass, wenn log_statement auf ‚ddl' oder höher gesetzt ist, die Änderung des Passworts einer Rolle über den Befehl ALTER ROLE dazu führt, dass das Passwort in den Protokollen aufgedeckt wird, außer in EDB Postgres Advanced Server 11 oder höher, wo der Befehl edb_filter_log.redact_password_command den Server anweist, gespeicherte Passwörter aus der Protokolldatei zu löschen. Klicken Sie hier für weitere Einzelheiten.

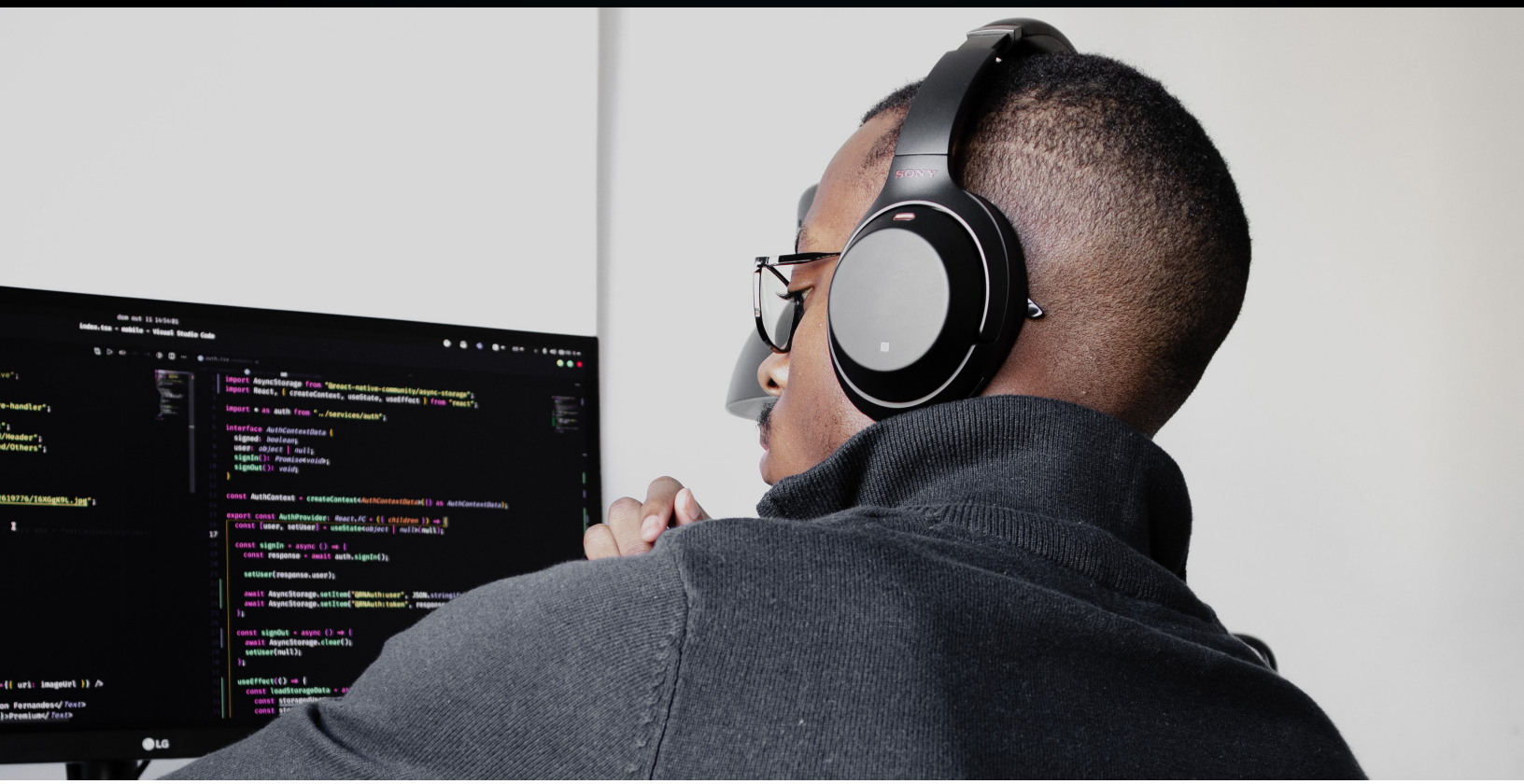
Wenn Authentifizierungsdaten (z. B. Nutzernamen und Passwörter) in einer Tabelle gespeichert sind, kann die Verwendung der Statement-Protokollierung diese Informationen aufdecken, selbst wenn die Tabelle prinzipiell sicher ist. Ähnlich verhält es sich, wenn sensible Informationen in Queries verwendet werden (z. B. jede Art von persönlich identifizierbaren Daten als Schlüssel); diese Parameter können durch Statement-Protokollierung

3.3.2 – Ansichten

Der Zugriff auf die Ansichten [Views] kann wie oben beschrieben kontrolliert werden (es handelt sich um Datenbankobjekte), und Ansichten können dazu verwendet werden, die Sichtbarkeit von Daten auf bestimmte Nutzergruppen zu beschränken, indem ein VIEW einer Tabelle erstellt und die Berechtigungen für diesen VIEW eingeschränkt werden. Die PostgreSQL Versionen 9.2 oder höher bieten die Option CREATE VIEW WITH (security_barrier für den Fall, dass zusätzliche Sicherheitsmaßnahmen als notwendig erachtet werden, um mögliche Sicherheitsprobleme, wie die von Robert Haas beschriebenen, zu vermeiden.

3.3.3 – Sicherheit auf Zeilenebene (RLS,Row Level Security)

PostgreSQL hat Sicherheit auf Zeilenebene (RLS, Row Level Security) in Version 9.5 eingeführt. RLS erlaubt detaillierten Zugriff auf Tabellenzeilen basierend auf der aktuellen Nutzerrolle. Dazu gehören SELECT-, UPDATE-, DELETE- und INSERT-Operationen [Auswählen, Aktualisieren, Löschen und Einfügen]. Weitere Informationen finden Sie hier. EDB Postgres Advanced Server enthält eine Oracle-kompatible Implementierung dieses Verfahrens in seinem DBMS_RLS-Paket, das auch Oracle-kompatible Implementierungen von ADD_POLICY, DROP_POLICY und UPDATE_POLICY enthält. Weitere Informationen finden Sie hier .



3.3.4 – Datenschwärzung

Datenschwärzung - die Möglichkeit, einige Datenelemente auszublenden oder Daten für bestimmte Nutzergruppen selektiv unlesbar zu machen, ist ein weiteres Verfahren zur Verwaltung des Datenzugriffs. EDB Postgres Advanced Server hat Datenschwärzung in Version 11 eingeführt.

Datenschwärzung ist ein regelbasiertes Tool, das auf PostgreSQL-Rollen beruht, um Lesezugriff auf bestimmte Datenelemente zu gewähren oder zu entziehen. Zum Beispiel sieht eine Nutzergruppe Sozialversicherungsnummern als XXX-XX-1235, während Datenverwaltungsrollen die gesamten Informationen sehen. Hier finden Sie zusätzliche Informationen zur Datenschwärzung.

Konstante	Typ	Wert	Beschreibung
NONE	INTEGER	0	Keine Schwärzung, keine Auswirkung auf das Ergebnis einer Abfrage gegen eine Tabelle.
FULL	INTEGER	1	Volle Schwärzung, schwärzt volle Werte der Spaltendaten.
PARTIAL	INTEGER	2	Teilweise Schwärzung, schwärzt einen Teil der Spaltendaten.
RANDOM	INTEGER	4	Zufallsschwärzung, jede Abfrage führt je nach Datentyp der Spalte zu einem anderen Zufallswert.
REGEXP	INTEGER	5	Auf regulärem Ausdruck basierende Schwärzung, sucht nach dem Muster der zu schwärzenden Daten.
CUSTOM	INTEGER	99	Benutzerdefinierter Schwärzungstyp.

3.13.1 Verwendung der DBMS_REDACT-Konstanten und Funktionsparameter

3.4 Auditierung

Advanced Server bietet die Möglichkeit, Audit-Berichte zu erstellen. Datenbank-Audits ermöglichen es Datenbank-Administratoren, -Auditoren und -Betreibern, Datenbankaktivitäten zur Unterstützung komplexer Audit-Anforderungen zu verfolgen und zu analysieren. Diese auditierten Aktivitäten umfassen den Datenbankzugriff und die Datenbanknutzung sowie die Erstellung, Änderung oder Löschung von Daten. Das Audit-System basiert auf Konfigurationsparametern, die in der Konfigurationsdatei definiert sind.

Wir empfehlen, dass Sie Folgendes auditieren (nach aufsteigendem Sicherheitsniveau sortiert):

- Nutzerverbindungen
- DDL-Änderungen
- Datenänderungen
- Daten-Ansichten

Sehr strenge Kontrollmaßnahmen können zu vielen Protokollmeldungen führen; protokollieren Sie nur so viel, wie Sie benötigen. Mit Postgres können Sie die Protokollierung auf Nutzer- und Datenbankbasis anpassen. Überprüfen Sie Ihre Audit-Protokolle häufig auf anomales Verhalten. Richten Sie eine Überwachungskette für Ihre Protokolle ein.

Denken Sie daran, dass eine hohe Protokollierungsstufe zusammen mit der Speicherung von Passwörtern in der Datenbank dazu führen kann, dass Passwörter in den Protokollen angezeigt werden. EDB Postgres Advanced Server hat in Version 11 die Erweiterung edb_filter_log.redact_ password_commands eingeführt, um den Server anzuweisen, gespeicherte Passwörter aus der Audit-Protokolldatei zu löschen.

Hier finden Sie weitere Informationen über die Audit-Protokollfunktion von Advanced Server.

> Advanced Server ermöglicht es Datenbank- und Sicherheitsadministratoren, Auditoren und Betreibern, Datenbankaktivitäten mit Hilfe der EDB-Audit-Logging-Funktionalität zu verfolgen und zu analysieren.

3.5 Datenverschlüsselung

PostgreSQL bietet Verschlüsselung auf mehreren Ebenen und bietet Flexibilität beim Schutz vor Offenlegung von Daten aufgrund von Datenbankserver-Diebstahl, unseriösen Administratoren und unsicheren Netzwerken:

- Nutzerverbindungen
- DDL-Änderungen
- Datenänderungen
- Daten-Ansichten

Sie können m ehr über diese Optionen in der **PostgreSQL-Dokumentation** nachlesen .

Wenn Sie Bedenken haben, dass ein Sniffer während der Übertragung zwischen einem Client und der Datenbank Daten lesen könnte, aktivieren Sie SSL in der Datei postgresql.conf, es sei denn, Sie können sicher sein, dass dies kein Risiko darstellt. Obwohl die SSL-Verschlüsselung einige zusätzliche Kosten verursachen und die Verwaltung von Zertifikaten schwierig sein kann, ist dies im Allgemeinen eine empfehlenswerte Praxis.

Sie können auch Daten innerhalb der Datenbank oder auf Dateisystemebene (entweder oder) verschlüsseln. Weitere Informationen über **transparente Datenverschlüsselung finden Sie auf dem EDB-Blog**. Bei dieser Verschlüsselung werden die Daten entschlüsselt, wenn sie aus dem Dateisystem gelesen werden, so dass DB-Administratoren Daten anzeigen können. Es ist unbedingt erforderlich, dass Rollen und Berechtigungen festgelegt sind. Zu den weiteren Optionen gehört die Verwendung von **Thales Vormetrischer Transparenter Verschlüsselung (VTE)**.

Verwenden Sie das Modul pgcrypto contrib, um Daten spaltenweise zu verschlüsseln. Dieses Verfahren hat einige Nachteile:

- Es zieht eine potenzielle Leistungseinbuße mit sich, abhängig von der Größe der Tabelle.
- Die verschlüsselten Felder können nicht durchsucht oder indiziert werden.
- Die Verschlüsselung muss zur Zeit der Tabellenerstellung angewendet werden. Das erfordert Vorausplanung.
- Die Verwaltung der Schlüssel kann auch aufwändig sein.

Zusätzlich muss Ihre Anwendung die Ver-/Entschlüsselung so handhaben, dass jeder Austausch mit der Datenbank verschlüsselt bleibt, um einen unseriösen DBA daran zu hindern, Daten anzuzeigen.

3.6 SQL-Injektions-Angriffe

Ein SQL-Injektions-Angriff ist ein Versuch, eine Datenbank zu kompromittieren, indem SQL-Befehle ausgeführt werden, die dem Angreifer Hinweise auf den Inhalt, die Struktur oder die Sicherheit der Datenbank liefern. Die Verhinderung eines SQL-Injektions-Angriffs liegt normalerweise in der Verantwortung des Entwicklers der Anwendung. Datenbankadministratoren haben normalerweise wenig oder gar keine Kontrolle über dieses potenzielle Risiko.

Die Standardmethode zur Verhinderung von SQL-Injektions-Angriffen in PostgreSQL ist die Parametrisierung von Queries. Wenn Sie EDB Postgres Advanced Server verwenden, empfehlen wir Ihnen die Verwendung des SQL/Protect-Moduls zum Schutz vor SQL-Injektions-Angriffen. SQL/Protect bietet eine zusätzliche Sicherheitsebene zu den normalen Datenbank-Sicherheits-Policies, indem eingehende Queries auf gängige SQL-Profile untersucht werden. SQL/Protect gibt dem Datenbankadministrator die Kontrolle zurück, indem es den Administrator vor potenziell gefährlichen Queries warnt und diese Queries blockiert. Für weitere Informationen klicken Sie hier.

```
shared_preload_libraries = '$libdir/dbms_pipe,$libdir/edb_gen,$libdir/sqlprotect'
                                        # (diese Änderung erfordert einen Neustart)
        .
        .
        .

edb_sql_protect.enabled = off edb_sql_protect.level = learn
edb_sql_protect.max_protected_roles = 64
edb_sql_protect.max_protected_relations = 1024
edb_sql_protect.max_queries_to_save = 5000
```

4.1.2 Konfigurieren von SQL/Protect

5

Weiterführende Lektüre

- EDB Security Technical Implementation Guidelines (STIG) for PostgreSQL on Windows and Linux EDB-Richtlinien zur technischen Implementierung der Sicherheit (STIG) in PostgreSQL auf Windows und Linux

- Blog: How to Secure PostgreSQL: Security Hardening Best Practices & Tips So sichert man PostgreSQL: Bewährte Praktiken und Tipps zur Verbesserung der Sicherheit

- Blog: Managing Roles with Password Profiles: Rollen mit Passwort-Profilen verwalten Teil 1

- Blog: Managing Roles with Password Profiles: Rollen mit Passwort-Profilen verwalten Teil 2

- Blog: Managing Roles with Password Profiles: Rollen mit Passwort-Profilen verwalten Teil 3

WHITEPAPER

Bewährte Sicherheitspraktiken für PostgreSQL

EnterpriseDB Corporation
34 Crosby Drive
Suite 201
Bedford, MA 01730, USA

EDB™

EDBPOSTGRES.COM