



EDB

Security Features Guide

リリース 13

EnterpriseDB

2020 年 11 月 23 日

目次

第 1 章	はじめに	1
第 2 章	SQL インジェクション攻撃からの保護	3
2.1	SQL/Protect の概要	4
2.1.1	SQL インジェクション攻撃の種類	4
2.1.2	SQL インジェクション攻撃の監視	6
2.1.2.1	保護された役割	6
2.1.2.2	攻撃試行統計	6
2.1.2.3	攻撃試行クエリ	7
2.2	SQL/Protect の構成	8
2.2.1	保護する役割の選択	10
2.2.1.1	保護された役割リストの設定	11
2.2.1.2	保護レベルの設定	12
2.2.2	保護されたロールの監視	13
2.2.2.1	学習モード	13
2.2.2.2	パッシブモード	15
2.2.2.3	アクティブモード	18
2.3	一般的なメンテナンス操作	20
2.3.1	保護された役割リストへの役割の追加	20
2.3.2	保護された役割リストからの役割の削除	20
2.3.3	役割の保護のタイプを設定する	21
2.3.4	保護された関係リストからの関係の削除	21
2.3.5	統計の削除	22
2.3.6	問題のあるクエリを削除する	23
2.3.7	監視の無効化と有効化	24
2.4	SQL/Protect データベースのバックアップと復元	25
2.4.1	SQL/Protect テーブルのオブジェクト識別番号	25
2.4.2	データベースのバックアップ	26
2.4.3	バックアップファイルからの復元	26
第 3 章	仮想プライベートデータベース	33
第 4 章	sslutils	35

4.1	openssl_rsa_generate_key	35
4.2	openssl_rsa_key_to_csr	35
4.3	openssl_csr_to_crt	36
4.4	openssl_rsa_generate_crl	37
第 5 章	データ編集	39
5.1	編集ポリシーの作成	41
5.2	ALTERREDACTIONPOLICY	46
5.3	ドロップ修正ポリシー	49
5.4	システムカタログ	50
5.4.1	edb_redaction_column	50
5.4.2	edb_redaction_policy	51
第 6 章	結論	53
索引		55

第 1 章

はじめに

このガイドでは、EDBPostgresAdvancedServer インストールにセキュリティを追加する機能について説明します。AdvancedServer に組み込まれている PostgreSQL が提供するセキュリティ機能の包括的なガイドではありません。

- *SQL/Protect* は、SQL インジェクション攻撃からの保護を提供します。
- *VirtualPrivateDatabase* は、機密データへのきめ細かいアクセス制御を提供します。
- *sslutils* は、SSL 証明書を生成できる Postgres 拡張機能です。
- *データ編集* 機能により、データの一部を動的にマスクできます。

Postgres の認証およびセキュリティ機能については、次の場所にある PostgreSQL コアドキュメントを参照してください。

<https://www.enterprisedb.com/edb-docs/p/postgresql>

第 2 章

SQL インジェクション攻撃からの保護

AdvancedServer は、SQL インジェクション攻撃に対する保護を提供します。SQL インジェクション攻撃は、データベースの内容、構造、またはセキュリティに関する結果を攻撃者に提供する SQL ステートメントを実行することにより、データベースを侵害する試みです。

SQL インジェクション攻撃の防止は、通常、アプリケーション開発者の責任です。データベース管理者は通常、潜在的な脅威をほとんどまたはまったく制御できません。データベース管理者にとっての困難は、アプリケーションが適切に機能するためにデータにアクセスできる必要があることです。

SQL/Protect は、データベース管理者がデータベースを SQL インジェクション攻撃から保護できるようにするモジュールです。SQL/Protect は、一般的な SQL インジェクションプロファイルの着信クエリを調べることにより、通常のデータベースセキュリティポリシーに加えてセキュリティのレイヤーを提供します。

SQL/Protect は、潜在的に危険なクエリを管理者に警告し、これらのクエリをブロックすることにより、データベース管理者に制御を戻します。

2.1 SQL/Protect の概要

このセクションでは、さまざまなタイプの SQL インジェクション攻撃の概要と、SQL/Protect がそれらを防御する方法について説明します。

2.1.1 SQL インジェクション攻撃の種類

SQL インジェクション攻撃を実行するために使用されるさまざまな手法があります。各手法は、特定の signature によって特徴付けられます。SQL/Protect は、次のシグネチャのクエリを調べます。

不正な関係

AdvancedServer では、管理者が関係（テーブル、ビューなど）へのアクセスを制限できますが、多くの管理者はこの退屈なタスクを実行しません。SQL/Protect は、ユーザーがアクセスする関係を追跡する learn モードを提供します。

これにより、管理者はアプリケーションのワークロードを調べることができ、SQL/Protect は、特定のユーザーまたはロール内のユーザーグループに対して、アプリケーションがアクセスできる関係を知ることができます。

SQL/Protect が passive `` または ``active モードのいずれかに切り替えられると、受信したクエリは学習された関係のリストに対してチェックされます。

ユーティリティコマンド

SQL インジェクション攻撃で使用される一般的な手法は、通常は SQL データ定義言語（DDL）ステートメントであるユーティリティコマンドを実行することです。例としては、他のシステムリソースにアクセスする機能を持つユーザー定義関数の作成があります。

SQL/Protect は、すべてのユーティリティコマンドの実行をブロックできます。これらのコマンドは、標準のアプリケーション処理では通常必要ありません。

SQL トートロジー

SQL インジェクション攻撃で最も頻繁に使用される手法は、トートロジー的な WHERE 句条件を発行することです（つまり、常に true の条件を使用します）。

次に例を示します。

```
WHERE password = 'x' OR 'x'='x'
```

攻撃者は通常、この手法を使用してセキュリティ上の弱点を特定し始めます。SQL/Protect は、トートロジー条件句を使用するクエリをブロックできます。

無制限の DML ステートメント

SQL インジェクション攻撃中に取られる危険なアクションは、無制限の DML ステートメントの実行です。これらは、WHERE `` 句のない ``UPDATE `` および ``DELETE ステートメントです。たとえば、攻撃者はす

すべてのユーザーのパスワードを既知の値に更新したり、キーテーブル内のすべてのデータを削除してサービス拒否攻撃を開始したりできます。

2.1.2 SQL インジェクション攻撃の監視

このセクションでは、SQL/Protect が SQL インジェクション攻撃を監視および報告する方法について説明します。

2.1.2.1 保護された役割

SQL インジェクション攻撃の監視には、セッションの現在のユーザーが保護されたロールであるデータベースセッションで発生した SQL ステートメントの分析が含まれます。protected role `` は、データベース管理者が SQL/Protect を使用して監視することを選択した AdvancedServer ユーザーまたはグループです。(AdvancedServer では、ユーザーとグループをまとめて ``roles と呼びます。)

保護される各ロールは、監視対象の SQL インジェクション攻撃のタイプに合わせてカスタマイズできるため、ロールごとに異なるレベルの保護を提供し、DBA のユーザーメンテナンスの負荷を大幅に削減できます。

スーパーユーザー特権を持つロールを保護されたロールにすることはできません。保護された非スーパーユーザーロールがその後スーパーユーザーになるように変更された場合、そのスーパーユーザーがコマンドを発行しようとするたびに特定の動作が発生します。

- 保護されたスーパーユーザーによって発行されたすべてのコマンドで、SQL/Protect によって警告メッセージが発行されます。
- edb_sql_protect_stats `` の列スーパーユーザーの統計は、保護されたスーパーユーザーが発行するコマンドごとに増加します。 ``edb_sql_protect_stats `` ビューの詳細については、 ``Attack Attempt Statistics を参照してください。
- SQL/Protect がアクティブモードの場合、保護されたスーパーユーザーによって発行されたすべてのコマンドは実行できなくなります。

スーパーユーザー特権を持つ保護されたロールは、スーパーユーザーでなくなるように変更するか、保護されていないロールに戻す必要があります。

2.1.2.2 攻撃試行統計

SQL/Protect による攻撃と見なされる保護されたロールによるコマンドの使用が記録されます。統計は、Types of SQL Injection Attacks で説明したように、SQL インジェクション攻撃の種類ごとに収集されます。

これらの統計は、潜在的な攻撃の開始を識別するために簡単に監視できるビュー edb_sql_protect_stats からアクセスできます。

edb_sql_protect_stats の列は以下を監視します:

- **username** 保護されたロールの名前。
- **スーパーユーザー** 保護されたロールがスーパーユーザーの場合に発行される SQL ステートメントの数。実際、保護されたスーパーユーザーによって発行された SQL ステートメントは、この統計を増加させます。

保護されたスーパーユーザーについては、Protected Roles を参照してください。

- **relations** 保護されたロールによって学習されなかった関係を参照して発行された SQL ステートメントの数。(つまり、ロールの保護された関係リストにない関係。)
- **command** 保護されたロールによって発行された DDL ステートメントの数。
- **tautology** トートロジー条件を含む保護されたロールによって発行された SQL ステートメントの数。
- **dml** WHERE `` 句を含まない保護されたロールによって発行された ``UPDATE `` および ``DELETE ステートメントの数。

これにより、データベース管理者は、貴重なデータやその他の悪意のあるアクションの盗難を防止するために積極的に対応する機会を得ることができます。

ロールが複数のデータベースで保護されている場合、各データベースでの攻撃のロールの統計は個別に保持され、それぞれのデータベースに接続されている場合にのみ表示されます。

注釈: SQL/Protect 統計は、データベースサーバーの実行中にメモリに保持されます。データベースサーバーがシャットダウンすると、統計は AdvancedServer ホームディレクトリの data/global `` サブディレクトリにある ``edb_sqlprotect.stat という名前のバイナリファイルに保存されます。

2.1.2.3 攻撃試行クエリ

SQL/Protect による攻撃と見なされる保護されたロールによるコマンドの使用はそれぞれ、edb_sql_protect_queries ビューに記録されます。

edb_sql_protect_queries ビューには次の列が含まれます:

- **username** データベースサーバーへのログインに使用される攻撃者のデータベースユーザー名。
- **ip_address** 攻撃が開始されたマシンの IP アドレス。
- **port** 攻撃の発信元のポート番号。
- **machine_name** 攻撃が発生したマシンの名前 (わかっている場合)。
- **date_time** データベースサーバーがクエリを受信した日時。時間は 1 分の精度で保存されます。
- **query** 攻撃者が送信したクエリ文字列。

edb_sql_protect_queries `` に保存される問題のクエリの最大数は、``edb_sql_protect.max_queries_to_save 構成パラメーターによって制御されます。

ロールが複数のデータベースで保護されている場合、各データベースでの攻撃に対するロールのクエリは個別に保持され、それぞれのデータベースに接続されている場合にのみ表示されます。

2.2 SQL/Protect の構成

SQL/Protect を構成する前に、次の前提条件が満たされていることを確認してください。

- SQL/Protect `` の実行に必要なライブラリファイル (Linux では ``0'', Windows では ``sqlprotect.dll ``) は、AdvancedServer ホームディレクトリの ``lib `` サブディレクトリにインストールする必要があります。Windows の場合、これは AdvancedServer インストーラーで実行する必要があります。Linux の場合、 ``edb-asxx-server-sqlprotect `` RPM パッケージをインストールします (``xx は AdvancedServer のバージョン番号です)。
- AdvancedServer ホームディレクトリの share/contrib `` サブディレクトリにある SQL スクリプトファイル ``sqlprotect.sql も必要です。
- SQL/Protect `` を使用するようにデータベースサーバーを構成する必要があります。また、 ``SQL/Protect で監視する各データベースを構成する必要があります。
 - データベースサーバー設定ファイル postgresql.conf `` は、 ``SQL/Protect で使用される設定パラメーターを追加して有効にすることで変更する必要があります。
 - SQL/Protect `` で使用されるデータベースオブジェクトは、 ``SQL/Protect で監視する各データベースにインストールする必要があります。

ステップ 1: AdvancedServer ホームディレクトリの data `` サブディレクトリにある ``postgresql.conf ファイルで、以下の設定パラメーターを編集します。

- **shared_preload_libraries** ライブラリのリストに \$libdir/sqlprotect を追加します。
- **edb_sql_protect.enabled** 保護されたロールによって発行された SQL ステートメントを分析し、 edb_sql_protect.level `` の設定に従って反応することにより、 ``SQL/Protect `` が保護されたロールをアクティブに監視するかどうかを制御します。 ``SQL/Protect `` で監視を開始する準備ができたなら、このパラメーターを ``on `` に設定します。このパラメーターが省略された場合、デフォルトは ``off です。
- **edb_sql_protect.level** 保護されたロールによって SQL ステートメントが発行されたときに SQL/Protect `` によって実行されるアクションを設定します。このパラメーターが省略された場合、デフォルトの動作は ``passive `` です。最初に、このパラメーターを ``learn に設定します。

詳細については、 [保護レベルの設定](#) を参照してください。

- **edb_sql_protect.max_protected_roles** 保護できるロールの最大数を設定します。このパラメーターが省略された場合、デフォルト設定は 64 です。
- **edb_sql_protect.max_protected_relations** ロールごとに保護できるリレーションの最大数を設定します。このパラメーターが省略された場合、デフォルト設定は 1024 です。

サーバーの保護された関係の総数は、保護された関係の数に保護されたロールの数を掛けたものになることに注意してください。保護された関係はすべて、共有メモリのスペースを消費します。可能な限り最大の保

護された関係のスペースは、データベースサーバーの起動時に予約されます。

- **edb_sql_protect.max_queries_to_save** edb_sql_protect_queries `` ビューに保存する問題のあるクエリの最大数を設定します。このパラメーターを省略すると、デフォルト設定は ``5000 です。問題のあるクエリが制限に達すると、追加のクエリはビューに保存されませんが、データベースサーバーのログファイルでアクセスできます。

このパラメーターの最小有効値は 100 `` です。 ``100 `` より小さい値が指定された場合、データベースサーバーは ``5000 のデフォルト設定を使用して起動します。警告メッセージがデータベースサーバーのログファイルに記録されます。

次の例は、postgresql.conf ファイルのこれらのパラメーターの設定を示しています。

```
shared_preload_libraries = '$libdir/dbms_pipe,$libdir/edb_gen,$libdir/
sqlprotect'
                                # (change requires restart)
                                .
                                .
                                .
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
```

ステップ 2: postgresql.conf ファイルを変更した後、データベースサーバーを再起動します。

Linux の場合: restart オプションで AdvancedServer サービススクリプトを呼び出します。

Redhat または CentOS6.x インストールでは、次のコマンドを使用します。

```
/etc/init.d/edb-as-13 restart
```

Redhat または CentOS7.x インストールでは、次のコマンドを使用します。

```
systemctl restart edb-as-13
```

Windows の場合: Windows サービスアプレットを使用して、edb-as-13 という名前のサービスを再起動します。

ステップ 3: SQL インジェクション攻撃から保護するデータベースごとに、スーパーユーザー（インストールオプションに応じて enterprisedb `` または ``postgres `` ）としてデータベースに接続し、AdvancedServer ホームディレクトリの ``share/contrib `` サブディレクトリにあるスクリプト ``sqlprotect.sql ``。このスクリプトは、 ``sqlprotect という名前のスキーマに SQL/Protect データベースオブジェクトを作成します。

次の例は、edb という名前のデータベースの保護を設定するこのプロセスを示しています。

```
$ /usr/edb/as13/bin/psql -d edb -U enterprisedb
Password for user enterprisedb:
psql.bin (13.0.0, server 13.0.0)
Type "help" for help.

edb=# \i /usr/edb/as13/share/contrib/sqlprotect.sql
CREATE SCHEMA
GRANT
SET
CREATE TABLE
GRANT
CREATE TABLE
GRANT
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
DO
CREATE FUNCTION
CREATE FUNCTION
DO
CREATE VIEW
GRANT
DO
CREATE VIEW
GRANT
CREATE VIEW
GRANT
CREATE FUNCTION
CREATE FUNCTION
SET
```

2.2.1 保護する役割の選択

データベースに SQL/Protect データベースオブジェクトを作成したら、保護のために SQL クエリを監視するロールと、各ロールに割り当てる保護レベルを選択できます。

2.2.1.1 保護された役割リストの設定

保護するデータベースごとに、監視する役割を決定し、それらの役割をそのデータベースの 保護された役割リストに追加する必要があります。

ステップ 1: psql または PostgresEnterpriseManagerClient で保護したいデータベースにスーパーユーザーとして接続します:

```
$ /usr/edb/as13/bin/psql -d edb -U enterprisedb
Password for user enterprisedb:
psql.bin (13.0.0, server 13.0.0)
Type "help" for help.

edb=#
```

ステップ 2: SQL/Protect のテーブル、関数、ビューは sqlprotect `` スキーマの下に構築されるため、 ``SET search_path `` コマンドを使用して検索パスに ``sqlprotect スキーマを含めます。これにより、SQL/Protect データベースオブジェクトを含む操作またはクエリをスキーマで修飾する必要がなくなります。

```
edb=# SET search_path TO sqlprotect;
SET
```

ステップ 3: 保護する各ロールは、保護されているロールのリストに追加する必要があります。このリストは、テーブル edb_sql_protect で維持されます。

ロールを追加するには、関数 protect_role('rolename') `` を使用します。次の例は、 ``appuser という名前のロールを保護します:

```
edb=# SELECT protect_role('appuser');
 protect_role
-----
(1 row)
```

次のクエリを発行することにより、保護された役割のリストに追加された役割を一覧表示できます。

```
edb=# SELECT * FROM edb_sql_protect;
 dbid  | roleid | protect_relations | allow_utility_cmds | allow_tautology |
allow_empty_dml
-----+-----+-----+-----+-----+-----
13917 | 16671 | t                  | f                   | f               |
(1 row)
```

オブジェクト識別番号 (OID) の代わりにオブジェクト名を使用して同じ情報を提供するビューも提供されます。

```
edb=# \x
Expanded display is on.
edb=# SELECT * FROM list_protected_users;
-[ RECORD 1 ]-----+-----
dbname          | edb
username        | appuser
protect_relations | t
allow_utility_cmds | f
allow_tautology   | f
allow_empty_dml   | f
```

2.2.1.2 保護レベルの設定

`edb_sql_protect.level` 構成パラメーターは保護レベルを設定し、保護されたロールが SQL ステートメントを発行したときの SQL/Protect の動作を定義します。定義された動作は、データベースサーバーで SQL/Protect を使用して構成されたすべてのデータベースの保護されたロールリスト内のすべてのロールに適用されます。

`edb_sql_protect.level` `` 構成パラメーター (```postgresql.conf` `` ファイル内) を次の値のいずれかに設定して、 ```learn` `` モード、 ```passive` `` モード、または ```active` のいずれかを使用できますモード:

- **学習** 保護されたロールのアクティビティを追跡し、ロールで使用される関係を記録します。これは、保護されたアプリケーションの予想される動作が学習されるように、最初に SQL/Protect を構成するときに使用されます。
- **passive** 保護されたロールが定義されたルールに違反している場合に警告を発行しますが、SQL ステートメントの実行を停止しません。これは、SQL/Protect が保護されたロールの予想される動作を学習した後の次のステップです。これは基本的に侵入検知モードで動作し、適切に監視されている場合は本番環境で実行できます。
- **active** 保護されたロールの無効なステートメントをすべて停止します。これは、危険なクエリの実行を防ぐ SQL ファイアウォールとして動作します。これは、攻撃者がアプリケーションの背後にある脆弱点とデータベースのタイプを特定しようとしている場合の早期侵入テストに対して特に効果的です。SQL/Protect はこれらの脆弱性ポイントを閉じるだけでなく、ブロックされたクエリを追跡するため、攻撃者がシステムに侵入する別の方法を見つける前に管理者に警告することができます。

`edb_sql_protect.level` `` パラメーターが設定されていないか、設定ファイルから省略されている場合、 ``SQL/Protect `` のデフォルトの動作は ```passive` です。

初めて SQL/Protect `` を使用する場合は、 ```edb_sql_protect.level` `` を ```learn` に設定します。

2.2.2 保護されたロールの監視

データベースで SQL/Protect を構成し、保護されたロールリストにロールを追加し、目的の保護レベルを設定したら、`learn` `` モード、`passive` `` モード、または `active` モード。その後、アプリケーションの実行を開始できます。

新しい SQL/Protect インストールでは、最初の手順は、保護されたロールが通常の操作中にアクセスを許可される関係を決定することです。学習モードでは、SQL/Protect がアクセスされる関係を記録している間、ロールがアプリケーションを実行できます。これらは、テーブル `edb_sql_protect_rel` `` に保存されているロールの `protected relations list` に追加されます。

SQL/Protect がパッシブモードまたはアクティブモードで実行されると、攻撃に対する保護の監視が開始されます。パッシブモードとアクティブモードでは、ロールは通常の使用時にロールがアクセスできる関係であると判断されたため、保護されたリレーションリストの関係にアクセスすることが許可されます。

ただし、ロールが保護された関係リストにない関係にアクセスしようとすると、SQL/Protect から `WARNING` `` または `ERROR` 重大度メッセージが返されます。関係に対するロールの試行されたアクションは、モードがパッシブかアクティブかによって実行される場合とされない場合があります。

2.2.2.1 学習モード

ステップ 1: 学習モードで SQL/Protect を有効にするには、以下に示すように `postgresql.conf` ファイルのパラメーターを設定します:

```
edb_sql_protect.enabled = on
edb_sql_protect.level = learn
```

ステップ 2: `postgresql.conf` ファイルをリロードします。

AdvancedServer アプリケーションメニューから `Expert Configuration` `` を選択し、次に `Reload Configuration` を選択します。

設定ファイルをリロードする別の方法については、`pg_reload_conf` `` 関数を使用します。スーパーユーザーとしてデータベースに接続していることを確認し、次の例に示すように `function pg_reload_conf` を実行します。

```
edb=# SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 row)
```

ステップ 3: 保護されたロールがアプリケーションを実行できるようにします。

例として、保護されたロール `appuser` `` によって `psql` アプリケーションで次のクエリが発行されます。

```

edb=> SELECT * FROM dept;
NOTICE:  SQLPROTECT: Learned relation: 16384
 deptno |   dtype   |   loc
-----+-----+-----
      10 | ACCOUNTING | NEW YORK
      20 | RESEARCH   | DALLAS
      30 | SALES       | CHICAGO
      40 | OPERATIONS | BOSTON
(4 rows)

edb=> SELECT empno, ename, job FROM emp WHERE deptno = 10;
NOTICE:  SQLPROTECT: Learned relation: 16391
 empno | ename  |   job
-----+-----+-----
   7782 | CLARK  | MANAGER
   7839 | KING   | PRESIDENT
   7934 | MILLER | CLERK
(3 rows)

```

SQL/Protect は、ロールの保護された関係リストに関係が追加されたことを示す NOTICE 重大度レベルのメッセージを生成します。

SQL/Protect 学習モードでは、疑いの原因となっている SQL ステートメントの実行は妨げられませんが、次の例に示すように、潜在的に危険なステートメントをユーザーに警告するメッセージが発行されます。

```

edb=> CREATE TABLE appuser_tab (f1 INTEGER);
NOTICE:  SQLPROTECT: This command type is illegal for this user
CREATE TABLE
edb=> DELETE FROM appuser_tab;
NOTICE:  SQLPROTECT: Learned relation: 16672
NOTICE:  SQLPROTECT: Illegal Query: empty DML
DELETE 0

```

ステップ 4: 保護されたロールがアプリケーションを実行するときに、ロールの保護されたリレーションリストへのリレーションの追加を監視するために、SQL/Protect テーブルを照会できます。

監視しているデータベースにスーパーユーザーとして接続し、検索パスに sqlprotect スキーマを含めるように設定します。

```

edb=# SET search_path TO sqlprotect;
SET

```

保護されたリレーションリストに追加されたリレーションを確認するには、edb_sql_protect_rel テーブルをクエリします。

```

edb=# SELECT * FROM edb_sql_protect_rel;
 dbid  | roleid | relid

```

(次のページに続く)

(前のページからの続き)

```

-----+-----+-----
13917 | 16671 | 16384
13917 | 16671 | 16391
13917 | 16671 | 16672
(3 rows)

```

list_protected_rels ビューは、OID の代わりにオブジェクト名とともにより包括的な情報を提供します:

```

edb=# SELECT * FROM list_protected_rels;
 Database | Protected User | Schema | Name      | Type  | Owner
-----+-----+-----+-----+-----+-----
 edb      | appuser       | public | dept      | Table | enterprisedb
 edb      | appuser       | public | emp       | Table | enterprisedb
 edb      | appuser       | public | appuser_tab | Table | appuser
(3 rows)

```

2.2.2.2 パッシブモード

役割のアプリケーションが必要なすべての関係にアクセスしたことを確認したら、保護レベルを変更して、SQL/Protect が着信 SQL クエリをアクティブに監視し、SQL インジェクション攻撃から保護できるようにすることができます。

パッシブモードは、パッシブモードとアクティブモードの2つの保護モードの制限が緩和されます。

ステップ 1: パッシブモードで SQL/Protect `` をアクティブにするには、以下に示すように ``postgresql.conf ファイルに以下のパラメーターを設定します。

```

edb_sql_protect.enabled = on
edb_sql_protect.level = passive

```

ステップ 2: 学習モード<learn_mode>`セクションの ``Step 2` に示されているように設定ファイルをリロードします。

これで、SQL/Protect はパッシブモードになりました。前の例の dept `` および ``emp `` テーブルのような学習されたりレーションの場合、次のクエリの実行で示されるように、 ``SQL/Protect `` によってクライアントへの特別な通知なしで SQL ステートメントが許可されますユーザー ``appuser`:

```

edb=> SELECT * FROM dept;
 deptno |  dname    |  loc
-----+-----+-----
    10 | ACCOUNTING | NEW YORK
    20 | RESEARCH  | DALLAS
    30 | SALES      | CHICAGO
    40 | OPERATIONS | BOSTON
(4 rows)

```

(次のページに続く)

(前のページからの続き)

```
edb=> SELECT empno, ename, job FROM emp WHERE deptno = 10;
 empno | ename  |      job
-----+-----+-----
   7782 | CLARK  |  MANAGER
   7839 | KING   | PRESIDENT
   7934 | MILLER |  CLERK
(3 rows)
```

SQL/Protect は、SQL ステートメントの実行を妨げませんが、次の例に示すように、学習されなかった関係に対して実行された SQL ステートメント、または禁止された署名を含む SQL ステートメントに対して、重大度 WARNING のメッセージを発行します:

```
edb=> CREATE TABLE appuser_tab_2 (f1 INTEGER);
WARNING: SQLPROTECT: This command type is illegal for this user
CREATE TABLE
edb=> INSERT INTO appuser_tab_2 VALUES (1);
WARNING: SQLPROTECT: Illegal Query: relations
INSERT 0 1
edb=> INSERT INTO appuser_tab_2 VALUES (2);
WARNING: SQLPROTECT: Illegal Query: relations
INSERT 0 1
edb=> SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
WARNING: SQLPROTECT: Illegal Query: relations
WARNING: SQLPROTECT: Illegal Query: tautology
f1
----
   1
   2
(2 rows)
```

ステップ 3: 疑わしいアクティビティの統計を監視します。

ビュー edb_sql_protect_stats `` をクエリすることにより、ロールの保護された関係リストにない関係、または SQL インジェクション攻撃シグネチャを含む SQL ステートメントが実行された回数を確認できます。ビュー ``edb_sql_protect_stats `` の詳細については、``Attack Attempt Statistics を参照してください。

以下は edb_sql_protect_stats に対するクエリです:

```
edb=# SET search_path TO sqlprotect;
SET
edb=# SELECT * FROM edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
  appuser  |          0 |          3 |          1 |          1 |          0
(1 row)
```

ステップ 4: 特定の攻撃に関する情報を表示します。

edb_sql_protect_queries ``ビューをクエリすることにより、ロールの保護された関係リストにない関係を参照した SQL ステートメント、または SQL インジェクション攻撃のシグネチャを含む SQL ステートメントを確認できます。ビュー ``edb_sql_protect_queries ``の詳細については、``Attack Attempt Queries を参照してください。

次のコードサンプルは、edb_sql_protect_queries に対するクエリを示しています。

```
edb=# SELECT * FROM edb_sql_protect_queries;
-[ RECORD 1 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:21:00 -04:00
query         | INSERT INTO appuser_tab_2 VALUES (1);
-[ RECORD 2 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:21:00 -04:00
query         | CREATE TABLE appuser_tab_2 (f1 INTEGER);
-[ RECORD 3 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:22:00 -04:00
query         | INSERT INTO appuser_tab_2 VALUES (2);
-[ RECORD 4 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:22:00 -04:00
query         | SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
```

注釈: 攻撃が Unix ドメインソケットを使用してデータベースサーバーと同じホストで発生した場合（つまり、pg_hba.conf ``接続タイプ ``local ``）。

2.2.2.3 アクティブモード

アクティブモードでは、許可されていない SQL ステートメントは実行できません。また、SQL/Protect によって発行されたメッセージの重大度レベルは、WARNING `` ではなく ``ERROR です。

ステップ 1: アクティブモードで SQL/Protect `` をアクティブにするには、以下に示すように ``postgresql.conf ファイルで以下のパラメーターを設定します。

```
edb_sql_protect.enabled = on
edb_sql_protect.level = active
```

ステップ 2: 学習モード<learn_mode>`セクションの ``Step 2`` に示されているように設定ファイルをリロードします。

次の例は、Passive Mode `` の ``Step 2`` の例に示されているものと似ていますが、``edb_sql_protect.level`` が ``active`` に設定されているときにユーザー ``appuser`` によって実行される SQL ステートメントを示しています。

```
edb=> CREATE TABLE appuser_tab_3 (f1 INTEGER);
ERROR: SQLPROTECT: This command type is illegal for this user
edb=> INSERT INTO appuser_tab_2 VALUES (1);
ERROR: SQLPROTECT: Illegal Query: relations
edb=> SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
ERROR: SQLPROTECT: Illegal Query: relations
```

結果の統計を以下に示します。

```
edb=# SELECT * FROM sqlprotect.edb_sql_protect_stats;
  username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
  appuser  |          0 |          5 |          2 |          1 |    0
(1 row)
```

以下は edb_sql_protect_queries に対するクエリです:

```
edb=# SELECT * FROM sqlprotect.edb_sql_protect_queries;
-[ RECORD 1 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:21:00 -04:00
query         | CREATE TABLE appuser_tab_2 (f1 INTEGER);
-[ RECORD 2 ]+-----
username      | appuser
ip_address    |
port          |
```

(次のページに続く)

(前のページからの続き)

```

machine_name |
date_time    | 20-JUN-14 13:22:00 -04:00
query        | INSERT INTO appuser_tab_2 VALUES (2);
-[ RECORD 3 ]+-----
username     | appuser
ip_address   | 192.168.2.6
port         | 50098
machine_name |
date_time    | 20-JUN-14 13:39:00 -04:00
query        | CREATE TABLE appuser_tab_3 (f1 INTEGER);
-[ RECORD 4 ]+-----
username     | appuser
ip_address   | 192.168.2.6
port         | 50098
machine_name |
date_time    | 20-JUN-14 13:39:00 -04:00
query        | INSERT INTO appuser_tab_2 VALUES (1);
-[ RECORD 5 ]+-----
username     | appuser
ip_address   | 192.168.2.6
port         | 50098
machine_name |
date_time    | 20-JUN-14 13:39:00 -04:00
query        | SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';

```

2.3 一般的なメンテナンス操作

以下に、他の一般的な操作の実行方法を説明します。

これらの操作を実行するには、スーパーユーザーとして接続し、検索パスに `sqlprotect` スキーマを含める必要があります。

2.3.1 保護された役割リストへの役割の追加

保護されたロールのリストにロールを追加するには、次の例に示すように `protect_role('rolename')` を実行します。

```
edb=# SELECT protect_role('newuser');
      protect_role
-----
(1 row)
```

2.3.2 保護された役割リストからの役割の削除

保護された役割のリストから役割を削除するには、次のいずれかの機能を使用します。

```
unprotect_role('rolename')
```

```
unprotect_role(roleoid)
```

保護されたロールのリストからロールを削除する前に `DROP ROLE ``` または ```DROP USER ``` の SQL ステートメントを使用してロールを削除する場合、```OID ``` を使用する関数のバリエーションが役立ちます。SQL/Protect リレーションのクエリがユーザー名に ```unknown (OID=16458) ``` などの値を返す場合、関数の ```unprotect_role(roleoid)` 形式を使用して、保護されたロールのリストから削除されたロールのエントリを削除します。

これらの機能を使用してロールを削除すると、ロールの保護された関係リストも削除されます。

削除されたロールの統計は、[drop_stats 関数](#) を使用するまで削除されません。

削除されたロールの問題のクエリは、[drop_queries 関数](#) を使用するまで削除されません。

以下は `unprotect_role` 関数の例です:

```
edb=# SELECT unprotect_role('newuser');
      unprotect_role
-----
(1 row)
```

または、OID に 16693 を指定することでロールを削除できます。

```
edb=# SELECT unprotect_role(16693);
 unprotect_role
-----
(1 row)
```

2.3.3 役割の保護のタイプを設定する

特定のタイプの SQL インジェクション攻撃からロールを保護するかどうかを変更できます。

ロールの保護を無効または有効にする SQL インジェクション攻撃のタイプに対応する `edb_sql_protect` の列のブール値を変更します。

`edb_sql_protect` `` を更新するステートメントの ``WHERE 句の次の列を必ず修飾してください:

- **dbid** 変更を行うデータベースの OID
- **roleid** ブール設定を変更するロールの OID

たとえば、特定のロールがユーティリティコマンドを発行できるようにするには、`allow_utility_cmds` 列を次のように更新します。

```
UPDATE edb_sql_protect SET allow_utility_cmds = TRUE WHERE dbid =
13917 AND roleid = 16671;
```

変更が行われたことを確認するには、`edb_sql_protect` `` または ``list_protected_users `` をクエリします。次のクエリでは、列 ``allow_utility_cmds `` に ``t が含まれていることに注意してください。

```
edb=# SELECT dbid, roleid, allow_utility_cmds FROM edb_sql_protect;
 dbid  | roleid | allow_utility_cmds
-----+-----+-----
 13917 | 16671  | t
(1 row)
```

更新されたルールは、変更が行われてからロールによって開始された新しいセッションで有効になります。

2.3.4 保護された関係リストからの関係の削除

SQL/Protect が特定のロールに特定のリレーションにアクセスできることを学習した場合、その後、そのリレーションをロールの保護されたリレーションリストから削除できます。

次のいずれかの関数を使用して、`edb_sql_protect_rel` テーブルからそのエントリを削除します。

```
unprotect_rel('rolename', 'relname') `` ``unprotect_rel('rolename',
'schema', 'relname') `` ``unprotect_rel(roleoid, reloid)
```

relname で指定されたリレーションが現在の検索パスにない場合、2 番目の関数形式を使用してリレーションのスキーマを指定します。

3 番目の関数形式では、テキスト名の代わりに、ロールとリレーションの OID をそれぞれ指定できます。

次の例は、ロール appuser `` の保護された関係リストから ``public.emp 関係を削除する方法を示しています。

```
edb=# SELECT unprotect_rel('appuser', 'public', 'emp');
 unprotect_rel
-----
(1 row)
```

次のクエリは、emp リレーションのエントリがなくなったことを示しています。

```
edb=# SELECT * FROM list_protected_rels;
 Database | Protected User | Schema | Name          | Type  | Owner
-----+-----+-----+-----+-----+-----
 edb      | appuser        | public | dept          | Table | enterprisedb
 edb      | appuser        | public | appuser_tab   | Table | appuser
(2 rows)
```

ロールがその関係を利用しようとするたびに、SQL/Protect は警告を発行するか、アクセスを完全にブロックします (edb_sql_protect.level の設定に依存)。

2.3.5 統計の削除

次の 2 つの機能のいずれかを使用して、ビュー edb_sql_protect_stats から統計を削除できます。

```
drop_stats('rolename')
```

```
drop_stats(roleoid)
```

OID を使用する関数のバリエーションは、drop_stats('rolename') `` を使用してロールの統計を削除する前に ``DROP ROLE `` または ``DROP USER `` の SQL ステートメントを使用してロールを削除する場合に役立ちます。 ``edb_sql_protect_stats `` に対するクエリがユーザー名に ``unknown (OID=16458) `` などの値を返す場合、関数の ``drop_stats(roleoid) `` 形式を使用して、 ``edb_sql_protect_stats から削除されたロールの統計を削除します。

以下は drop_stats 関数の例です:

```
edb=# SELECT drop_stats('appuser');
 drop_stats
-----
(1 row)
```

(次のページに続く)

(前のページからの続き)

```
edb=# SELECT * FROM edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
(0 rows)
```

以下は、統計を削除する前にロールが削除されたときに、関数の `drop_stats(roleoid)` 形式を使用する例です。

```
edb=# SELECT * FROM edb_sql_protect_stats;
 username          | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
 unknown (OID=16693) |          0 |          5 |          3 |          1 |          0
 appuser           |          0 |          5 |          2 |          1 |          0
(2 rows)

edb=# SELECT drop_stats(16693);
 drop_stats
-----
(1 row)

edb=# SELECT * FROM edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
 appuser  |          0 |          5 |          2 |          1 |          0
(1 row)
```

2.3.6 問題のあるクエリを削除する

次の 2 つの関数のいずれかを使用して、問題のクエリを view `edb_sql_protect_queries` から削除できます。

```
drop_queries('rolename')
```

```
drop_queries(roleoid)
```

OID を使用する関数のバリエーションは、`drop_queries('rolename')` `` を使用してロールの問題のあるクエリを削除する前に ```DROP ROLE``` または ```DROP USER``` の SQL ステートメントを使用してロールを削除する場合に役立ちます。 ```edb_sql_protect_queries``` のクエリがユーザー名に ```unknown (OID=16454)``` などの値を返す場合、関数の ```drop_queries(roleoid)``` 形式を使用して、削除されたロールの問題のあるクエリを ```edb_sql_protect_queries``` から削除します。

以下は `drop_queries` 関数の例です:

```
edb=# SELECT drop_queries('appuser');
drop_queries
```

```
-----
          5
(1 row)
```

```
edb=# SELECT * FROM edb_sql_protect_queries;
username | ip_address | port | machine_name | date_time | query
-----+-----+-----+-----+-----+-----
(0 rows)
```

以下は、ロールを削除してからクエリを削除するときに、関数の `drop_queries(roleoid)` 形式を使用する例です。

```
edb=# SELECT username, query FROM edb_sql_protect_queries;
username | query
-----+-----
unknown (OID=16454) | CREATE TABLE appuser_tab_2 (f1 INTEGER);
unknown (OID=16454) | INSERT INTO appuser_tab_2 VALUES (2);
unknown (OID=16454) | CREATE TABLE appuser_tab_3 (f1 INTEGER);
unknown (OID=16454) | INSERT INTO appuser_tab_2 VALUES (1);
unknown (OID=16454) | SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
(5 rows)
```

```
edb=# SELECT drop_queries(16454);
drop_queries
-----
          5
(1 row)
```

```
edb=# SELECT * FROM edb_sql_protect_queries;
username | ip_address | port | machine_name | date_time | query
-----+-----+-----+-----+-----+-----
(0 rows)
```

2.3.7 監視の無効化と有効化

SQL/Protect 監視をオフにする場合は、`postgresql.conf` `` ファイルを変更し、```edb_sql_protect.enabled` `` パラメーターを ```off` に設定します。ファイルを保存した後、サーバー構成を再ロードして設定を適用します。

SQL/Protect モニタリングを有効にする場合は、`postgresql.conf` `` ファイルを変更し、```edb_sql_protect.enabled` `` パラメーターを ```on` に設定します。ファイルを保存した後、サーバー構成を再ロードして設定を適用します。

2.4 SQL/Protect データベースのバックアップと復元

SQL/Protect で構成されたデータベースをバックアップし、バックアップファイルを新しいデータベースに復元するには、バックアップおよび復元手順に通常関連付けられていることをさらに考慮する必要があります。これは主に、このセクションで説明するように、SQL/Protect テーブルでオブジェクト識別番号（OID）を使用しているためです。

注釈: このセクションは、バックアップおよび復元手順により、pg_dump バックアッププログラムを使用する場合など、新しい OID で新しいデータベースにデータベースオブジェクトが再作成される場合に適用されます。

オペレーティングシステムのコピーユーティリティを使用して AdvancedServer データファイルのバイナリイメージを作成するだけで AdvancedServer データベースサーバーをバックアップする場合（ファイルシステムのバックアップ方法）、このセクションは適用されません。

2.4.1 SQL/Protect テーブルのオブジェクト識別番号

SQL/Protect は 2 つのテーブル（`edb_sql_protect` および `edb_sql_protect_rel`）を使用して、データベース、ロール、リレーションなどのデータベースオブジェクトに関する情報を保存します。これらのテーブル内のこれらのデータベースオブジェクトへの参照は、オブジェクトのテキスト名ではなく、オブジェクトの OID を使用して行われます。OID は、AdvancedServer が各データベースオブジェクトを一意に識別するために使用する数値データ型です。

データベースオブジェクトが作成されると、AdvancedServer はオブジェクトに OID を割り当てます。OID は、データベースカタログ内のオブジェクトへの参照が必要なときに使用されます。同じ CREATE TABLE ステートメントを持つテーブルなど、2 つのデータベースに同じデータベースオブジェクトを作成する場合、各テーブルには各データベースで異なる OID が割り当てられます。

バックアップされたデータベースオブジェクトが再作成されるバックアップおよび復元操作では、復元されたオブジェクトは、元のデータベースで割り当てられたものとは異なる OID が新しいデータベースで作成されます。その結果、`edb_sql_protect` および `edb_sql_protect_rel` テーブルに保存されたデータベース、ロール、およびリレーションを参照する OID は、これらのテーブルが単にバックアップファイルにダンプされてから新しいデータベースに復元されると無効になります。

以下のセクションでは、SQL/Protect テーブルの OID が正しいデータベースオブジェクトを参照するようにするために、SQL/Protect テーブルのバックアップと復元に特に使用される 2 つの関数、`export_sqlprotect` と `import_sqlprotect` について説明します。テーブルが復元されます。

2.4.2 データベースのバックアップ

次の手順は、SQL/Protect で構成されたデータベースをバックアップします。

ステップ 1: pg_dump を使用してバックアップファイルを作成します。

次の例は、pg_dump `` ユーティリティプログラムを使用してデータベース ``edb `` から作成された `` /tmp/edb.dmp という名前のプレーンテキストバックアップファイルを示しています。

```
$ cd /usr/edb/as13/bin
$ ./pg_dump -U enterprisedb -Fp -f /tmp/edb.dmp edb
Password:
$
```

ステップ 2: データベースにスーパーユーザーとして接続し、export_sqlprotect('sqlprotect_file') `` 関数を使用して SQL/Protect データをエクスポートします ( ``sqlprotect_file `` は ``SQL/Protect のファイルへの完全修飾パスです) `データが保存されます)。

enterprisedb `` オペレーティングシステムアカウント (AdvancedServer を PostgreSQL 互換モードでインストールした場合は ``1'') には、 ``sqlprotect_file で指定されたディレクトリへの読み取りおよび書き込みアクセスが必要です。

```
edb=# SELECT sqlprotect.export_sqlprotect('/tmp/sqlprotect.dmp');
      export_sqlprotect
-----
(1 row)
```

ファイル /tmp/edb.dmp `` および `` /tmp/sqlprotect.dmp は、データベース全体のバックアップを構成します。

2.4.3 バックアップファイルからの復元

ステップ 1: バックアップファイルを新しいデータベースに復元します。

次の例では、psql `` ユーティリティプログラムを使用して、プレーンテキストバックアップファイル `` /tmp/edb.dmp `` を ``newdb という名前の新しく作成されたデータベースに復元します。

```
$ /usr/edb/as13/bin/psql -d newdb -U enterprisedb -f /tmp/edb.dmp
Password for user enterprisedb:
SET
SET
SET
SET
SET
SET
COMMENT
```

(次のページに続く)

(前のページからの続き)

```
CREATE SCHEMA
.
.
.
```

ステップ 2: 新しいデータベースにスーパーユーザーとして接続し、edb_sql_protect_rel テーブルからすべての行を削除します。

このステップは、元のデータベースからバックアップされた edb_sql_protect_rel テーブル内の既存の行を削除します。これらの行には、バックアップファイルが復元されたデータベースに関連する正しい OID が含まれていません。

```
$ /usr/edb/as13/bin/psql -d newdb -U enterprisedb
Password for user enterprisedb:
psql.bin (13.0.0, server 13.0.0)
Type "help" for help.

newdb=# DELETE FROM sqlprotect.edb_sql_protect_rel;
DELETE 2
```

ステップ 3: edb_sql_protect テーブルからすべての行を削除します。

このステップは、元のデータベースからバックアップされた edb_sql_protect テーブル内の既存の行を削除します。これらの行には、バックアップファイルが復元されたデータベースに関連する正しい OID が含まれていません。

```
newdb=# DELETE FROM sqlprotect.edb_sql_protect;
DELETE 1
```

ステップ 4: データベースに存在する可能性のある統計を削除します。

この手順により、バックアップの復元先のデータベースに存在する可能性のある既存の統計がすべて削除されます。次のクエリは、既存の統計を表示します。

```
newdb=# SELECT * FROM sqlprotect.edb_sql_protect_stats;
  username  | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
(0 rows)
```

上記のクエリに表示される各行に対して、エントリのロール名を指定する drop_stats 関数を使用します。

たとえば、username `` 列に ``appuser が含まれる行がある場合、次のコマンドを発行して削除します。

```
newdb=# SELECT sqlprotect.drop_stats('appuser');
drop_stats
```

(次のページに続く)

(前のページからの続き)

```
-----
(1 row)
```

ステップ 5: データベースに存在する問題のあるクエリを削除します。

この手順により、バックアップの復元先のデータベースに存在する可能性のある既存のクエリが削除されます。次のクエリは、既存のクエリを表示します。

```
edb=# SELECT * FROM sqlprotect.edb_sql_protect_queries;
 username | ip_address | port | machine_name | date_time | query
-----+-----+-----+-----+-----+-----
(0 rows)
```

上記のクエリに表示される各行に対して、エントリのロール名を指定する `drop_queries` 関数を使用します。

たとえば、`username` 列に `appuser` が含まれる行がある場合、次のコマンドを発行して削除します。

```
edb=# SELECT sqlprotect.drop_queries('appuser');
 drop_queries
-----
(1 row)
```

手順 6: 元のデータベースで SQL/Protect によって保護されたロール名が、新しいデータベースが存在するデータベースサーバーに存在することを確認します。

元のデータベースと新しいデータベースが同じデータベースサーバーにある場合、これらのロールをデータベースサーバーから削除していないと仮定して、何もする必要はありません。

ステップ 7: ファンクション `import_sqlprotect('sqlprotect_file')` を実行します。`sqlprotect_file` は `Backing Up the Database` の Step 2 で作成したファイルへの完全修飾パスです。

```
newdb=# SELECT sqlprotect.import_sqlprotect('/tmp/sqlprotect.dmp');
 import_sqlprotect
-----
(1 row)
```

テーブル `edb_sql_protect` および `edb_sql_protect_rel` には、新しいデータベースで割り当てられたデータベースオブジェクトの OID を含むエントリが入力されます。統計ビュー `edb_sql_protect_stats` には、元のデータベースからインポートされた統計も表示されるようになりました。

これで、このデータベースの SQL/Protect テーブルと統計が適切に復元されます。これは、AdvancedServer システムカタログに対する次のクエリによって検証されます。

```

newdb=# SELECT datname, oid FROM pg_database;
 datname | oid
-----+-----
 template1 |      1
 template0 |   13909
   edb     |   13917
   newdb   |   16679
(4 rows)

newdb=# SELECT rolname, oid FROM pg_roles;
 rolname | oid
-----+-----
 enterprisedb |      10
   appuser    |   16671
   newuser    |   16678
(3 rows)

newdb=# SELECT relname, oid FROM pg_class WHERE relname IN
('dept','emp','appuser_tab');
 relname | oid
-----+-----
 appuser_tab | 16803
   dept     | 16809
   emp      | 16812
(3 rows)

newdb=# SELECT * FROM sqlprotect.edb_sql_protect;
 dbid | roleid | protect_relations | allow_utility_cmds | allow_tautology |
 allow_empty_dml
-----+-----+-----+-----+-----+-----
 16679 | 16671 | t                  | t                  | f                |
(1 row)

newdb=# SELECT * FROM sqlprotect.edb_sql_protect_rel;
 dbid | roleid | relid
-----+-----+-----
 16679 | 16671 | 16809
 16679 | 16671 | 16803
(2 rows)

newdb=# SELECT * FROM sqlprotect.edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
 appuser  |           0 |           5 |           2 |           1 | 0
(1 row)

newdb=# \x

```

(次のページに続く)

(前のページからの続き)

```

Expanded display is on.
newdb=# SELECT * FROM sqlprotect.edb_sql_protect_queries;
-[ RECORD 1 ]+-----
username      | appuser
ip_address     |
port          |
machine_name   |
date_time      | 20-JUN-14 13:21:00 -04:00
query         | CREATE TABLE appuser_tab_2 (f1 INTEGER);
-[ RECORD 2 ]+-----
username      | appuser
ip_address     |
port          |
machine_name   |
date_time      | 20-JUN-14 13:22:00 -04:00
query         | INSERT INTO appuser_tab_2 VALUES (2);
-[ RECORD 3 ]+-----
username      | appuser
ip_address     | 192.168.2.6
port          | 50098
machine_name   |
date_time      | 20-JUN-14 13:39:00 -04:00
query         | CREATE TABLE appuser_tab_3 (f1 INTEGER);
-[ RECORD 4 ]+-----
username      | appuser
ip_address     | 192.168.2.6
port          | 50098
machine_name   |
date_time      | 20-JUN-14 13:39:00 -04:00
query         | INSERT INTO appuser_tab_2 VALUES (1);
-[ RECORD 5 ]+-----
username      | appuser
ip_address     | 192.168.2.6
port          | 50098
machine_name   |
date_time      | 20-JUN-14 13:39:00 -04:00
query         | SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';

```

テーブル `edb_sql_protect` `` および `` `edb_sql_protect_rel` の列については、次のことに注意してください。

- **dbid** `pg_database` `` から `` `newdb` `` の `` `oid` 列の値と一致します
- **roleid** `appuser` `` の `` `pg_roles` `` から `` `oid` 列の値と一致します

また、テーブル `edb_sql_protect_rel` `` では、リレーション `` `dept` `` および `` 5 `` の `` `relid` `` 列の値が `` `pg_class` `` の `` `oid` 列の値と一致することに注意してください。`。

ステップ 8: 新しいデータベースを実行しているデータベースサーバーの `postgresql.conf` ファイルで、SQL/Protect 構成パラメーターが希望どおりに設定されていることを確認します。必要に応じて、データベースサーバーを再起動するか、構成ファイルを再読み込みします。

これで、SQL/Protect を使用してデータベースを監視できます。

第 3 章

仮想プライベートデータベース

Virtual Private Database `` は、セキュリティポリシーを使用したきめの細かいアクセス制御の一種です。 ``Fine-grained access control は、データへのアクセスを、セキュリティポリシーで定義された特定の行まで制御できることを意味します。

セキュリティポリシーをエンコードするルールは、*policyfunction**で定義されます。これは、特定の入力パラメーターと戻り値を持つ *SPL* 関数です。*セキュリティポリシー は、ポリシー関数と特定のデータベースオブジェクト（通常はテーブル）の名前付き関連付けです。

AdvancedServer では、SPL に加えて、SQL や PL/pgSQL など、AdvancedServer がサポートする任意の言語でポリシー関数を作成できます。

注釈：現在 AdvancedServerVirtualPrivateDatabase でサポートされているデータベースオブジェクトはテーブルです。ポリシーをビューまたはシノニムに適用することはできません。

仮想プライベートデータベースを使用する利点は次のとおりです。

- きめ細かいレベルのセキュリティを提供します。 GRANT コマンドによって与えられるデータベースオブジェクトレベルの特権は、データベースオブジェクトのインスタンス全体へのアクセス特権を決定します。一方、仮想プライベートデータベースは、データベースオブジェクトインスタンスの個々の行に対するアクセス制御を提供します。
- SQL コマンドのタイプ（ INSERT, UPDATE, DELETE, or SELECT ）に応じて、異なるセキュリティポリシーを適用できます。
- セキュリティポリシーは、データベースオブジェクトにアクセスするアプリケーションのセッションユーザーなどの要因に応じて、データベースオブジェクトに影響を与える該当する SQL コマンドごとに動的に変化します。
- セキュリティポリシーの呼び出しは、データベースオブジェクトにアクセスするすべてのアプリケーションに対して透過的であるため、セキュリティポリシーを適用するために個々のアプリケーションを変更する必

要はありません。

- セキュリティポリシーが有効になると、アプリケーション（新しいアプリケーションを含む）がセキュリティポリシーを回避することはできません。ただし、以下に記載されているシステム権限が必要です。
- スーパーユーザーであっても、以下に記載されているシステム権限を除き、セキュリティポリシーを回避することはできません。

注釈: セキュリティポリシーを回避できる唯一の方法は、ユーザーに `EXEMPT ACCESS POLICY` `` システム権限が付与されている場合です。この特権を持つユーザーはデータベース内のすべてのポリシーから免除されるため、`` `EXEMPT ACCESS POLICY` 特権は細心の注意を払って付与する必要があります。

DBMS_RLS パッケージは、ポリシーの作成、ポリシーの削除、ポリシーの有効化、およびポリシーの無効化の手順を提供します。

第 4 章

sslutils

sslutils `` は、EDBPostgresEnterpriseManager サーバーが使用する SSL 証明書生成機能を AdvancedServer に提供する Postgres 拡張機能です。sslutils は ``edb-asxx-server-sslutils `` RPM パッケージを使用してインストールされます。ここで、 ``xx は AdvancedServer のバージョン番号です。

sslutils パッケージは、次のセクションに示す機能を提供します。

これらのセクションでは、関数のパラメーターリストの各パラメーターは、Parameters `` サブセクションの下の ``parameter n で記述されます。、3 番目など）関数のパラメーターリスト内。

4.1 openssl_rsa_generate_key

openssl_rsa_generate_key 関数は RSA 秘密鍵を生成します。関数のシグネチャは次のとおりです。

```
openssl_rsa_generate_key(<integer>) RETURNS <text>
```

関数を呼び出すときに、ビット数を整数値として渡します。関数は生成されたキーを返します。

4.2 openssl_rsa_key_to_csr

openssl_rsa_key_to_csr 関数は証明書署名要求（CSR）を生成します。署名は次のとおりです。

```
openssl_rsa_key_to_csr(<text>, <text>, <text>, <text>, <text>, <text>,
<text>) RETURNS text
```

この関数は、証明書署名要求を生成して返します。

パラメータ

parameter 1

RSA キーファイルの名前。

parameter 2

署名要求を使用するエージェントの共通名（例: agentN）。

parameter 3

サーバーが存在する国の名前。

parameter 4

サーバーが存在する州の名前。

parameter 5

サーバーが存在する州内の場所（都市）。

parameter 6

証明書を要求している組織単位の名前。

parameter 7

証明書を要求しているユーザーのメールアドレス。

4.3 openssl_csr_to_crt

openssl_csr_to_crt 関数は、自己署名証明書または認証局証明書を生成します。署名は次のとおりです。

```
openssl_csr_to_crt(<text>, <text>, <text>) RETURNS <text>
```

この関数は、自己署名証明書または認証局証明書を返します。

パラメータ

parameter 1

要求に署名する証明書の名前。

parameter 2

認証局証明書へのパス、または認証局証明書を生成する場合は NULL。

parameter 3

認証局の秘密鍵へのパス、または（引数 2 が NULL の場合）秘密鍵へのパス。

4.4 openssl_rsa_generate_crl

openssl_rsa_generate_crl 関数はデフォルトの証明書失効リストを生成します。署名は次のとおりです。

```
openssl_rsa_generate_crl(<text>, <text>) RETURNS <text>
```

この関数は、証明書失効リストを返します。

パラメータ

parameter 1

認証局証明書へのパス。

parameter 2

認証局の秘密鍵へのパス。

第 5 章

データ編集

データ編集 は、特定のユーザーに表示されるデータを動的に変更することにより、機密データの露出を制限します。

たとえば、社会保障番号 (SSN) は 021-23-9567 `` として保存されます。特権ユーザーは完全な SSN を見ることができますが、他のユーザーは最後の 4 桁の ``xxx-xx-9567 しか見ることができません。

データ改訂は、改訂が適用される各フィールドに関数を定義することにより実装されます。この関数は、データ修正の対象となるユーザーに表示される値を返します。

したがって、たとえば、SSN フィールドの場合、リダクション関数は 021-23-9567 `` の入力 SSN に対して ``xxx-xx-9567 を返します。

給与フィールドの場合、リダクション関数は、入力給与値に関係なく、常に \$0.00 を返します。

これらの関数は、CREATE REDACTION POLICY コマンドを使用して、redaction ポリシーに組み込まれます。このコマンドは、ポリシーが適用されるテーブル、指定されたリダクション関数の影響を受けるテーブル列、影響を受けるセッションユーザーを決定する式、およびその他のオプションを指定します。

postgresql.conf `` ファイルの ``edb_data_redaction パラメーターは、データ編集を適用するかどうかを決定します。

デフォルトでは、パラメーターは有効になっているため、redaction ポリシーが有効になり、次のことが行われます。

- スーパーユーザーとテーブル所有者は、データ編集をバイパスして、元のデータを表示します。
- 他のすべてのユーザーは、redaction ポリシーが適用され、再フォーマットされたデータが表示されます。

セッション中にパラメーターを FALSE に設定して無効にした場合、次のようになります。

- スーパーユーザーとテーブル所有者は、データ編集をバイパスして、元のデータを表示します。
- 他のすべてのユーザーが取得すると、エラーが発生します。

redaction ポリシーは、ALTER REDACTION POLICY `` コマンドを使用して変更するか、``DROP REDACTION POLICY コマンドを使用して削除できます。

リダクションポリシーコマンドについては、以降のセクションで詳しく説明します。

5.1 編集ポリシーの作成

CREATE REDACTION POLICY は、テーブルの新しいデータ編集ポリシーを定義します。

概要

```
CREATE REDACTION POLICY <name> ON <table_name>
[ FOR ( <expression> ) ]
[ ADD [ COLUMN ] <column_name> USING <funcname_clause>
  [ WITH OPTIONS ( [ <redaction_option> ]
    [, <redaction_option> ] )
  ]
] [, ...]
```

ここで、redaction_option は次のとおりです。

```
{ SCOPE <scope_value> |
  EXCEPTION <exception_value> }
```

説明

CREATE REDACTION POLICY `` コマンドは、リダクション機能を使用して列データを編集することにより、テーブルの新しい列レベルのセキュリティポリシーを定義します。新しく作成されたデータ修正ポリシーは、デフォルトで有効になります。ポリシーは ``ALTER REDACTION POLICY ... DISABLE を使用して無効にできます。

FOR (expression)

このフォームは、redaction ポリシー式を追加します。

ADD [COLUMN]

このオプションのフォームは、テーブルの列をデータ改訂ポリシーに追加します。USING `` は redaction 関数式を指定します。作成中のデータ編集ポリシーにテーブルの複数の列を追加する場合は、複数の ``ADD [COLUMN] `` フォームを使用できます。オプションの ``WITH OPTIONS (...) `` 節は、適用されるデータ編集ポリシーのスコープまたは例外を指定します。スコープや例外が指定されていない場合、スコープと例外のデフォルト値はそれぞれ ``query `` と ``none になります。

パラメータ

name

作成するデータ改訂ポリシーの名前。これは、テーブルの他の既存のデータ改訂ポリシーの名前とは異なる必要があります。

table_name

データ修正ポリシーが適用されるテーブルの名前（オプションでスキーマ修飾）。

expression

データ改訂ポリシー式。この式が `false` と評価された場合、編集は適用されません。

column_name

データ修正ポリシーが作成されるテーブルの既存の列の名前。

funcname_clause

改訂された列の値を計算する方法を決定するデータ改訂関数。リダクション関数の戻り値の型は、データリダクションポリシーが追加される列の型と同じである必要があります。

scope_value

スコープは、列に編集が適用されるクエリパーツを識別しました。スコープ値は `query`、`top_tlist`、または `top_tlist_or_error` です。スコープが `query` の場合、クエリのどこに表示されているかに関係なく、列に編集が適用されます。スコープが `top_tlist` の場合、列の編集はクエリのトップターゲットリストに表示されたときにのみ適用されます。スコープが `top_tlist_or_error` の場合、振る舞いは `top_tlist` と同じになりますが、クエリ内のどこかに列が現れるとエラーがスローされます。

exception_value

例外は、リダクションが免除されるクエリ部分を識別しました。例外値は `none`、`equal`、または `leakproof` です。例外が `none` の場合、免除はありません。例外が `equal` の場合、列は等価性テストで使用されるときに編集されません。例外が `leakproof` の場合、漏れ防止機能が適用されても列は編集されません。

注:

データリダクションポリシーを作成または変更するには、テーブルの所有者である必要があります。

スーパーユーザーとテーブル所有者は、データ改訂ポリシーから免除されます。

例

以下は、この機能を実稼働環境で使用方法の例です。employees テーブルにデータ編集ポリシーのコンポーネントを作成します。

```
CREATE TABLE employees (  
  id          integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  
  name        varchar(40) NOT NULL,  
  ssn         varchar(11) NOT NULL,  
  phone       varchar(10),  
  birthday    date,  
  salary      money,
```

(次のページに続く)

(前のページからの続き)

```

email          varchar(100)
);

-- Insert some data
INSERT INTO employees (name, ssn, phone, birthday, salary, email)
VALUES
( 'Sally Sample', '020-78-9345', '5081234567', '1961-02-02', 51234.34,
'sally.sample@enterprisedb.com'),
( 'Jane Doe', '123-33-9345', '6171234567', '1963-02-14', 62500.00,
'jane.doe@gmail.com'),
( 'Bill Foo', '123-89-9345', '9781234567', '1963-02-14', 45350,
'william.foe@hotmail.com');

-- Create a user hr who can see all the data in employees
CREATE USER hr;
-- Create a normal user
CREATE USER alice;
GRANT ALL ON employees TO hr, alice;

-- Create redaction function in which actual redaction logic resides
CREATE OR REPLACE FUNCTION redact_ssn (ssn varchar(11)) RETURN varchar(11) IS
BEGIN
    /* replaces 020-12-9876 with xxx-xx-9876 */
    return overlay (ssn placing 'xxx-xx' from 1) ;
END;

CREATE OR REPLACE FUNCTION redact_salary () RETURN money IS BEGIN return
0::money;
END;
```

次に、employees `` にデータのリダクションポリシーを作成して、 ``ssn `` の列をリダクションします。これは、同等の条件でアクセス可能であり、 ``salary `` はデフォルトのスコープと例外でアクセスできます。リダクションポリシーは、 ``hr ユーザーに対しては免除されます。

```

CREATE REDACTION POLICY redact_policy_personal_info ON employees FOR
(session_user != 'hr')
ADD COLUMN ssn USING redact_ssn(ssn) WITH OPTIONS (SCOPE query,
EXCEPTION equal),
ADD COLUMN salary USING redact_salary();
```

hr ユーザーの可視データは次のようになります。

```

-- hr can view all columns data
edb=# \c edb hr
edb=> SELECT * FROM employees;
 id | name          | ssn          | phone          | birthday          |
salary          | email
```

(次のページに続く)

(前のページからの続き)

```

-----+-----+-----+-----+-----+-----+-----
--+-----
1 | Sally Sample | 020-78-9345 | 5081234567 | 02-FEB-61 00:00:00 |
$51,234.34 | sally.sample@enterprisedb.com
2 | Jane Doe      | 123-33-9345 | 6171234567 | 14-FEB-63 00:00:00 |
$62,500.00 | jane.doe@gmail.com
3 | Bill Foo      | 123-89-9345 | 9781234567 | 14-FEB-63 00:00:00 |
$45,350.00 | william.foe@hotmail.com
(3 rows)

```

通常のユーザー alice の可視データは次のようになります。

```

-- Normal user cannot see salary and ssn number.
edb=> \c edb alice
edb=> SELECT * FROM employees;
id | name          | ssn          | phone          | birthday          | salary |
email
-----+-----+-----+-----+-----+-----+-----
1 | Sally Sample | xxx-xx-9345 | 5081234567 | 02-FEB-61 00:00:00 | $0.00 |
sally.sample@enterprisedb.com
2 | Jane Doe     | xxx-xx-9345 | 6171234567 | 14-FEB-63 00:00:00 | $0.00 |
jane.doe@gmail.com
3 | Bill Foo     | xxx-xx-9345 | 9781234567 | 14-FEB-63 00:00:00 | $0.00 |
william.foe@hotmail.com
(3 rows)

```

ただし、exception_value `` の設定により、等値チェックに使用する場合、 ``ssn のデータにアクセスできます。

```

-- Get ssn number starting from 123
edb=> SELECT * FROM employees WHERE substring(ssn from 0 for 4) = '123';
id | name          | ssn          | phone          | birthday          | salary |
email
-----+-----+-----+-----+-----+-----+-----
2 | Jane Doe     | xxx-xx-9345 | 6171234567 | 14-FEB-63 00:00:00 | $0.00 |
jane.doe@gmail.com
3 | Bill Foo     | xxx-xx-9345 | 9781234567 | 14-FEB-63 00:00:00 | $0.00 |
william.foe@hotmail.com
(2 rows)

```

警告

1. 継承階層で作成されたデータ改訂ポリシーはカスケードされません。たとえば、データリダクションポリシーが親に対して作成される場合、それを継承する子テーブルには適用されません。子テーブルはそれを継承します。これらの子テーブルにアクセスできる人は、編集されていないデータを見ることができます。継

承階層の詳細については、次の場所にある PostgreSQLCoreDocumentation の Inheritance を参照してください。

<https://www.postgresql.org/docs/current/static/ddl-inherit.html>

2. スーパーユーザーまたはテーブルの所有者がテーブルにマテリアライズドビューを作成し、テーブルおよびマテリアライズドビューのアクセス権 GRANT SELECT を非スーパーユーザーに提供している場合、非スーパーユーザーはマテリアライズドビューを介した編集されていないデータ。
3. リダクション関数の本体でアクセスされるオブジェクトは、スキーマで修飾されている必要があります。そうでない場合、pg_dump が失敗する可能性があります。

互換性

CREATE REDACTION POLICY は EDB 拡張機能です。

参照

ALTER REDACTION POLICY, DROP REDACTION POLICY

5.2 ALTERREDACTIONPOLICY

ALTER REDACTION POLICY はテーブルのデータ改訂ポリシーの定義を変更します。

概要

```
ALTER REDACTION POLICY <name> ON <table_name> RENAME TO <new_name>

ALTER REDACTION POLICY <name> ON <table_name> FOR ( <expression> )

ALTER REDACTION POLICY <name> ON <table_name> { ENABLE | DISABLE}

ALTER REDACTION POLICY <name> ON <table_name>
  ADD [ COLUMN ] <column_name> USING <funcname_clause>
    [ WITH OPTIONS ( [ <redaction_option> ]
      [, <redaction_option> ] )
    ]

ALTER REDACTION POLICY <name> ON <table_name>
  MODIFY [ COLUMN ] <column_name>
  {
    [ USING <funcname_clause> ]
  |
    [ WITH OPTIONS ( [ <redaction_option> ]
      [, <redaction_option> ] )
    ]
  }

ALTER REDACTION POLICY <name> ON <table_name>
  DROP [ COLUMN ] <column_name>
```

ここで、redaction_option は次のとおりです。

```
{ SCOPE <scope_value> |
  EXCEPTION <exception_value> }
```

説明

ALTER REDACTION POLICY は既存のデータ改訂ポリシーの定義を変更します。

ALTER REDACTION POLICY を使用するには、データ改訂ポリシーが適用されるテーブルを所有する必要があります。

FOR (expression)

このフォームは、データ編集ポリシー式を追加または置換します。

ENABLE

以前に無効化されたテーブルのデータ修正ポリシーを有効にします。

DISABLE

テーブルのデータ改訂ポリシーを無効にします。

ADD [COLUMN]

このフォームは、テーブルの列を既存の **redaction** ポリシーに追加します。詳細については、**CREATE REDACTION POLICY** を参照してください。

MODIFY [COLUMN]

このフォームは、テーブルの列のデータ編集ポリシーを変更します。列の **redaction** 関数句や **redaction** オプションを更新できます。USING `` 節は更新するリダクション関数式を指定し、``WITH OPTIONS (...) `` 節はスコープや例外を指定します。リダクション関数句、リダクションスコープ、リダクション例外の詳細については、``CREATE REDACTION POLICY を参照してください。

DROP [COLUMN]

このフォームは、テーブルの列をデータ改訂ポリシーから削除します。

パラメータ

name

変更する既存のデータ改訂ポリシーの名前。

table_name

データ改訂ポリシーが有効になっているテーブルの名前（オプションでスキーマ修飾）。

new_name

データ改訂ポリシーの新しい名前。これは、テーブルの他の既存のデータ改訂ポリシーの名前とは異なる必要があります。

expression

データ改訂ポリシー式。

column_name

データ編集ポリシーが変更またはドロップされるテーブルの既存の列の名前。

funcname_clause

列のデータ修正関数式。詳細については、**CREATE REDACTION POLICY** を参照してください。

scope_value

スコープは、列に編集が適用されるクエリパーツを識別しました。詳細については、CREATE REDACTION POLICY を参照してください。

exception_value

例外は、リダクションが免除されるクエリ部分を識別しました。詳細については、CREATE REDACTION POLICY を参照してください。

例

employees `` という名前のテーブルの ``redact_policy_personal_info というデータ改訂ポリシーを更新します。

```
ALTER REDACTION POLICY redact_policy_personal_info ON employees
FOR (session_user != 'hr' AND session_user != 'manager');
```

そして、同じポリシーの列 ssn のデータ編集機能を更新するには:

```
ALTER REDACTION POLICY redact_policy_personal_info ON employees
MODIFY COLUMN ssn USING redact_ssn_new(ssn);
```

互換性

ALTER REDACTION POLICY は EDB 拡張機能です。

参照

CREATE REDACTION POLICY, DROP REDACTION POLICY

5.3 ドロップ修正ポリシー

DROP REDACTION POLICY はテーブルからデータ編集ポリシーを削除します。

概要

```
DROP REDACTION POLICY [ IF EXISTS ] <name> ON <table_name>
[ CASCADE | RESTRICT ]
```

説明

DROP REDACTION POLICY は、指定されたデータ改訂ポリシーをテーブルから削除します。

DROP REDACTION POLICY を使用するには、redaction ポリシーが適用されるテーブルを所有する必要があります。

パラメータ

IF EXISTS

データ編集ポリシーが存在しない場合でもエラーをスローしないでください。この場合、通知が発行されます。

name

削除するデータ改訂ポリシーの名前。

table_name

データ改訂ポリシーが有効になっているテーブルの名前（オプションでスキーマ修飾）。

CASCADE

RESTRICT

これらのキーワードは、データ編集ポリシーに依存関係がないため、効果がありません。

例

employees `` という名前のテーブルに ``redact_policy_personal_info というデータ編集ポリシーをドロップするには:

```
DROP REDACTION POLICY redact_policy_personal_info ON employees;
```

互換性

DROP REDACTION POLICY は EDB 拡張機能です。

参照

CREATE REDACTION POLICY, ALTER REDACTION POLICY

5.4 システムカタログ

このセクションでは、redaction ポリシー情報を保存するシステムカタログについて説明します。

5.4.1 edb_redaction_column

edb_redaction_column システムカタログには、テーブルの列に添付されたデータ改訂ポリシーに関する情報が格納されます。

コラム	種類	参照資料	説明
oid	oid		行識別子（非表示属性、明示的に選択する必要があります）
rdpolicyid	oid	edb_redaction_policy.oid	データ改訂ポリシーは、説明されている列に適用されます
rdrelid	oid	pg_class.oid	説明された列が属するテーブル
rdattnum	int2	pg_attribute.attnum	記載されている列の番号
rdscope	int2		リダクションスコープ:1=クエリ、2=top_tlist、4=top_tlist_or_error
rdexception	int2		リダクション例外:8=なし、16=等しい、32=漏れ防止
rdfuncexpr	pg_node_tree		データ修正関数式

注釈: 表の redaction ポリシー edb_redaction_column.rdpolicyid `` が有効で、redaction ポリシー式 ``edb_redaction_policy.rdexpr `` が ``true に評価される場合、説明されている列は redaction されます。

5.4.2 edb_redaction_policy

カタログ edb_redaction_policy には、テーブルの redaction ポリシーに関する情報が格納されます。

コラム	種類	参照資料	説明
oid	oid		行識別子（非表示属性、明示的に選択する必要があります）
rdname	お名前		データ編集ポリシーの名前
rdrelid	oid	pg_class.oid	データ編集ポリシーが適用されるテーブル
rdenable	ブール値		データ編集ポリシーは有効になっていますか？
rdexpr	pg_node_tree		データ改訂ポリシー式

注釈：データ編集ポリシーは、テーブルが有効で、式が真と評価された場合にテーブルに適用されます。

第 6 章

結論

EDBPostgresTMAdvancedServer セキュリティ機能ガイド

著作権 ©2007-2020EnterpriseDBCorporation。

無断複写・転載を禁じます。

EnterpriseDB®Corporation

34CrosbyDrive、Suite201、Bedford、MA01730、アメリカ

T+17813573390F+19784671307E

info@enterprisedb.com

www.enterprisedb.com

- EnterpriseDB および PostgresEnterpriseManager は、EnterpriseDBCorporation の登録商標です。EDB および EDBPostgres は、EnterpriseDBCorporation の商標です。Oracle は、Oracle、Inc の登録商標です。その他の商標は、それぞれの所有者の商標です。
- EDB は、存在する場合、EDB 製品のログオン UI の設計、コーディングのベストプラクティスの確立、レビュー、および入力検証の検証を行います。EDB は追加の入力コンポーネントに対して同じアプローチに従いますが、製品の性質上、限定された検証が可能な信頼できるユーザーによって入力および送信されるフリーフォーム SQL、WMI またはその他の文字列を受け入れる必要がある場合があります。このような場合、ユーザーが誤った入力や危険な入力を禁止することはできません。
- EDB は、将来、認証された信頼できるユーザーからのフリーフォーム SQL、WMI またはその他の潜在的に危険な入力を受け入れる製品に機能を追加する権利を留保しますが、そのような機能はすべて、可能な限り最小のリスクを提供するように設計およびテストされます可能であれば、スーパーユーザーまたは同等の特権が必要です。
- EDB は、すべての潜在的な脅威を予測できることを保証しないため、すべての潜在的な脆弱性が対処または検討されたことを完全に保証することはできません。

索引

Active Mode, 17
 ALTER REDACTION POLICY, 46
 Attack Attempt Queries, 7
 Attack Attempt Statistics, 6

 Backing Up the Database, 26

 Common Maintenance Operations, 20
 Configuring SQL/Protect, 8
 CREATE REDACTION POLICY, 41

 Data Redaction, 37
 Deleting Offending Queries, 23
 Deleting Statistics, 22
 Disabling and Enabling Monitoring, 24
 DROP REDACTION POLICY, 49

 edb_redaction_column, 50
 edb_redaction_policy, 50

 Monitoring Protected Roles, 12
 Monitoring SQL Injection Attacks, 6

 Object Identification Numbers in SQL/Protect
 Tables, 25
 openssl_csr_to_crt, 36
 openssl_rsa_generate_crl, 36
 openssl_rsa_generate_key, 35
 openssl_rsa_key_to_csr, 35

 Passive Mode, 15
 Protecting Against SQL Injection Attacks, 1

 Removing a Relation From the Protected
 Relations List, 21
 Removing a Role From the Protected Roles List,
 20
 Restoring From the Backup Files, 26

 Selecting Roles to Protect, 10
 Setting the Protected Roles List, 10
 Setting the Protection Level, 12
 Setting the Types of Protection for a Role, 21
 SQL/Protect Learn Mode, 13
 SQL/Protect Overview, 4
 sslutils, 34
 System Catalogs, 50

 Types of SQL Injection Attacks, 4

 Virtual Private Database, 31

 結論, 53