The security guide of Security-Enhanced PostgreSQL

(日本語版)

SE-PostgreSQL 開発チーム 著

改版履歴
2007/07/01　日本語版(ベータ)公開

本書は Security-Enhanced PostgreSQL(SE-PostgreSQL)開発チームによる公式のドキュメントであり、現在のバージョンの SE-PostgreSQL が提供するセキュリティ機能の理解に有益な内容を含んでいます。
本書の著作権は SE-PostgreSQL 開発チームが有していますが、BSD ライセンスに基づいて使用・改変・再配布することができます。
BSD ライセンス：http://www.freebsd.org/copyright/freebsd-license.html

目次

1 はじめに

Security-Enhanced PostgreSQL（SE-PostgreSQL）は、オープンソースのリレーショナルデータベースシステム (RDBMS) である PostgreSQL に、SELinux のセキュリティモデルに基づいたアクセス制御メカニズムを提供し、リレーショナルデータベースシステム管理システムのセキュリティに革新的な向上をもたらします。

SE-PostgreSQL の核となるアイデアは、OS レベルのセキュリティポリシーを提供する SELinux との統合と、行レベル／列レベルのアクセス制御を含む細粒度のアクセス制御、特権ユーザであっても不可能な強制アクセス制御です。これらの特徴は、データベースシステムを OS と一体化した情報フロー制御の枠組みに組み込むことを可能にし、情報資産の漏えいや改ざん、破壊といった脅威から保護します。

1.1 Security Enhanced PostgreSQL とは

従来、DBMS におけるアクセス制御は、OS の提供するアクセス制御とは独立に行われてきました。

SE-PostgreSQL は SELinux と連携することによって、OS のセキュリティポリシーに基づいたデータベースオブジェクト(以下、DB オブジェクト)へのアクセス制御を提供します。そして、このインフラストラクチャを利用することによって細粒度の強制アクセス制御を実現しています。

強制アクセス制御

PostgreSQL には特権ユーザの概念があり、クライアントが特権ユーザとして接続している場合には、ネイティブ PostgreSQL のアクセス制御は全てバイパスされます。しかし、SE-PostgreSQL の提供するアクセス制御は、特権ユーザを含む全てのクライアントに対して例外なく実施されます。

この関係は、OS 上での DAC（任意アクセス制御）と MAC（強制アクセス制御）の関係に類似しています。つまり、ネイティブ PostgreSQL のアクセス制御と SE-PostgreSQL のアクセス制御双方がアクセスを許可することなく、クライアントは DB オブジェクトへアクセスすることはできません。

細粒度のアクセス制御

DBMS におけるアクセス制御とは、クライアントの有する権限に基づいて、DB オブジェクトに対するアクセスを許可／禁止することです。DBMS 製品の種類によってアクセス制御の対象となるデータベースオブジェクトの種類は異なりますが、SE-PostgreSQL は、ごく限られた商用 DBMS でしかサポートされていない行レベル／列レベルでのアクセス制御を提供します。

リレーショナルデータベースにおいて、行および列は最も小さな単位の DB オブジェクトであり、データベース管理者は最大限に柔軟な設定を加えることが可能です。

一貫したクライアント権限

SE-PostgreSQL は、データベースに接続するプロセスのセキュリティコンテキストを取得して、それをクライアントの持つ権限として利用します。ネイティブ PostgreSQL のデータベース認証は SE-PostgreSQL のアクセス制御に影響を与えません。

SE-PostgreSQL への接続に TCP/IP を利用する場合には追加の設定が必要です。5.3「Labeled 」の設定を参照してください。

一貫したセキュリティポリシー

SE-PostgreSQL は、管理下にあるデータベースオブジェクトに対して、SELinux セキュリティポリシー

に基づいて、暗黙的又は明示的な方法でセキュリティコンテキストを関連付けることができます。
SE-PostgreSQL は、クライアント及びデータベースオブジェクトのセキュリティコンテキストを利用してアクセス制御を行います。このアクセス制御は全て SELinux セキュリティポリシーに基づいて実行されるため、システムの管理者は、システム全体で一貫したアクセス制御の設定を行うことが可能です。
このような特徴は、以下のような情報フロー制御の枠組みにデータベースを組みこむ事を可能にし、情報漏えいに対する有効な対策となります。

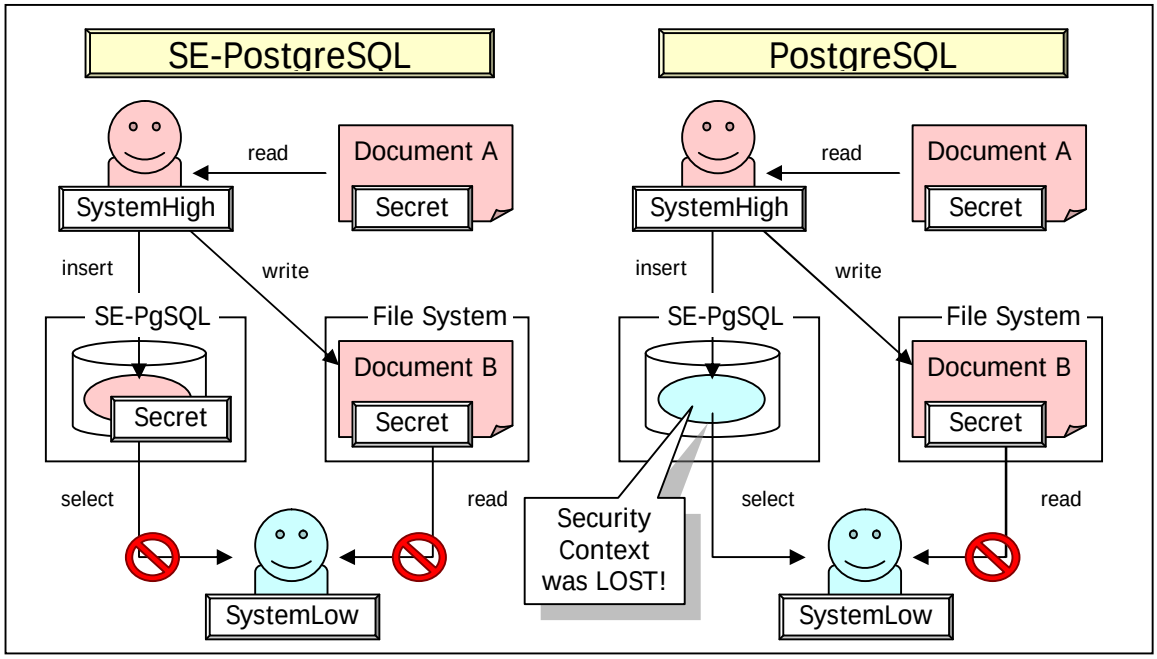


図 1 情報フロー制御

"Secret"とラベル付けされたファイルを読み出し可能なプロセス(SystemHigh)が存在し、そのプロセスがファイルを作成した場合、そのファイルにも"Secret"ラベルが付与されるために、権限の低いプロセス(SystemLow)はこれを読み出すことができません。
一方、SystemHigh プロセスがこの情報をデータベースに挿入した場合、従来の RDBMS ではセキュリティコンテキストは失われ、その結果 SystemLow プロセスからの読み出しを許してしまいます。しかし、SE-PostgreSQL は挿入されたデータにも"Secret"というラベルを付与するため、SystemLow プロセスは RDBMS を経由したとしても"Secret"のデータを参照することができません。

1.2 ライセンスと著作権

SE-PostgreSQL はオリジナルの PostgreSQL に対する拡張として実装されています。オリジナル PostgreSQL 部分の著作権は PostgreSQL global development group が有しています。PGACE を含む SE-PostgreSQL の拡張部分は、SE-PostgreSQL 開発チームが有しています。
SE-PostgreSQL は PostgreSQL の派生物であり、SE-PostgreSQL 開発チームは将来の PostgreSQL への統合を意図しています。そのため、SE-PostgreSQL は PostgreSQL と同一の BSD ライセンスの適用を選択しました。原文は以下の URL を参照してください。

```
BSD License: http://www.postgresql.org/about/licence
```

2　SE-PostgreSQL のアクセス制御

この章では、SE-PostgreSQL のアクセス制御について説明します。

2.1　SELinux との連携

SE-PostgreSQL は PostgreSQL に組み込まれたリファレンスモニタです。クライアントから渡された全ての SQL クエリをチェックし、クライアントが権限を持たない DB オブジェクトへのアクセスを禁止します。
このチェックは Linux カーネルに格納されている SELinux セキュリティポリシーに基づいて実施されます。セキュリティポリシーは SELinux のアクセス制御ルールを全て含んでいます。

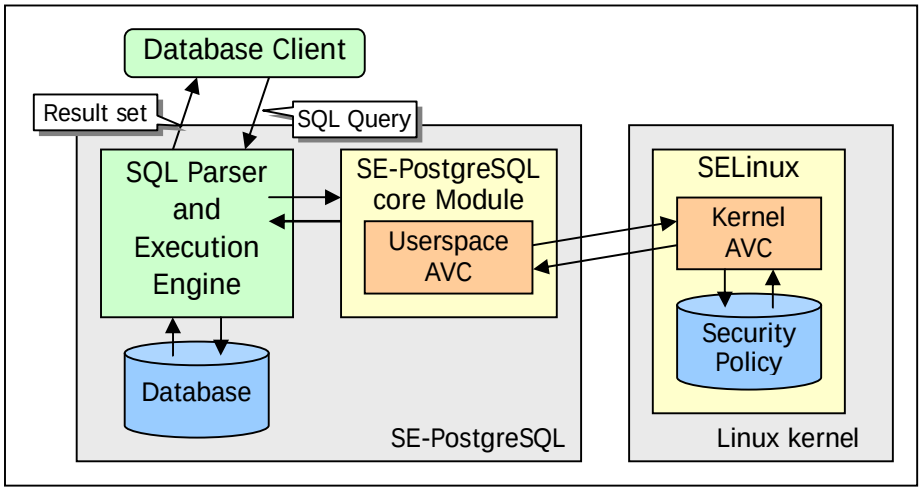


図 2　SE-PostgreSQL アーキテクチャ

SELinux のアクセス制御はホワイトリスト方式を採っており、明示的に「誰が(Subject)」「何に(Object)」「何をする(Action)」かをセキュリティポリシーで定義しなければ、要求されたアクションを拒否します。アクセス制御方式の詳細は「4.SELinux のセキュリティデザイン」を参照してください。
SE-PostgreSQL においても OS と同様に TE や MLS/MCS のアクセス制御が適用され、ポリシーの記述も OS 管理下のオブジェクトクラスと同様に行うことができます。
通常、カーネル呼出しはコストの高い処理ですが、SE-PostgreSQL は AVC(Access Vector Cache)と呼ばれる領域にセキュリティポリシーの一部をキャッシュすることでパフォーマンスの低下を最小限に抑えています。

2.2　セキュリティコンテキスト

SELinux をインストール済みのホストで `id –Z` を実行すると、以下のような文字列が表示されます。

```
[root@masu ~]# id -Z
root:system_r:unconfined_t:SystemLow-SystemHigh
```

これはセキュリティコンテキストと呼ばれる文字列で、SELinux がリソースのセキュリティ属性を識別するために利用するものです。`id –Z` で表示されるのはプロセスのセキュリティコンテキストです。

セキュリティコンテキストが関連付けられるのはプロセスだけではありません。SELinux がアクセス制御の対象とする全てのリソース、例えばファイルやソケット、IPC オブジェクト等にも関連付けられます。

ファイルのセキュリティコンテキストを表示するには ls –Z を実行します。

```
[root@masu ~]# ls -Z /etc/passwd
-rw-r--r--  root root system_u:object_r:etc_t         /etc/passwd
```

セキュリティコンテキストの詳しい解説は「4.1 セキュリティコンテキスト」を参照してください。

セキュリティコンテキストには SELinux がリソースを識別するための情報が全て含まれており、SELinux のアクセス制御は全てセキュリティコンテキストに基づいて行われます。

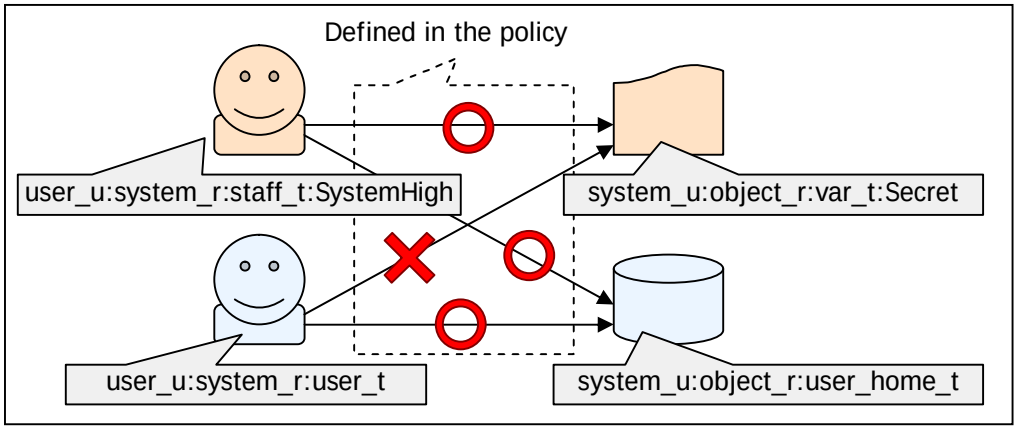


図 3 セキュリティコンテキストとアクセス制御

図 3 は SELinux のアクセス制御の模式図です。プロセス及びファイルは各々のセキュリティコンテキストを有しており、セキュリティポリシーにはセキュリティコンテキスト同士の間に許可する操作を記述します。

SE-PostgreSQL は DB オブジェクトに対するアクセス制御を実施しますが、SELinux のアクセス制御メカニズムを利用するということは、つまり DB オブジェクトに対してセキュリティコンテキストを関連付ける必要があるということです。

SE-PostgreSQL は全てのタプル(行)に対してセキュリティコンテキストを関連付けます。システム列 security_context を参照することでこれを確認することができます。

```
kaigai=# select security_context, * from drink;
            security_context                | id |  name  | price | alcohol
--------------------------------------------+----+--------+-------+--------
 user_u:object_r:sepgsql_table_t            |  1 | coffee |   120 | f
 user_u:object_r:sepgsql_table_t            |  2 | tea    |   120 | f
 user_u:object_r:sepgsql_table_t            |  5 | water  |   110 | f
 user_u:object_r:sepgsql_table_t            |  6 | coke   |   110 | f
 user_u:object_r:sepgsql_table_t:Secret     |  3 | wine   |   360 | t
 user_u:object_r:sepgsql_table_t:Secret     |  4 | beer   |   240 | t
(6 rows)
```

PostgreSQL の内部では、テーブルやカラム等の DB オブジェクトもタプルとして扱われています。これらの DB オブジェクトはシステムカタログと呼ばれる特殊なテーブルに格納されたタプルで、例えばテーブルのメタ情報は pg_class に、カラムのメタ情報は pg_attribute に格納されたタプルによって表現されています。これらのタプルに関連付けられたセキュリティコンテキストが、テーブルやカラムのそれとして扱われます。

SE-PostgreSQL は、上記のようにタプルに関連付けられたセキュリティコンテキストをアクセス制御に利用しています。

2.2.1 クライアントのセキュリティコンテキスト

一方、DB オブジェクトに対するアクセスの主体となるクライアントのセキュリティコンテキストは、DB オブジェクトとは別の方法で関連付けられます。

SELinux はソケットを通じて接続しているもう一方の端点のセキュリティコンテキストを取得する API を持っており、SE-PostgreSQL はこの API を通じて接続元のプロセスのセキュリティコンテキストを取得し、クライアントのセキュリティコンテキストとして利用します。言い換えれば、クライアントはあたかも OS 上でアクセス制御を受けているのと同様に SE-PostgreSQL のアクセス制御を受けます。

なお、UNIX ドメインソケット経由で SE-PostgreSQL に接続する場合は特別な設定は必要ありませんが、TCP/IP 経由で SE-PostgreSQL に接続する場合には IPsec を使った Labeled Network の設定が必要です。詳しくは「5.3 Labeled 」を参照してください。

2.3 SE-PostgreSQL の強制アクセス制御

PostgreSQL がクライアントから渡された SQL クエリを処理し、データベースにアクセスして結果セットを返却するまでの間には、いくつかの処理フェーズが存在します。

SE-PostgreSQL は、SQL クエリ処理プロセスの途中で全ての SQL クエリを検査し、DB オブジェクトを権限を持たないクライアントから保護します。

このチェックは、たとえ PostgreSQL の特権ユーザによって実行されたクエリであっても例外なく適用されます。これは強制アクセス制御と呼ばれ、SE-PostgreSQL の重要な特徴のひとつです。

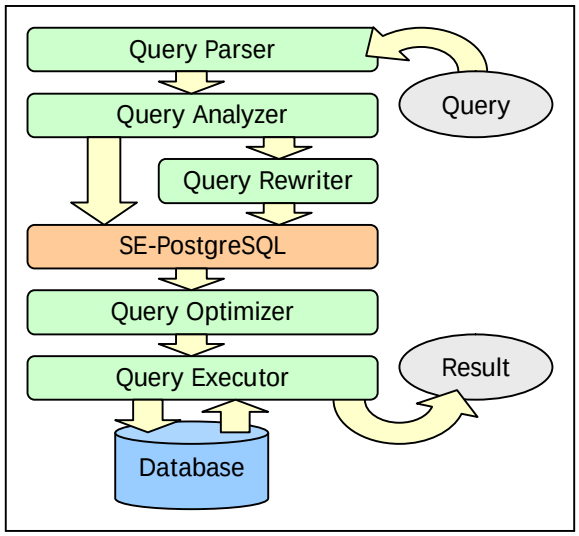


図 4 SE-PostgreSQL のクエリ処理フロー

ネイティブ PostgreSQL は Query analyzer の処理フェーズでアクセス制御を実施しますが、SE-PostgreSQL は Query rewriter の次の処理フェーズでアクセス制御を実施します。

この違いは、SE-PostgreSQL のアクセス制御に興味深い性質を与えています。

Query rewriter の処理フェーズでは、ビュー経由の参照が実際のテーブルへの参照に書き換えられます。例えば、以下の SQL クエリ(1)で定義されたビュー my_view を利用する SQL クエリ(2)を実行する時、Query rewriter は SQL クエリ(2)をビュー定義に基づいて書き換え、あたかも SQL クエリ(3)が入力されたかのように修正します。

```
(1)   CREATE VIEW my_view AS SELECT a, b + c AS x FROM my_table;
(2)   SELECT a, x * x FROM my_view;
(3)   SELECT a, (b + c) * (b + c) FROM my_table;
```

SE-PostgreSQL は、Query rewriter が書き換えた後の SQL クエリを検査して強制アクセス制御を実施します。このことは、SQL クエリの実行の可否が、どのようなビューを経由してテーブルにアクセスしようとも、最終的にアクセスするテープル自身のセキュリティコンテキストによってのみ決定されるということを意味します。

ネイティブ PostgreSQL では、ビューの ACL によってアクセス可能な DB ユーザを制限します。テーブルには複数のビューを定義することができるため、あるテーブルに対するアクセス経路を自明には列挙できません。また、ビュー経由のアクセスを行った場合、テーブルの ACL は評価されないため、ビューを定義する権限を持った DB ユーザは、データベース内の全ての情報にアクセスする権限を付与されているのと等価であるということになります。

ビューが展開された後のクエリを検査するという SE-PostgreSQL の性質によって、同一のテーブルに対するアクセスには同一のアクセス制御ルールが適用されます。この特性は、アクセス制御ポリシーの一貫性の向上に寄与します。

2.4 SQL クエリの検査

SE-PostgreSQL はクライアントから与えられた SQL クエリをスキャンし、その中でアクセスされている DB オブジェクトに対して、クライアントが適切な権限を持っているかどうかを検査します。セキュリティポリシーに反するアクセスがあった場合、SE-PostgreSQL は直ちに SQL クエリの実行を中止してトランザクションをアボートします。

次の SQL クエリを例として考えます。

```
SELECT name, price, weight FROM product;
```

この非常に単純な SQL クエリは、テーブル `product` とカラム `name,price,weight` をアクセスします。SE-PostgreSQL はセキュリティポリシーに基づいて、これら DB オブジェクトに対してクライアントが適切な権限を持っているかどうかを検査します。つまり、この SQL クエリを実行するためには、クライアントはテーブル `product` に対し `table:select` 権限を、カラム `name,price,weight` に対して `column:select` 権限を有している必要があります。

SE-PostgreSQL が利用するオブジェクトクラス/パーミッションの一覧は「3.DB オブジェクトとパーミッション」を参照してください。

次の SQL クエリはやや複雑です。値の計算と WHERE 句を含んでいます。

```
SELECT name, 1.05 * price FROM product WHERE weight > 500;
```

この SQL クエリは、テーブル `product` とカラム `name` をアクセスします。加えて、値の計算過程で `price` カラムに、WHERE 句で `weight` カラムにアクセスします。即ち、クライアントはテーブル `product` に対し `table:{use select}`権限を、カラム `name,price` に対して `column:select` 権限を、WHERE 句で利用しているカラム `weight` に対して `column:use` 権限を有している必要があります。

また、この SQL クエリは'*'演算子と'>'演算子を含みますが、演算子は PostgreSQL で内部的に関数として実装[1]されており、SE-PostgreSQL は関数の実行権限をチェックします。即ち、クライアントはこれらの演算子の実装である関数に対して `procedure:execute` 権限を有している必要があります。

[1] 同じ演算子でも型によって利用する関数が異なる。例えば、'*'演算子は int 型では int4mul だが、double 型では float8mul が使用される。

次の SQL クエリは更新を伴います。

```
UPDATE product SET price = 1.20 * price, name = 'rice' WHERE id = 51;
```

この SQL クエリは、テーブル product 及びカラム price,name,id をアクセスします。しかし、これらのカラムに対するアクセスの目的はそれぞれ異なります。

price カラムは更新の対象であるのと同時に、値の計算過程で利用されます。そのため、クライアントは price カラムに column:{select update}権限を持っている必要があります。name カラムは単純に更新の対象であるので、column:update 権限が要求されます。id カラムは更新の対象ではありませんが、WHERE 句で利用されるために column:use 権限が要求されます。

この SQL クエリは UPDATE 構文ですが、price カラム及び id カラムの参照によって、もはや更新のみではなく、同時に参照されていることに留意してください。そのため、クライアントは product テーブルに table:{use select update}権限を有している必要があります。

```
DELETE FROM product RETURNING *;
```

この SQL クエリはテーブル product に含まれる全てのタプルを削除し、削除したタプルをクライアントに返却します。返却するカラムは、*が指定されているのでテーブル product の全てのカラムです。

product テーブルは削除の対象であると同時に、RETURNING 句で参照されることになります。そのため、クライアントは product テーブルに table:{select delete}権限を持っている必要があります。

また、*を指定するのは、テーブルに含まれる全てのカラムを指定することと等価です。そのため、クライアントは product テーブル内の全てのカラムに対して column:{select}権限を有している必要があります。

DDL 構文を用いる場合にも、これらの DML 構文と同様の SQL クエリの検査が実施されます。例えば、以下の構文はテーブルの名前を変更しますが、これはテーブルのメタ情報の更新を意味します。

```
ALTER TABLE product RENAME TO old_product;
```

そのため、クライアントはテーブルに対して table:setattr 権限を持っている必要があります。また、PostgreSQL は内部的にテーブルに対応するテーブル型を生成するため、このメタ情報を更新するため、テーブル型に対する database:setattr 権限が必要になります。

CREATE TABLE 構文など、複数の DB オブジェクトを同時に多数作成する場合はさらに複雑です。以下の構文は、最も単純なテーブル定義の構文ですが、クライアントはテーブル simple_tbl に対する table:create 権限とカラム x,y に対する column:create 権限が必要になります。

新しく作成されるテーブルやカラムのセキュリティコンテキストは、セキュリティポリシーに基づいて暗黙的に付与されます。SE-PostgreSQL のアクセス制御は、このセキュリティコンテキストに基づいて実施されます。

```
CREATE TABLE simple_tbl (
    x   integer,
    y   varchar(32)
);
```

これ以外にも、TEXT 型を利用した場合に同時に TOAST テーブルが作成されたり、主キー制約を指定した場合に自動的にインデックスが作成されたりするなど、クライアントは対象となる全ての DB オブジ

ェクトに対して必要十分な権限を持っている必要があります。

2.5 SQL クエリの書き換え

SQL クエリの検査フェーズでは、SQL クエリによってアクセスされる DB オブジェクトに対する権限を検査します。しかし、タプルに対するアクセス権をチェックすることはできません。なぜなら、実際にクエリを実行しない限り、SQL クエリがどのタプルをアクセスするのかを特定できないからです。

SE-PostgreSQL は、行レベルのアクセス制御を実施するために SQL クエリの一部を書き換え、クライアントがアクセスに必要な権限を持っていないタプルを結果セットから、あるいは更新/削除の対象から除外するための条件句を追加します。
例えば、以下のクエリは SE-PostgreSQL による SQL クエリ書き換えによって WHERE 句に条件句が追加されています。

```
BEFORE:
    SELECT * FROM product WHERE price > 500;
ALTER:
    SELECT * FROM product WHERE price > 500 AND sepsql_tuple_perms(...);
```

sepgsql_tuple_perms()関数は、クライアントがタプルに対して必要なパーミッションを保持しているかどうかをチェックする関数で、クライアントが必要なパーミッションを持っていたなら true を、必要なパーミッションを持っていなかった場合には false を返却します。
これによって、クライアントがアクセスできないタプルは結果セットから強制的に除外されます。
結果セットのフィルタリングは、JOIN 句を利用した場合やサブクエリーなど、他の全てのテーブルへのアクセスでも実施され、クライアントは常にテーブルに含まれるタプルのサブセットだけを参照することとなります。

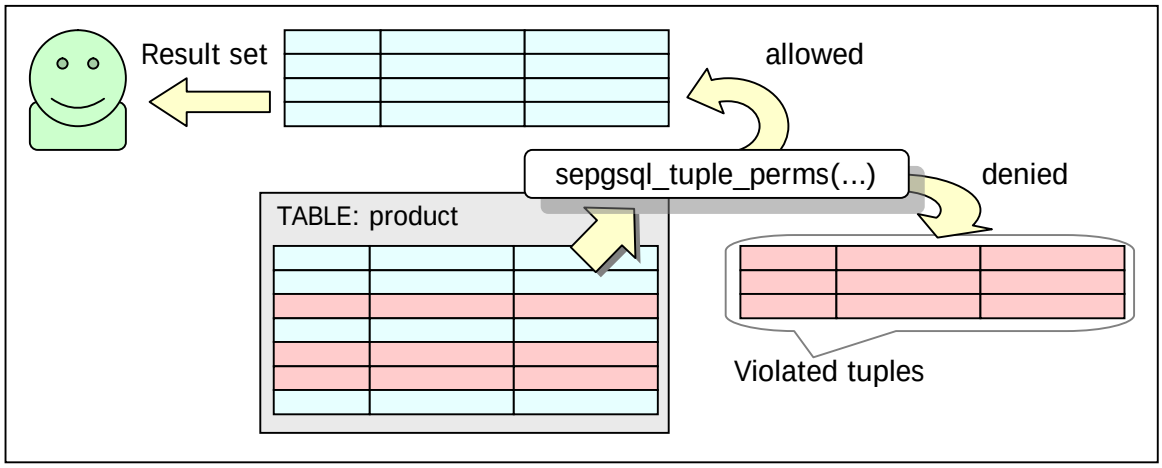


図 5 タプルのフィルタリング

また、アクセス権限の無いタプルをフィルタリングするのは、参照だけではなく更新/削除の場合でも同様です。例えば、クライアントが権限を持たないタプルに対する更新操作を抑制するため、
SE-PostgreSQL は UPDATE 構文を以下のように書き換えます。

```
BEFORE:
    UPDATE product SET price = 1.05 * price;
AFTER:
```

```
UPDATE product SET price = 1.05 * price WHERE sepgsql_tuple_perms(...);
```

2.5.1 特殊ケース: 一意性制約(UNIQUE 制約)

UNIQUE 制約が定義されたカラムの組は、2つ以上の異なるタプルが同一の値のセットを持つことを許可しません。

SE-PostgreSQL の行レベルアクセス制御によって、クライアントがアクセスに必要な権限を持たないタプルは結果セットから除去されます。この時、クライアントの視点からは新しく挿入/更新する値が UNIQUE 制約を満足しているように見えるかもしれません。

しかし、UNIQUE 制約は SE-PostgreSQL の行レベルアクセス制御とは独立に、実テーブル内のタプルに対して一意性制約を満たすよう働きます。したがって、クライアントからは不可視のタプルと新しく挿入/更新しようとするタプルの値が重複して UNIQUE 制約に抵触する可能性があります。理論上は、このような操作を繰り返すことによって、クライアントから不可視のタプルの内容を推測することが可能です。

これは、現在のバージョンの SE-PostgreSQL の制限事項です。この問題を解決するため、将来のバージョンの SE-PostgreSQL では Polyinstantiation テーブルのサポートが予定されています。

2.5.2 特殊ケース: 外部キー制約(Foreign Key 制約)

外部キー制約は2つのテーブル間に参照整合性制約を定義します。PostgreSQL においては、被参照側は常に主キーである必要があります。

SE-PostgreSQL の行レベルアクセス制御下で外部キーを用いる場合に、2つの留意すべき点があります。一つは外部キー値を挿入/更新した場合の被参照側主キー値の可視性に関する問題、もう一つは主キー値を更新/削除した場合の外部キー値へのアクションに関する問題です。

外部キー値を挿入/更新する場合、被参照側テーブルの主キーがクライアントからアクセス可能である必要があります。即ち、クライアントは外部キー側テーブルに対する挿入/更新の権限だけでなく、主キー側テーブルに対する参照の権限も必要となります。

TABLE : team (refered table)

tid	tname
10	'Seattle Mariners'
20	'New York Yankees'
25	'Chicago White Sox'
30	'Boston Red Soxl'

TABLE : player (refering table)

id	tid	name	pitcher
51	10	'Suzuki Ichiro'	false
55	20	'Matsui Hideki'	false
18	30	'Matsuzaka Daisuke'	true
2	10	'Johjima Kenji'	false
29	20	'Igawa Kei'	true

Foreign Key constraint

図 6 外部キー制約の例

図 6 は外部キー制約の例です。player テーブルの tid に、team テーブルの主キーである tid に対する外部キー制約が設定されています。

この時、ピンク色で示されたタプルにアクセスできないクライアントが、player テーブルに以下の内容のタプルを挿入したとします。

```
{id=15, tid=25, name='Iguchi Tadahito', pitcher=false}
```

外部キー制約によって、player テーブルの tid に挿入する値は、必ず team テーブルの tid に存在している必要があります。実際には team テーブルに tid=25 を持つタプルが存在していますが、クライアントにはこのタプルにアクセスする権限が無いため、新しいタプルの挿入は失敗します。

一方、全てのタプルにアクセス可能なクライアントが同一の内容のタプルを player テーブルに挿入した場合、team テーブルの tid=25 のタプルをアクセス可能ですので、新しいタプルの挿入は成功します。

PostgreSQL の外部キー制約には、被参照側の主キー値が更新/削除された時に、参照側の外部キーをそれに連動して更新/削除する機能があります。これらは外部キー制約の ON UPDATE 又は ON DELETE によって指定されるオプションで、CASCADE/SET NULL/SET DEFAULT アクションによって、主キー値の更新/削除と連動して外部キー値を適切な状態に保ちます。

外部キー制約のオプションの詳細については、対応するバージョンの PostgreSQL 文書の「5.3.5 外部キー」節を参照してください。

SE-PostgreSQL は、被参照側主キー値の更新/削除に必要なアクセス権のチェックと、それに連動して実施される参照側外部キー値の更新/削除に必要なアクセス権のチェックを別々に行います。

主キー値の更新/削除に連動して実施される外部キー値の更新/削除では、クライアントは更新/削除の対象となる全てのタプルに対して適切な権限を持っている必要があります。一つ以上のタプルの更新/削除が拒否された場合、SE-PostgreSQL はクエリの実行を直ちに中止してトランザクションをアボートします。

これは、参照整合性を保全するための特殊ケースです。外部キー値の更新/削除では、クライアントが適切な権限を持たないタプルを対象から除外するのではなく、トランザクション全体をアボートすることで、SE-PostgreSQL は参照性合成制約が満たされない状況が発生するのを防いでいます。

2.5.3 特殊ケース：外部結合(OUTER JOIN)

SE-PostgreSQL は、外部結合 (LEFT/RIGHT/FULL OUTER JOIN) を用いてテーブルを結合する場合に特殊な SQL クエリの書き換えを実施します。これは、外部結合を利用した場合に結合条件とは無関係な ON 句での条件が評価されないという PostgreSQL に仕様に起因する問題です。

左外部結合 (LEFT OUTER JOIN) では、結果セットに少なくとも一個の左テーブルに属するタプルが含まれ、これを ON 句の結合条件によって制約することはできません。即ち、単純な条件句の書き換えだけでは、クライアントがアクセスできない左テーブル内のタプルを結果セットから取り除くことはできません。これは、右外部結合(RIGHT OUTER JOIN)や両側外部結合(FULL OUTER JOIN)であっても同様です。

クライアントが正当な権限無くタプルにアクセスすることを防ぐために、SE-PostgreSQL が外部結合に対して実施する SQL クエリの書き換えは、テーブルの参照をサブクエリへと置き換えるものです。

```
SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.id = t2.id;
```

例えば上記の最も単純な左外部結合は、SE-PostgreSQL によって以下の SQL 構文へと書き換えられます。左側テーブルへの参照は条件句付きのサブクエリへと変換され、クライアントがアクセス権を持たないタプルは JOIN の対象とはなりません。

```
SELECT * FROM
       (SELECT * FROM t1 WHERE sepgsql_tuple_perms(...)) AS t1
       LEFT OUTER JOIN t2
       ON t1.id = t2.id and sepgsql_tuple_perms(...);
```

外部結合の対象テーブルのシステム列を参照している場合にも、矛盾なくアクセスできるようにクエリは書き換えられます。

2.6 Trusted Procedure

SE-PostgreSQL は関数の実行を契機としてクライアントのドメイン遷移を引き起こすことが可能で、ドメイン遷移のエントリポイントとなる関数を Trusted Procedure と呼んでいます。

ドメイン遷移は SELinux の概念の一つです。詳細は 4.2.1「ドメイン遷移(Domain Transition)」を参照してください。

クライアントのセキュリティコンテキストは、Trusted Procedure の実行中セキュリティポリシーに基づいて変更されます。そのため、Trusted Procedure の内部からはクライアントが直接アクセスできない DB オブジェクトにアクセスすることが可能です。

このコンセプトは、OS の SetUID コマンドや SQL の Security Invoker に似ています。しかし、ドメイン遷移はセキュリティポリシーによって定義されるもので、DB オブジェクトの所有者とは無関係です。

Trusted Procedure が有効に機能する利用例を紹介します。以下のような構造を持つ customer テーブルがあり、このうち、password 列の内容は認証に利用されますが、クライアントが password 列へ直接アクセスすることを禁止したいとします。

customer テーブル

id	name	email	password
11	'KaiGai'	kaigai@example.com	'aaa'
12	'tak'	tak@example.net	'bbb'
13	'ymj'	ymj@example.org	'ccc'

このような場合、password 列を参照して認証の結果を返却する SQL 関数を定義し、それを Trusted Procedure として設定すれば、クライアントが password 列へ直接アクセスすることを禁止しつつ password 列を利用した認証が可能になります。

次のように、customer.password を利用した認証を行う Trusted Procedure の設定を行うことができます。

最初に SQL 関数を定義します。check_customer_password 関数は、customer.id と認証文字列を引数とし、それが一致しているかどうかを bool 値で返却します。

```
CREATE or REPLACE FUNCTION check_customer_password (integer, text)
       RETURNS bool LANGUAGE 'sql'
       as 'SELECT password = $2 FROM customer WHERE uid=$1';
```

次に、Trusted Procedure のセキュリティコンテキストを変更し、エントリポイントとして設定します。標準のセキュリティポリシーでは sepgsql_trusted_proc_t が Trusted Procedure のエントリポイントとして設定済みです。

```
ALTER FUNCTION check_customer_password (integer, text)
       CONTEXT = 'user_u:object_r:sepgsql_trusted_proc_t';
```

Trusted Procedure を含む SQL クエリを実行した結果、以下のような実行結果を得ることができるでしょう。

クライアントは password 列の内容を読み出すことができないものの、認証を行うには十分な情報を得ていることに留意してください。

```
kaigai=# SELECT * FROM customer WHERE id = 11;
ERROR:  SELinux: denied { select } scontext=user_u:system_r:initrc_t
tcontext=user_u:object_r:sepgsql_secret_table_t tclass=column name=password
```

最初に password 列を含む全てのカラムを参照しようとして SE-PostgreSQL にアクセスを拒否されています。

```
kaigai=# SELECT id, name, email FROM customer WHERE id = 11;
 id |  name  |       email
----+--------+-------------------
 11 | KaiGai | kaigai@example.com
(1 row)
```

次に、password 列を SELECT の対象から外し、これにより結果セットを得ましたが、ここには password 列は含まれていません。

```
kaigai=# SELECT id, name, email, check_customer_password(id, 'bbb') FROM customer;
 id |  name  |       email        | check_customer_password
----+--------+--------------------+-------------------------
 11 | KaiGai | kaigai@example.com | f
 12 | tak    | tak@example.net    | t
 13 | ymj    | ymj@example.org    | f
 14 | erina  | erina@example.mil  | f
(4 rows)
```

次に Trusted Procedure を呼び出します。クライアントには check_customer_password 関数によって認証の結果が返却されますが、password 列の内容自体は外部に出ていません。

3 DB オブジェクトとパーミッション

この章では、DB オブジェクトの種類ごとに SE-PostgreSQL のアクセス制御がどのように行われるのかを解説します。SELinux のアクセス制御に関する一般的は情報は「4.SELinux のセキュリティデザイン」で解説しています。本章と併せて参照してください。

3.1 オブジェクトクラスとパーミッション

例えばテーブルと関数ではアクセスの方法が異なるように、DB オブジェクトはそれぞれ固有の特徴を持っています。SELinux では、これをオブジェクトクラスとそれに関連付けられたパーミッションの集合(アクセスベクタ)として記述します。

SE-PostgreSQL のセキュリティポリシーを記述するため、6つのオブジェクトクラスと、それらに関連付けられた合計 58 種類のパーミッションが新しく追加されました。下の表はその一覧です。

database	table	procedure	column	blob	tuple
create	create	create	create	create	relabelfrom
drop	drop	drop	drop	drop	relabelto
getattr	getattr	getattr	getattr	getattr	use
setattr	setattr	setattr	setattr	setattr	select
relabelfrom	relabelfrom	relabelfrom	relabelfrom	relabelfrom	update
relabelto	relabelto	relabelto	relabelto	relabelto	insert
access	use	execute	use	read	delete
install_module	select	entrypoint	select	write	
load_module	update		update	import	
get_param	insert		insert	export	
set_param	delete				
	lock				

表 1 オブジェクトクラス/パーミッション一覧

理想的には全ての種類の DB オブジェクトが固有のオブジェクトクラスを持つべきです。

しかし PostgreSQL では、トリガーやロールなど、固有のオブジェクトクラスを持たない DB オブジェクトも存在します。これらに対するアクセス制御は、特殊なテーブルであるシステムカタログに対する操作として `table`, `column`, `tuple` クラスのパーミッションを利用して行われます。

3.2 各 DB オブジェクト共通のアクセス制御

タプルを除く DB オブジェクトには、いくつかの共通する操作があります。即ち、DB オブジェクトの生成(`create`)、削除(`drop`)、メタ情報の取得(`getattr`)と設定(`setattr`)、そしてセキュリティ属性の変更(`relabelfrom`、`relabelto`)です。

タプルの場合、これらのうちセキュリティ属性の変更だけが定義されています。これは、生成･削除が

insert/delete と等価であること、そしてそれ自身のメタ情報を持っていないことが理由です。
本節では、これらの DB オブジェクトに共通するパーミッションについて解説します。

create パーミッション

CREATE TABLE などの CREATE 構文で DB オブジェクトを作成する時、新しい DB オブジェクトには暗黙的にセキュリティコンテキストが関連付けられます。クライアントはこの新しい DB オブジェクトに対する create パーミッションを持っている必要があります。

一回の SQL クエリで複数個の DB オブジェクトが生成される可能性に留意してください。例えば CREATE TABLE 構文では、テーブルの他に複数個のカラムが生成されます。

なお、ラージオブジェクトは lo_create()関数によって生成され、この時に create パーミッションが評価されます。

drop パーミッション

DROP TABLE などの DROP 構文で DB オブジェクトを削除する時、クライアントは対象の DB オブジェクトに対して drop パーミッションを持っている必要があります。

一回の SQL クエリで複数個の DB オブジェクトが削除される可能性に留意してください。例えば DROP TABLE 構文では、テーブル以外にも、それに含まれる全てのカラムが削除されます。

なお、ラージオブジェクトは lo_unlink()関数によって削除され、この時に drop パーミッションが評価されます。

getattr パーミッション

前述のように、タプルを除く DB オブジェクトは、システムカタログと呼ばれる特殊なテーブルに格納されたタプルとして表現されています。システムカタログに対して SELECT を実行することで DB オブジェクトのメタ情報を参照することができます。

この時、クライアントは参照する DB オブジェクトに対して getattr パーミッションを持っている必要があります。

setattr パーミッション

ALTER TABLE などの ALTER 構文で DB オブジェクトのメタ情報を更新する時、クライアントは対象の DB オブジェクトに対して setattr パーミッションを持っている必要があります。

ALTER 構文であっても、DB オブジェクトの作成/削除として扱われるケースがあることに留意してください。例えば、ALTER TABLE tblname ADD colname ... ; 構文は、新しいカラムの作成を意味します。

relabelfrom/relabelto パーミッション

拡張された ALTER 構文や security_context 列に対する UPDATE によって、DB オブジェクトのセキュリティコンテキストを変更することができます。relabelfrom 及び relabelto パーミッションはこの時に評価され、クライアントは DB オブジェクトの変更前のセキュリティコンテキストに対して relabelfrom パーミッションを、変更後のセキュリティコンテキストに対して relabelto パーミッションを持っている必要があります。

3.3 データベースに対するアクセス制御

database オブジェクトクラスは、データベースそれ自身に対する権限セットを含んでおり、5つの固有パーミッションを持っています。

暗黙的セキュリティコンテキスト

CREATE DATABASE 構文を用いて新しくデータベースを作成する場合、新しいデータベースのタイプは SE-PostgreSQL サーバプロセスのドメインを継承します。また、SE-PostgreSQL サーバプロセスのドメインをターゲットとしてタイプ遷移ルールを記述することができます。

標準のセキュリティポリシーでは、タイプ遷移ルールによって、データベースに sepgsql_db_t タイプが付与されます。

access パーミッション

クライアントがデータベースに接続する時、接続先のデータベースに対して access パーミッションを持っている必要があります。SE-PostgreSQL に接続するため、最低限必要なパーミッションです。

install_module パーミッション

動的リンクライブラリ(DLL)をインストールする時、クライアントは DLL とデータベースのそれぞれに対して install_module パーミッションを持っている必要があります。

DLL は LOAD 構文によって明示的にロードされる以外にも、DLL 内で実装された関数を定義することによって暗黙的にロードされる事に留意してください。

例えば、staff_t ドメインで動作しているクライアントが、sepgsql_db_t タイプのデータベースに接続しており、ロードする DLL のタイプが lib_t である時には、以下のようなセキュリティポリシーが必要です。

```
allow staff_t sepgsql_db_t : database install_module;
allow staff_t lib_t : database install_module;
```

load_module パーミッション

動的リンクライブラリ(DLL)が SE-PostgreSQL のプロセス空間にロードされる時、データベースは DLL に対して load_module パーミッションを持っている必要があります。SE-PostgreSQL の扱う中では、load_module はクライアントがサブジェクトとならない唯一のパーミッションです。

install_module との違いは、DLL が SE-PostgreSQL のプロセス空間にロードされる時に何度でも評価されるということです。

以下は、セキュリティポリシーのサンプルです。

```
allow sepgsql_db_t lib_t : database load_module;
```

get_param パーミッション

SHOW 構文によって実行時パラメータを参照する時、クライアントは接続中のデータベースに対して get_param パーミッションを持っている必要があります。

このパーミッションはデータベースに対して評価されるため、個々の実行時パラメータに対して個別に許可/禁止を設定することはできません。

set_param パーミッション

SET 構文によって実行時パラメータを設定する時、クライアントは接続中のデータベースに対して set_param パーミッションを持っている必要があります。

このパーミッションはデータベースに対して評価されるため、個々の実行時パラメータに対して個別に許可/禁止を設定することはできません。

3.4 テーブルに対するアクセス制御

table オブジェクトクラスは、テーブルに対する権限セットを含んでおり、6 つの固有のパーミッションを持っています。

暗黙的セキュリティコンテキスト

CREATE TABLE 構文を用いて新しくテーブルを作成する場合、新しいテーブルのタイプはデータベースのタイプを継承します。また、データベースのタイプをターゲットとして table クラスのタイプ遷移ルールを記述することができます。

標準のセキュリティポリシーでは、タイプ遷移ルールによって新しいテーブルに sepgsql_table_t タイプが付与されます。

use パーミッション

クライアントは、SQL 構文の条件句(WHERE 句、JOIN〜ON 句、HAVING 句)や、GROUP BY 句、ORDER BY 句で参照するカラムを含んでいるテーブルに対して use パーミッションを持っている必要があります。

例えば以下の UPDATE 構文では、drink テーブルに対する update パーミッションだけでなく、use パーミッションも必要になります。なぜなら、WHERE 句で drink テーブルの id カラムを参照しているからです。

```
UPDATE drink SET price = 120 WHERE id = 4;
```

select パーミッション

SELECT 構文によってテーブルを参照する時、又は COPY TO 構文によってテーブルの内容をダンプする時、クライアントは対象のテーブルに対して select パーミッションを持っている必要があります。

これ以外にも、UPDATE 構文で更新後の値の計算にテーブルの参照を含む計算式を利用する場合や、RETURNING 句で更新系 SQL 構文の実行結果をクライアントに返却する場合に、このパーミッションは評価されます。

例えば以下の DELETE 構文では、drink テーブルに対する delete パーミッションだけでなく、select パーミッションも必要になります。なぜなら、RETURNING 句によってテーブルの内容を参照しているからです。

```
DELETE FROM drink RETURNING *;
```

use パーミッションとの違いは、クライアントが何らかの方法でテーブルの内容を読み出し、その内容をクライアントに返却する場合に select パーミッションが評価されるという点です。use パーミッションは条件句や ORDER BY 句などテーブルの内容がクライアントに返却されない場合に評価されます。

update パーミッション

UPDATE 構文によってテーブルを更新する時、クライアントは対象のテーブルに対して update パーミッションを持っている必要があります。

insert パーミッション

INSERT 構文によってテーブルにタプルを挿入する時、又は COPY FROM 構文によってテーブルにリストアする時、クライアントは対象のテーブルに対して insert パーミッションを持っている必要があ

ります。

delete パーミッション

DELETE 構文によってテーブルからタプルを削除する時、又は TRUNCATE 構文によってテーブルの内容を消去する時、クライアントは対象のテーブルに対して delete パーミッションを持っている必要があります。

lock パーミッション

LOCK 構文によって明示的にテーブルロックを獲得する時、クライアントは対象のテーブルに対して lock パーミッションを持っている必要があります。

3.5 カラムに対するアクセス制御

column オブジェクトクラスは、カラムに対する権限セットを含んでおり、4 つの固有のパーミッションを持っています。

暗黙的セキュリティコンテキスト

CREATE TABLE や ALTER TABLE 構文を用いて新しくカラムを作成する場合、新しいカラムのタイプは、カラムの属するテーブルのタイプを継承します。また、テーブルのタイプをターゲットとして、column クラスのタイプ遷移ルールを記述することができます。

use パーミッション

クライアントは、SQL 構文の条件句(WHERE 句、JOIN〜ON 句、HAVING 句)や、GROUP BY 句、ORDER BY 句で参照するカラムに対して use パーミッションを持っている必要があります。

例えば以下の SELECT 構文では、id および name カラムに対する select パーミッションだけでなく、alcohol および price カラムに対する use パーミッションが必要になります。なぜなら、WHERE 句で alcohol カラムを、ORDER BY 句で price カラムを参照しているからです。

```
SELECT id,name FROM drink WHERE alcohol = true ORDER BY price;
```

select パーミッション

クライアントがカラムから情報を取り出す時、対象のカラムに対して select パーミッションを持っている必要があります。

最も典型的なケースは SELECT 構文の選択リストにカラムまたはカラムを含む式を含めている場合です。しかし、それ以外にも以下のケースで、クライアントは対象のカラムに対する select パーミッションを評価されることに留意してください。

– INSERT/UPDATE/DELETE 構文の RETURNING 句として指定した場合
– COPY TO 構文でテーブルをダンプした場合
– 関数の引数にカラムを指定した場合

例えば、以下の SQL クエリを実行する場合、クライアントは uid および uname カラムに対する select パーミッションだけでなく、birthday カラムに対する select パーミッションも必要になります。

```
SELECT uid, uname, age(birthday) FROM person;
```

use パーミッションとの違いは、クライアントが何らかの方法でカラムの内容を読み出し、その内容をクライアントに返却する場合に select パーミッションが評価されるという点です。use パーミッションが評価されるのは、条件句や ORDER BY 句でカラムを使用した場合で、この場合はカラムの内容は

クライアントには返却されません。

update パーミッション

UPDATE 構文によってテーブルを更新する時、クライアントは対象のカラムに対して update パーミッションを持っている必要があります。更新の対象でないカラムは評価されません。

insert パーミッション

INSERT 構文によって新しいタプルを挿入する時、クライアントが明示的に値を指定したカラムに対して insert パーミッションを持っている必要があります。

新しいタプルの中で、値を指定しなかったカラムに対しては insert パーミッションは評価されません。これらのフィールドには NULL 値又はデフォルト値がセットされます。

3.6 タプルに対するアクセス制御

tuple オブジェクトクラスは、タプルに対する権限セットを含んでおり、7 つの固有のパーミッションを持っています。

暗黙的セキュリティコンテキスト

INSERT 構文や COPY FROM 構文を用いてタプルを挿入する場合、新しいタプルのタイプは、タプルの属するテーブルのタイプを継承します。また、テーブルのタイプをターゲットとして、tuple クラスのタイプ遷移ルールを記述することができます。

pg_class や pg_proc など、一部のシステムカタログへの INSERT は、tuple クラスに対する操作としては扱われないことに留意してください。なぜなら、これらのシステムカタログへのタプルの挿入は、テーブルや関数などの作成と等価であるため、これらの操作はそれぞれ対応するオブジェクトクラスの create として扱われます。

relabelfrom/relabelto パーミッション

security_context 列に対する UPDATE によって、タプルのセキュリティコンテキストを変更することができます。relabelfrom 及び relabelto パーミッションはこの時に評価され、クライアントはタプルの変更前のセキュリティコンテキストに対して relabelfrom パーミッションを、変更後のセキュリティコンテキストに対して relabelto パーミッションを持っている必要があります。

use パーミッション

SQL クエリが条件句(WHERE 句、JOIN〜ON 句、HAVING 句)や、GROUP BY 句、ORDER BY 句を含んでいる場合、クライアントはタプルに対して use パーミッションを持っている必要があります。

クライアントが use パーミッションを持っていないタプルは、結果セット又は更新/削除の対象から除外されます。

select パーミッション

SELECT 構文によってタプルを読み出す場合や、RETURNING 句によって更新系クエリの対象となったタプルを返却する場合、または COPY TO 構文によってテーブルをダンプする場合に、クライアントはタプルに対して select パーミッションを持っている必要があります。

クライアントが select パーミッションを持っていないタプルは結果セットから除外されます。

update パーミッション

UPDATE 構文によってタプルを更新する時、クライアントはタプルに対して update パーミッションを

持っている必要があります。

クライアントが update パーミッションを持っていないタブルは更新の対象から除外されます。

insert パーミッション

INSERT 構文によってタプルを挿入する時、又は COPY FROM によってテーブルをリストアする時、クライアントは新しく挿入されるタプルに対して insert パーミッションを持っている必要があります。

新しいタプルのセキュリティコンテキストは暗黙的に付与されますが、security_context 列に値をセットした INSERT 構文を実行することで、明示的にセキュリティコンテキストを指定することができます。どちらの場合であっても、insert パーミッションを持たないタプルを挿入することはできません。

delete パーミッション

DELETE 構文や TRUNCATE 構文によってタプルを削除する時、クライアントは対象となるタプルに対して delete パーミッションを持っている必要があります。クライアントが delete パーミッションを持っていないタプルは、削除の対象から除外されます。

SE-PostgreSQL において TRUNCATE 構文は何のメリットも無いことに留意してください。内部的に、TRUNCATE 構文は無条件の DELETE 構文に変換され、delete パーミッションを持たないタプルはテーブルに残り続けます。

3.7 関数に対するアクセス制御

procedure オブジェクトクラスは、関数に対する権限セットを含んでおり、2つの固有パーミッションを含んでいます。

暗黙的セキュリティコンテキスト

CREATE FUNCTION 構文を用いて新しく関数を作成する場合、新しい関数のタイプはデータベースのタイプを継承します。また、データベースのタイプをターゲットとして、procedure クラスのタイプ遷移ルールを記述することができます。

標準のセキュリティポリシーでは、新しく生成された関数には sepgsql_proc_t タイプが付与されるよう設定されています。

execute パーミッション

SQL クエリに含まれる関数を実行する時、クライアントは関数に対して execute パーミッションを持っている必要があります。

PostgreSQL は演算子の実装を SQL 関数として持っています。例えば、4byte 整数型同士の比較演算のために int4eq 関数が実行されます。この場合であっても、クライアントは対象の関数に対する execute パーミッションを持っている必要があります。

一方、PostgreSQL は自身の内部処理のために SQL 関数を呼び出すことがありますが、この場合は関数の execute パーミッションを評価しません。

entrypoint パーミッション

Trusted Procedure として定義されている関数を実行する時、クライアントは関数に対して entrypoint パーミッションを持っている必要があります。Trusted Procedure の詳細については 2.6「Trusted Procedure」を参照してください。

Trusted Procedure の実行はドメイン遷移を伴います。従って、クライアントは遷移先のドメインに対して procedure オブジェクトクラスの transition パーミッションを持っている必要があります。

Trusted Procedure を設定するためのセキュリティポリシーの記述は以下の通りです。

```
allow <client domain> <procedure type>
            : procedure { execute entrypoint };       ... (1)
allow <client domain> <new domain}
            : process { transition };                 ... (2)
type_transition <client domain> <procedure type>
            : process <new domain> ;                  ... (3)
```

(1)の設定により、クライアントは Trusted Procedure を実行することができます。(2)の設定により、クライアントは<new domain>へ遷移することができます。(3)の設定により、Trusted Procedure を実行した時に、<new domain>へのドメイン遷移が発生することになります。

3.8 ラージオブジェクトに対するアクセス制御

blob オブジェクトクラスは、ラージオブジェクトに対する権限セットを含んでおり、4つの固有パーミッションを含んでいます。ネイティブ PostgreSQL はラージオブジェクトに対するアクセス制御を提供していないため、SE-PostgreSQL によるものが唯一のアクセス制御となります。

暗黙的セキュリティコンテキスト

lo_create()関数を用いて新しいラージオブジェクトを作成する場合、新しいラージオブジェクトのタイプはデータベースのタイプを継承します。また、データベースのタイプをターゲットとして、blob クラスのタイプ遷移ルールを記述することができます。

標準のセキュリティポリシーでは、新しく生成されたラージオブジェクトには sepgsql_blob_t タイプが付与されます。

read パーミッション

loread()関数を用いてラージオブジェクトを読み込み時、クライアントはラージオブジェクトに対して read パーミッションを持っている必要があります。pg_largeobject システムカタログを直接参照する場合であっても同様です。

write パーミッション

lowrite()関数を用いてラージオブジェクトに書き込む時、クライアントはラージオブジェクトに対して write パーミッションを持っている必要があります。pg_largeobject システムカタログを直接更新する場合であっても同様です。

import/export パーミッション

lo_import()関数は、指定されたファイルを OS のファイルシステム上から読込み、ラージオブジェクトとして格納します。lo_export()関数は、指定されたラージオブジェクトを、指定されたファイル名で OS のファイルシステム上に書き出します。どちらも、引数として与えるファイル名が、クライアント OS 上のものではなく、サーバ OS 上のものであることに留意してください。

lo_import()関数と lo_export()関数においては、クライアントと DB オブジェクトの間で直接データの read/write が行われているわけではありません。OS 上のファイルを read/write しているのはあくまでも SE-PostgreSQL のサーバプロセスであり、これに伴う read/write は SELinux によってカーネルレベルで制御されています。

一方、クライアントは lo_import()や lo_export()によって、サーバプロセスに対してこれらの

処理の開始を指示します。従って、これはクライアントと SE-PostgreSQL サーバプロセス間の関係です。

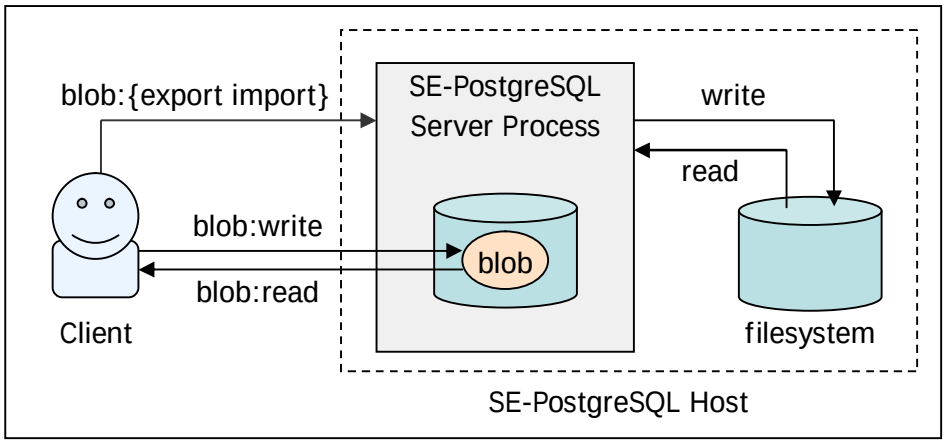


図 7 ラージオブジェクトの import/export

`import/export` パーミッションは以下のように機能します。

`lo_import()`関数を呼び出してラージオブジェクトをインポートする時、クライアントは SE-PostgreSQL サーバプロセスに対して `import` パーミッションを持っている必要があります。また、`lo_export()`関数を呼び出してラージオブジェクトをエクスポートする時、クライアントは SE-PostgreSQL サーバプロセスに対して `export` パーミッションを持っている必要があります。

4　SELinux のセキュリティデザイン

本章では、SE-PostgreSQL に限定しない一般的な SELinux のアクセス制御方式について概説します。

4.1　セキュリティコンテキスト

セキュリティコンテキストとは以下の書式で表現される文字列で、SELinux がアクセス制御に必要とする情報を全て含んでいます。SELinux 設定済みのシステムでは、全てのプロセスやファイル・ソケットなどのオブジェクトにセキュリティコンテキストが付与されています。SE-PostgreSQL 管理下にある DB オブジェクトに対しても同様です。

セキュリティコンテキストの各フィールドは":"によって4つの部分に分離することができます。これらは各々「ユーザ」「ロール」「タイプ(ドメイン)」「MLS ラベル」を意味します。

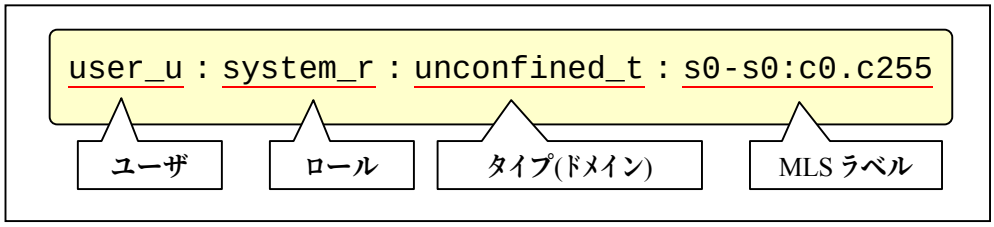


図 8 セキュリティコンテキスト

ユーザは OS 上のユーザに対応します。セキュリティポリシーで定義されていないユーザの場合は、その代用として「user_u」を利用します。

ロールは後述の RBAC で利用される識別子です。プロセスのセキュリティコンテキストでのみ意味をもち、それ以外は一律に「object_r」が付与されます。

タイプは後述の TE で利用される識別子です。プロセスに付与されたタイプのことを特別にドメインと呼び、それ以外のオブジェクトとは区別しています。

MLS ラベルは後述の MLS/MCS で利用される識別子です。mcstrans サービスが起動しているとき、MLS ラベルを「SystemHigh」などの意味を持った別名で表示することができます。

SELinux は TE, MLS/MCS, RBAC によるアクセス制御を提供しており、これらは全てプロセスのセキュリティコンテキスト、及びそれらがアクセスするオブジェクトのセキュリティコンテキストに基づいて行われます。以下の節で、SELinux のアクセス制御方式をそれぞれ説明します。

4.2　TE(Type Enforcement)

Type Enforcement とは、プロセスのドメインとアクション、それによってアクセスされるオブジェクトのタイプの3つのファクターから成るアクセス制御方式で、SELinux における最も重要なアクセス制御方式です。

セキュリティポリシーには、どのドメインが、何のタイプに対して、どのようなアクションを実行可能であるかが全て列挙されており、明示的に許可されていないアクションは全て実行を拒否されます。

ここで言うアクションとは、オブジェクトクラスとアクセスベクタの組み合わせで、例えばファイルに対する読出しのアクションであれば「file:read」と、TCP ソケットが接続を待ち受けるなら「tcp_socket:accept」となります。アクセスベクタの種類は対象のオブジェクトクラスによって異なり

ます。プロセスがアクションの対象となることもあり、例えばシグナルの送出は「process:kill」となります。

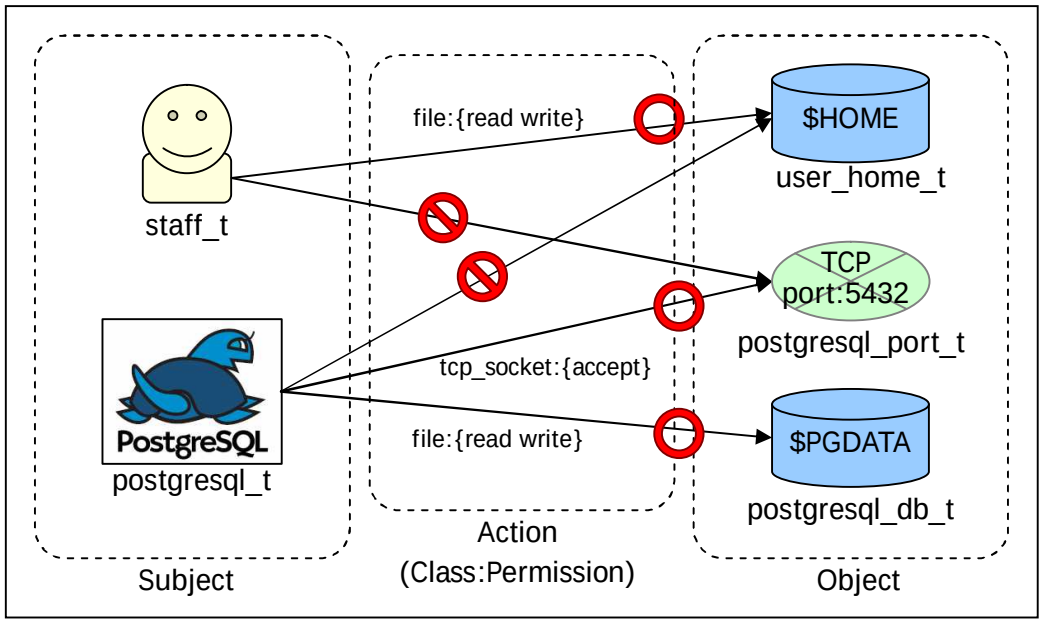


図 9 Type Enforcement

セキュリティポリシーにおいて、TE に基づいたアクセス制御ルールは以下のように記述されています。

```
allow postgresql_t postgresql_db_t : file { read getattr };
```

このようなアクセス制御ポリシーを、アプリケーションが必要とする全ての項目について事前に記載する必要があります。

しかし、このような記述を逐一行うと、セキュリティポリシーの記述が煩雑になってメンテナンスが困難となるため、SELinux コミュニティでは Reference Policy と呼ばれる高度にモジュール化されたマクロ群を提供しています。これにより、セキュリティポリシーの開発者は可読性が高くモジュール化されたセキュリティポリシーを開発することができます。

4.2.1 ドメイン遷移(Domain Transition)

特別な設定が行われていない場合、プロセスは親プロセスのドメインを継承します。

つまり、unconfined_t ドメインで動作するプロセスが子プロセスを起動した場合は、子プロセスも unconfined_t ドメインで、staff_t ドメインで動作するプロセスが子プロセスを起動した場合には staff_t ドメインで子プロセスが動作します。

しかし、これだけでは全てのプロセスが全く同一の権限で動作することになり、意味のあるアクセス制御を行うことができません。これを解決するのがドメイン遷移です。

ドメイン遷移とは、プロセスのドメインと実行可能バイナリ形式ファイルのタイプに基づいて、新しいプロセスが動作するドメインを決定するメカニズムです。これによって、子プロセスは親プロセスとは異なったドメインで動作することが可能になります。

セキュリティポリシーにおいて、ドメイン遷移は以下のように記述することができます。

```
TYPE_TRANSITION <source domain> <target type> : process <dest domain> ;
```

例えば、システム初期化スクリプトが Apache を起動し、Apache が httpd_t ドメインで動作する場合には以下のような記述が必要になります。

```
TYPE_TRANSITION initrc_t httpd_exec_t : process httpd_t ;
```

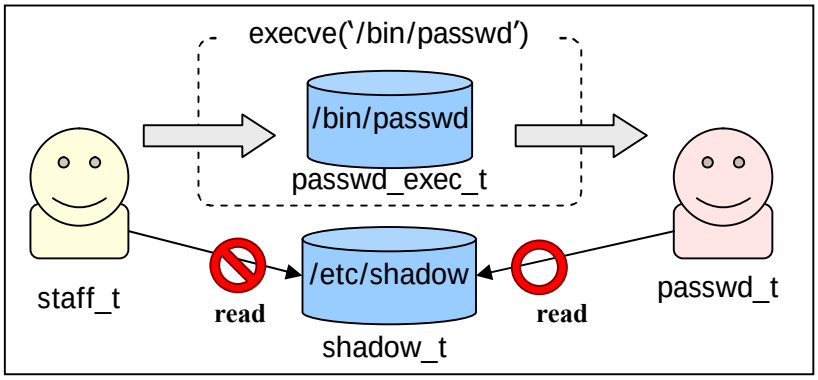


図 10 ドメイン遷移

図 10 は典型的なドメイン遷移の模式図です。ユーザシェルからは直接/etc/shadow ファイルにアクセスすることはできませんが、/bin/passwd コマンドを実行することで passwd_t ドメインへ遷移し、/etc/shadow へのアクセスが可能となります。

これは、/bin/passwd コマンドという安全な手続きを経由することによってのみ、パスワードファイルにアクセスを許可するという考え方に基づいています。このアイデアは伝統的 UNIX の SetUID に似ていますが、遷移元ドメインによって異なった結果を得ることが可能であることや、遷移先のドメインをより制約された権限とすることが可能であるなどの違いがあります。

Linux では/sbin/init を頂点として全てのプロセスが生成されますが、これらのプロセスはセキュリティポリシーに基づいてドメイン遷移を繰り返すことで、各プロセスは適切な権限で実行されることになります。

4.3 RBAC(Role Based Access Control)

SELinux において、RBAC は前述のドメイン遷移を制御するためのものです。

セキュリティポリシーの定義により、ロールには任意の数のドメインを関連付けることが可能です。プロセスがドメイン遷移によって遷移できるのは、プロセスのロールに関連付けられた範囲のドメインのみに制約されます。プロセスのロールは、ドメイン遷移の前後で不変であることに留意してください。

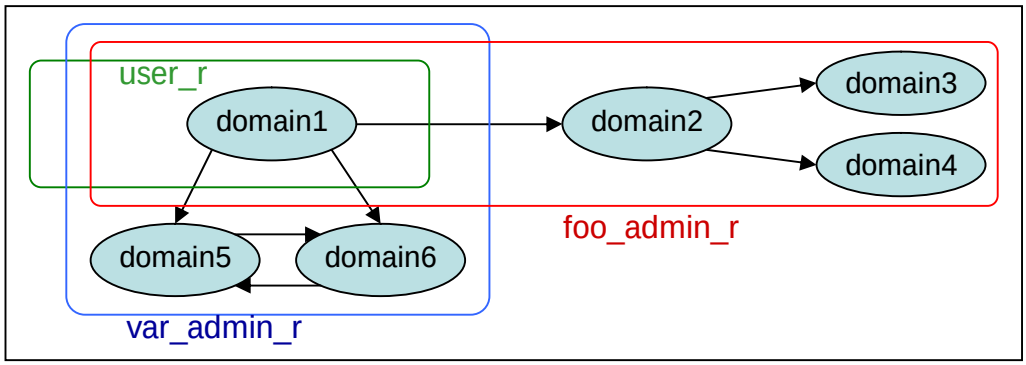


図 11 RBAC 模式図

図 11 は RBAC の概念図です。domain1 から domain2, domain5, domain6 にドメイン遷移が可能ですが、プロセスが user_r ドメインに属している場合はドメイン遷移を行うことができません。また、foo_admin_r に属している場合は domain5, domain6 にドメイン遷移を行うことができず、

var_admin_r に属している場合は domain2 にドメイン遷移を行うことができません。
このような制約を行うことにより、システムの一部のサービスのみに限定した管理者を設定することが可能です。しかしそのためには、システムの提供するサービス毎にドメインが十分に細かく分割されている必要があります。

4.4 MLS(Multi Level Security) & MCS(Multi Category Security)

MLS は伝統的な Bell-La-Padulla モデルに基づくアクセス制御を提供する仕組みです。
プロセス及びファイルなどのオブジェクトには、上下関係に基づく機密度(sensitivity)と、包含関係に基づく機密区分(category)が割り当てられます。プロセスとオブジェクトのそれぞれの機密度・機密区分を用いて、以下のルールに基づいたアクセス制御が実施されます。

1. プロセスがオブジェクトから情報を読み込む際には、プロセスの機密度がオブジェクトの機密度よりも高いか同一でなければならない。
2. プロセスがオブジェクトから情報を読み込む際には、プロセスの機密区分がオブジェクトの機密区分を完全に包含していなければならない。
3. プロセスがオブジェクトへ情報を書き込む際には、プロセスの機密度・機密区分がオブジェクトのそれと同一でなければならない。

このルールを適用することにより、機密度の高い情報が機密度の低いドメインへと、また関係のない機密区分のドメインへと流出することを禁止することができます。

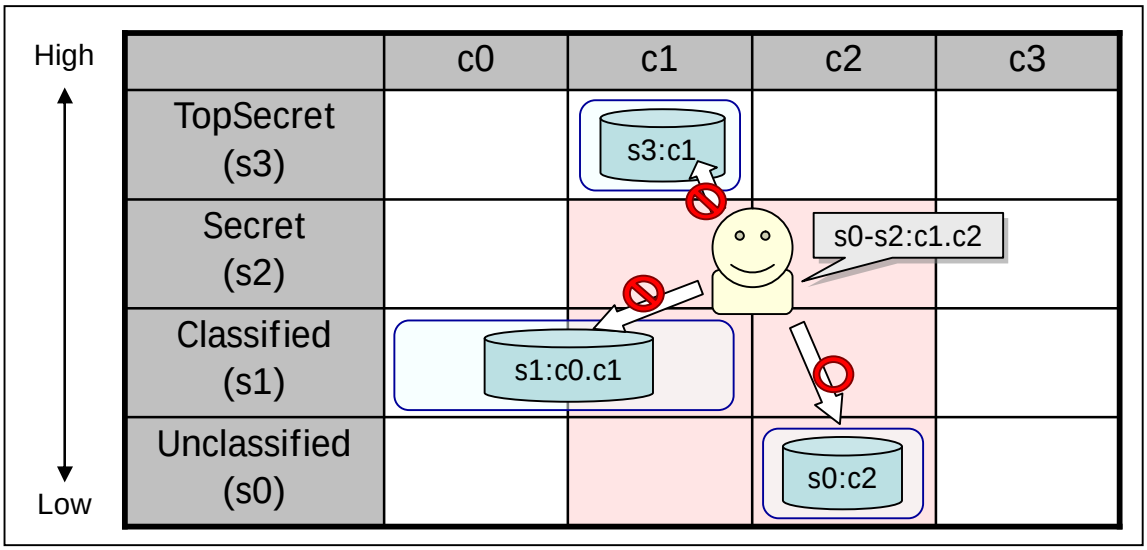


図 12 MLS 概念図

図 12 は MLS によるアクセス制御の模式図です。プロセスの MLS ラベルが「s0-s2:c1.c2」である時、プロセスはピンク色で図示された機密度・機密区分を支配しています。
プロセスは「s0:c2」のファイルにアクセスすることができます。しかし、「s3:c1」のファイルは機密度がプロセスよりも高いためにアクセスすることができません。同様に、「s1:c0.c1」のファイルはプロセスの機密区分に完全に包含されてはいないため、アクセスすることはできません。

MCS は MLS を簡略化したもので、単一の機密度を用います。そのため、機密度の高低によるアクセス制御は行われず、機密区分の包含関係によってのみアクセス制御が実施されます。

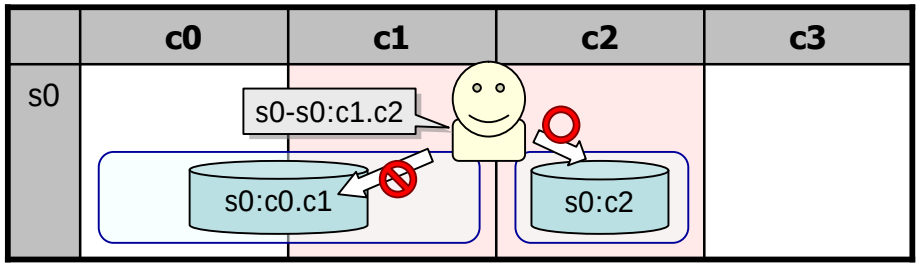


図 13 MCS 模式図

図 13 は MCS によるアクセス制御の模式図です。プロセスの MLS ラベルが「s0-s0:c1.c2」である時、プロセスはピンク色で示された機密区分を支配しています。

プロセスは「s0:c2」のファイルにアクセスすることができます。しかし、「s0:c0.c1」のファイルはプロセスの機密区分に完全に包含されていないため、アクセスすることはできません。

4.5 SELinux のカスタマイズ

4.5.1 Enforcing mode と Permissive mode

SELinux には Enforcing mode と Permissive mode という2つの動作モードがあります。

セキュリティポリシーに基づいた強制アクセス制御が実施されるのは、SELinux が Enforcing mode に設定されている場合だけです。Permissive mode 下でもセキュリティポリシーの評価は行われますが、その結果がアクセス制御に適用されることはありません。アクセス拒否ログを確認してセキュリティポリシーのデバッグを行うために利用します。

Enforcing mode と Permissive mode を切り替えるには setenforce コマンドを利用します。また、システム起動時のモードは/etc/selinux/config によって設定することができます。

4.5.2 条件変数(Boolean)

条件変数(boolean)を用いて、システムの動作中に SELinux セキュリティポリシーの特定箇所の有効/無効を切り替えることができます。条件変数は on 又は off の二値をとり、setsebool コマンドを利用して設定することができます。条件変数の一覧の取得や値の確認には getsebool コマンドを利用します。

個々の条件変数がどのようにセキュリティポリシーをカスタマイズするのかは、セキュリティポリシーでの定義次第です。SE-PostgreSQL に関連した条件変数については 5.2.6「条件変数(boolean)」を参照してください。

4.5.3 semanage コマンド

semanage コマンドを用いて、ログイン時にユーザシェルに関連付けられる MLS ラベルや、mcstrans サービス有効時の MLS ラベルの別名などを設定することができます。詳しくは semanage コマンドの manpage を参照してください。

5 SE-PostgreSQL システム管理

本章では、SE-PostgreSQL を利用するにあたって便利な機能を紹介します。

5.1 SE-PostgreSQL のバックアップとリストア

PostgreSQL のバックアップ/リストア用のユーティリティである pg_dump、pg_dumpall に機能追加を行い、セキュリティコンテキスト情報付きのバックアップ/リストアを可能にしました。

バックアップを行う際には、バックアップの対象となるデータベース/テーブル/カラム/タプルの全てに対して参照を行う権限が必要です。これらの DB オブジェクトに対して必要な権限を持っていない場合、バックアップは失敗するか、アクセス権を持たないタプルが含まれていない不完全な状態となります。

リストアを行う際には、バックアップの場合とは逆に、リストアの対象となるデータベース/テーブル/タプルに対して新規作成又は挿入の権限が必要です。これらの DB オブジェクトに対して必要な権限を持っていない場合、リストアは失敗します。

pg_dump コマンド

--enable-security オプションによって、セキュリティコンテキスト情報付きのバックアップを行うことが可能になりました。

セキュリティコンテキスト情報付きバックアップの対象となるデータベースオブジェクトは、テーブル、カラム、SQL 関数、ラージオブジェクト及びタプルです。

pg_dumpall コマンド

--enable-security オプションによって、セキュリティコンテキスト情報付きのバックアップを行うことが可能になりました。

セキュリティコンテキスト情報付きバックアップの対象となるデータベースオブジェクトは、データベース、テーブル、カラム、SQL 関数、ラージオブジェクト及びタプルです。

pg_restore コマンド

pg_restore コマンドは、pg_dump および pg_dumpall コマンドの—enable-security オプションによって生成された、セキュリティコンテキスト情報付きバックアップからのリストアを行うことが可能です。

5.2 標準のセキュリティポリシー

本節では、標準のセキュリティポリシーについて説明します。

標準のセキュリティポリシーでは、非 SELinux 環境を想定して作成されたクライアント・アプリケーションとの互換性が最大限に配慮された設定が行われています。しかし、テーブルや関数などの DB オブジェクトに明示的にセキュリティコンテキストを割り当てることで、タイプに固有のアクセス制御ポリシーを適用するすることができます。

SELinux のセキュリティポリシーには Targeted と Strict の二種類のポリシーが存在しますが、Targeted ポ

リシーにおける unconfined_t ドメイン及び Strict ポリシーにおける sysadm_t ドメインは、DDL 系 SQL の実行や機密情報へのアクセスが許可されるなど、他のドメインと比較して高い権限を持っています。便宜上、これを管理用ドメインと呼びます。

それ以外の SE-PostgreSQL に接続可能なドメインは、DDL 系 SQL の実行や機密情報へのアクセスに制限が加えられます。便宜上、これを一般ドメインと呼びます。

標準のセキュリティポリシーには、各 DB オブジェクト向けにいくつかのタイプが既に定義済みです。管理用ドメインは DB オブジェクトのセキュリティコンテキストを変更することが可能です。各 DB オブジェクトに適切なタイプを付与することで、DB オブジェクトの特性に応じたアクセス制御を行うことが可能です。

5.2.1 クライアントのドメイン

前述の通り、Targeted ポリシーにおける unconfined_t ドメインと、Strict ポリシーにおける sysadm_t ドメインは、DDL 系 SQL の実行や機密情報のアクセスが許可されるなど、他の SE-PostgreSQL に接続可能なドメインと比べて高い権限を持っています。

データベースの構築やバックアップ/リストアは、これらのドメインから実施する必要があります。

5.2.2 テーブル/カラム/タプルのタイプ

テーブルに付与されたタイプは、そのテーブルに含まれるカラム及びタプルに暗黙的に継承されます。つまり、明示的に変更を行わない限り、カラムおよびタプルのタイプはテーブルのそれと一致します。

標準セキュリティポリシーには、テーブル/カラム/タプル向けに以下の 5 つのタイプが定義済みです。利用目的に応じてこれらのタイプを付与することができます。

sepgsql_table_t

標準セキュリティポリシーにおいて、新しく作成したテーブルに付与されるデフォルトのタイプです。

このタイプを付与されたテーブルに対しては、全てのクライアントがセキュリティコンテキストの変更を除く全ての操作を行うことができます。すなわち、ネイティブの PostgreSQL を利用した時と同じようにテーブル/カラム/タプルを参照することができます。

sepgsql_secret_table_t

管理用ドメインからのアクセス、又は Trusted Procedure を経由したアクセスを除いて、このタイプを付与されたテーブル/カラム/タプルに対するあらゆるアクセスを禁止します。

このタイプは、例えばパスワードやクレジットカード番号などの機密性の高い情報を格納する場合に有用です。

sepgsql_ro_table_t

管理用ドメインからのアクセス、又は Trusted Procedure を経由したアクセスを除いて、このタイプを付与されたテーブル/カラム/タプルに対する SELECT 文以外の DML の実行を禁止します。

このタイプは、例えば商品マスタ表などの読出し専用のデータを改ざんから保護するのに有用です。

sepgsql_fixed_table_t

管理用ドメインからのアクセス、又は Trusted Procedure を経由したアクセスを除いて、このタイプを付与されたテーブル/カラム/タプルに対する SELECT 文及び INSERT 文以外の DML の実行を禁止します。

このタイプの目的は、一度データベースに格納された情報が、その後改ざん･破棄されないことを保障するためのものです。例えば法令等で一定期間の保存が義務付けられているドキュメント情報を保持する際に有用です。

sepgsql_sysobj_t

PostgreSQL において、DB オブジェクトを管理するためのメタ情報はシステムカタログと呼ばれる特殊なテーブルに保存されています。このタイプは、システムカタログ内の各タプルに付与されるデフォルトのタイプです。

sepgsql_enable_users_ddl によって、一般ドメインによるシステムカタログ内の各タプルの更新を許可するか否かを切り替えることができます。

5.2.3 関数のタイプ

sepgsql_proc_t

標準セキュリティポリシーにおいて、管理用ドメインが新しく作成した関数に付与されるデフォルトのタイプです。

全てのクライアントは sepgsql_proc_t タイプを持つ関数を実行することが可能です。このタイプを持った関数の実行によってドメイン遷移が発生することはありません。

sepgsql_user_proc_t

標準セキュリティポリシーにおいて、一般ドメインが新しく作成した関数に付与されるデフォルトのタイプです。

一般ドメインは sepgsql_user_proc_t タイプを持つ関数を実行することが可能ですが、管理用ドメインはこの関数を実行することはできません。これは、一般ドメインによって定義された関数を管理用ドメインの権限で動作させることが、しばしば危険な結果を招くことがあるからです。

管理用ドメインは関数の定義を検証し、sepgsql_proc_t にタイプを変更することでこの関数を実行できるようになります。

sepgsql_trusted_proc_t

このタイプを付与された関数は Trusted Procedure です。全てのクライアントは sepgsql_trusted_proc_t タイプを持つ関数を実行することが可能で、関数の実行によってドメイン遷移が発生します。ドメイン遷移により、管理用ドメインと同じ権限で DB オブジェクトにアクセスすることが可能になります。

Trusted Procedure は、機密情報を保持した DB オブジェクトを参照する方法を限定するのに有用な機能ですが、設定を誤ると SE-PostgreSQL のセキュリティ機能を台無しにするリスクがあることを理解の上で設定するようにしてください。

5.2.4 ラージオブジェクトのタイプ

sepgsql_blob_t

標準セキュリティポリシーにおいて、新しく作成したラージオブジェクトに付与されるデフォルトのタイプです。全てのクライアントは sepgsql_blob_t タイプを持つラージオブジェクトに対して読み書き可能で、ネイティブの PostgreSQL を利用した時と同じようにラージオブジェクトにアクセスすることができます。

sepgsql_ro_blob_t

管理用ドメインからのアクセス、又は Trusted Procedure を経由したアクセスを除いて、sepgsql_ro_blob_t タイプを持つラージオブジェクトへの書き込みを禁止します。

sepgsql_secret_blob_t

管理用ドメインからのアクセス、又は Trusted Procedure を経由したアクセスを除いて、sepgsql_secret_blob_t タイプを持つラージオブジェクトへのアクセスを禁止します。
このタイプは、機密性の高いドキュメントなどのバイナリデータを格納する場合に有用です。

5.2.5 データベースのタイプ

sepgsql_db_t

標準セキュリティポリシーにおいて、データベースに付与される唯一のタイプです。データベースにこれ以外のタイプを付与することは想定されていません。

5.2.6 条件変数(boolean)

SE-PostgreSQL に関連する条件変数(boolean)は以下の4つです。これらを用いて SE-PostgreSQL 向けのセキュリティポリシーをカスタマイズすることができます。

sepgsql_enable_unconfined

管理用ドメインの有効/無効を切り替えます。この条件変数が off の場合、管理用ドメインであっても一般ドメインと同様の権限で動作します。デフォルトは on です。

sepgsql_enable_users_ddl

一般ドメインによる DDL 構文の実行を制御します。この条件変数が off の場合、一般ドメインが CREATE TABLE などの DDL 文を実行することは禁止されます。デフォルトは on です。

sepgsql_enable_auditallow

「アクセス許可ログ」の出力を有効にします。この条件変数が on の場合、アクセス許可ログが出力され、どの DB オブジェクトに対して、どのようなチェックが行われたのかをログによって確認することができます。デフォルトは off です。

sepgsql_enable_auditdeny

「アクセス拒否ログ」の出力を有効にします。この条件変数が on の場合、アクセス拒否ログが出力され、ログを参照することで、セキュリティポリシーに違反する DB オブジェクトのアクセスを確認することができます。デフォルトは on です。

sepgsql_enable_audittuple

タプルに対するアクセス許可/拒否ログ出力の有効/無効を切り替えます。
アクセス許可/拒否ログを出力するために sepgsql_enbale_auditallow 又は sepgsql_enable_auditdeny の設定を on にしている場合でも、タプルに対するログの出力は sepgsql_enable_audittuple の設定に依存します。
これは、タプルの件数が非常に大きなテーブルをスキャンした際に、ログがアクセス拒否/許可ログで溢れないようにするためのものです。デフォルトは off です。

5.3 Labeled IPsec

Labeled IPsec はネットワーク経由で通信しているプロセス間で、通信相手先のセキュリティコンテキストを取得するための技術です。クライアントが TCP/IP 経由で SE-PostgreSQL に接続する場合、クライアントのセキュリティコンテキストを取得するために Labeled IPsec の設定を行うことが必須です。
本節では、SELinux の Labeled IPsec の設定方法を紹介します。

必要なパッケージの確認

以下のパッケージ群がインストールされていることを確認してください。

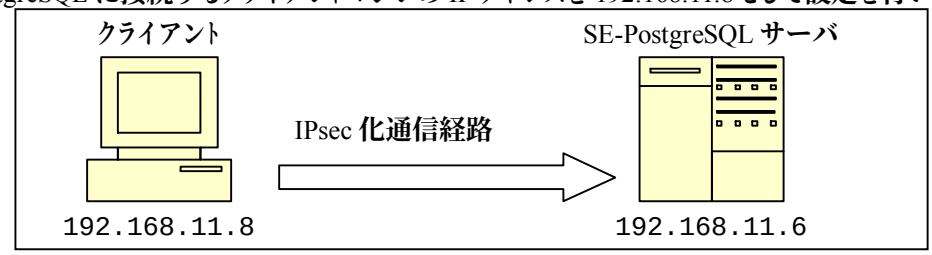
- `kernel` (Labeled Networking 対応版)
- `ipsec-tools-0.6.5-6` 以降

Labeled IPsec 対応版 kernel では、CONFIG_SECURITY_NETWORK_XFRM=y でビルドされている必要があります。Red Hat Enterprise Liux 5 やそのクローンである CentOS 5、及び Fedora core 6 の最近のカーネルでは有効化されています。

Labeled IPsec では、IPsec の鍵交換プロセスを拡張してサーバ/クライアントのセキュリティコンテキストを相手側ホストに送出します。`ipsec-tools` に含まれる鍵交換サーバ `racoon` には、Labeled IPsec 用の拡張が加えられています。

サンプル環境の説明

以下の設定例では、SE-PostgreSQL が動作するサーバ側の IP アドレスを 192.168.11.6、SE-PostgreSQL に接続するクライアントマシンの IP アドレスを 192.168.11.8 として設定を行います。



SPD(Security Policy Database)へのポリシーの追加

クライアント～サーバ間の通信を IPsec で暗号化し、鍵交換プロセス中に互いのセキュリティコンテキストを交換するため、以下の設定を行います。

1. クライアント/サーバの双方で、次の内容の SPD 定義ファイルを作成します。クライアント側では IP アドレスの箇所が逆になることに留意してください。

```
[サーバ側(192.168.11.6)]
spdadd 192.168.11.6 192.168.11.8 any
-ctx 1 1 "system_u:object_r:unlabeled_t:s0"
-P out ipsec
esp/transport//require;

spdadd 192.168.11.8 192.168.11.6 any
-ctx 1 1 "system_u:object_r:unlabeled_t:s0"
-P in ipsec
esp/transport//require;
```

2. setkey コマンドを実行し、SPD 定義ファイルの内容をロードします。

```
# setkey –f <SPD 定義ファイル>
```

/etc/racoon/racoon.conf の設定

鍵交換サーバ `racoon` の設定を行うために、`/etc/racoon/racoon.conf` を編集します。太字で示した箇所への追加設定が必要です。以下の設定はサーバ側(192.168.11.6)の例ですが、クライ

アント側では IP アドレスの箇所が逆になることに留意してください。

```
[サーバ側(192.168.11.6)での設定]
# Racoon IKE daemon configuration file.
# See 'man racoon.conf' for a description of the format and entries.

path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

sainfo anonymous
{
       pfs_group 2;
       lifetime time 1 hour ;
       encryption_algorithm 3des, blowfish 448, rijndael ;
       authentication_algorithm hmac_sha1, hmac_md5 ;
       compression_algorithm deflate ;
}

remote 192.168.11.8
{
       exchange_mode aggressive, main;
       my_identifier address;
       proposal {
              encryption_algorithm 3des;
              hash_algorithm sha1;
              authentication_method pre_shared_key;
              dh_group 2 ;
       }
}
```

/etc/racoon/psk.txt の設定

設定を単純化するため、ここでは事前共有鍵(Pre Shared Key)方式による通信経路の暗号化を行います。事前共有鍵を格納するのは/etc/racoon/psk.txt で、このファイルに相手側ホストのエントリを追加します。以下の設定はサーバ側(192.168.11.6)の例ですが、クライアント側では IP アドレスの箇所が逆になることに留意してください。

```
[サーバ側(192.168.11.6)での設定]
# file for pre-shared keys used for IKE authentication
# format is:  'identifier' 'key'
# For example:
#
#  10.1.1.1           flibbertigibbet
#  www.example.com     12345
#  foo@www.example.com  micropachycephalosaurus
192.168.11.8          somethingsecrettext
```

racoon の起動

以上の設定を加えて racoon を起動します。

```
# racoon
```

これで、クライアント側(192.168.11.8)～サーバ側(192.168.11.6)間の通信が暗号化され、コネクション確立時には racoon によって両端点のセキュリティコンテキストが交換されるようになります。

6 付録

6.1 拡張 SQL 構文

SE-PostgreSQL のセキュリティ機能を活用するため、いくつかの SQL 構文が拡張されています。本節では、これらの拡張 SQL 構文について解説します。

6.1.1 CREATE DATABASE 構文

書式

```
CREATE DATABASE dbname CONTEXT = 'context'
```

説明

PostgreSQL の CREATE DATABASE 構文の拡張によって、明示的にセキュリティコンテキストを指定してデータベースを作成することが可能になりました。

クライアントは明示的に指定したセキュリティコンテキストに対して database:{create}権限を有していなければいけません

例

```
kaigai=# CREATE DATABASE testdb
     CONTEXT = 'system_u:object_r:sepgsql_db_t';
CREATE DATABASE
kaigai=#
```

6.1.2 ALTER DATABSE 構文

書式

```
ALTER DATABASE dbname CONTEXT = 'context'
```

説明

PostgreSQL の ALTER DATABASE 構文の拡張によって、データベースのセキュリティコンテキストを変更することが可能になりました。

クライアントはデータベースに対して database:{setattr relabelfrom}権限を、そして新しく付与するセキュリティコンテキストに対して database:{relabelto}権限を有していなければいけません。

例

```
kaigai=# ALTER DATABASE testdb
     CONTEXT = 'user_u:object_r:sepgsql_db_t:s0:c0';
ALTER DATABASE
kaigai=#
```

注釈

この構文を実行した結果、データベースのセキュリティコンテキストが変化しない場合、database:{relabelfrom relabelto}権限は評価されません。

6.1.3 CREATE TABLE 構文

書式

```
CREATE TABLE tblname (
     colname <TYPE> [<CONSTRAINT>] CONTEXT = 'column context',
       :
) [<table options>] CONTEXT = 'table context'
```

説明

PostgreSQL の CREATE TABLE 構文の拡張によって、明示的にテーブル及びカラムのセキュリティコンテキストを指定してテーブルを作成することが可能になりました。

クライアントは明示的に指定したセキュリティコンテキストに対して、それぞれ table:{create} 又は column:{create} 権限を有していなければいけません。

例

```
kaigai=# create table tbl1 (
   id integer primary key
      context = 'system_u:object_r:sepgsql_table_t',
   body text
      context = 'system_u:object_r:sepgsql_secret_table_t'
) context = 'system_u:object_r:sepgsql_table_t:s0:c0';
CREATE TABLE
kaigai=#
```

6.1.4 ALTER TABLE 構文

書式

```
ALTER TABLE tblname [ALTER colname] CONTEXT = 'context'
```

説明

PostgreSQL の ALTER TABLE 構文の拡張によって、テーブル又はカラムのセキュリティコンテキストを変更することが可能になりました。

クライアントはテーブルに対して table:{setattr relabelfrom} 権限を、指定したセキュリティコンテキストに対して table:{relabelto} 権限を有していなければいけません。

同様に、カラムに対して column:{setattr relabelfrom} 権限を、指定したセキュリティコンテキストに対して column:{relabelto} 権限を有していなければいけません。

例

```
kaigai=# ALTER TABLE drink
       CONTEXT = 'user_u:object_r:sepgsql_secret_table_t';
ALTER TABLE
kaigai=#
```

注釈

この構文を実行した結果、テーブルのセキュリティコンテキストに変化がない場合、table:{relabelfrom relabelto} 権限は評価されません。同様にカラムのセキュリティコンテキストに変化がない場合、column:{relabelto} 権限は評価されません。

6.1.5 CREATE FUNCTION 構文

書式

```
CREATE [OR REPLACE] FUNCTION (<type>,...)
    RETURNS <type> [<options> ...] CONTEXT = 'context'
    [AS '<definition>']
```

説明

PostgreSQL の CREATE FUNCTION 構文の拡張によって、明示的にセキュリティコンテキストを指定して関数を作成することが可能になりました。

クライアントは明示的に指定したセキュリティコンテキストに対して procedure:{create}権限を有していなければいけません。

例

```
kaigai=# create or replace function less_than (integer, integer)
       returns bool language 'sql'
       context = 'system_u:object_r:sepgsql_trusted_proc_t'
       ss 'select $1 < $2';
CREATE FUNCTION
kaigai=#
```

6.1.6 ALTER FUNCTION 構文

書式

```
ALTER FUNCTION funcname CONTEXT = 'context'
```

説明

PostgreSQL の ALTER FUNCTION 構文の拡張によって、関数のセキュリティコンテキストを変更することが可能になりました。

クライアントは関数に対して procedure:{setattr relabelfrom}権限を、そして新しく付与するセキュリティコンテキストに対して procedure:{relabelto}権限を有していなければいけません。

例

```
kaigai=# alter function check_person_passwd(integer, text)
     context = 'user_u:object_r:sepgsql_trusted_proc_t';
ALTER FUNCTION
kaigai=#
```

注釈

この構文を実行した結果、関数のセキュリティコンテキストが変化しない場合、procedure:{relabelfrom relabelto}権限は評価されません。

6.2 拡張 SQL 関数

SE-PostgreSQL のセキュリティ機能を活用するため、いくつかの SQL 関数が追加されています。本節では、これらの拡張 SQL 関数について解説します。

6.2.1 sepgsel_getcon() 関数

定義

```
sepgsql_getcon() returns security_label
```

説明

この関数はクライアントのセキュリティコンテキストを取得するために新たに追加されました。sepgsql_getcon() 関数を実行すると、現在のセキュリティコンテキストを返します。Trusted Procedure の内部から呼び出された場合には、ドメイン遷移を行った先のセキュリティコンテキストを行います。

例

```
kaigai=# select sepgsql_getcon();
              sepgsql_getcon
-------------------------------------------------
 root:system_r:unconfined_t:SystemLow-SystemHigh
(1 row)
```

6.2.2 lo_get_security() 関数

定義

```
lo_get_security(Oid loid) returns security_label
```

説明

この関数はラージオブジェクトのセキュリティコンテキストを取得するために新たに追加されました。lo_get_security()関数を実行すると、 loid で指定したラージオブジェクトのセキュリティコンテキストを返却します。クライアントは、ラージオブジェクトに対して blob:{getattr}権限を有していなければいけません。

例

```
kaigai=# select lo_get_security(16410);
       lo_get_security
-------------------------------
 user_u:object_r:sepgsql_blob_t
(1 row)
```

6.2.3 lo_set_security() 関数

定義

```
lo_set_security(Oid loid, security_label context) returns bool
```

説明

この関数はラージオブジェクトのセキュリティコンテキストを変更するために新たに追加されました。lo_set_security()関数を実行すると、loid で指定したラージオブジェクトのセキュリティコンテキストを context に変更します。クライアントは、ラージオブジェクトに対して blob:{setattr relabelfrom}権限を、そして新しく付与するセキュリティコンテキストに対して blob:{relabelto}権限を有していなければいけません。

例

```
kaigai=# select
lo_set_security(16410,'user_u:object_r:sepgsql_secret_blob_t');
 lo_set_security
-----------------
 t
(1 row)
```