Controle de Acesso

Controle de Acesso em Camadas

Vamos pensar no controle de acesso como um gerenciamento em camadas, desde a conexão de um usuário ao banco no servidor do cluster de dados até o acesso a objetos individuais.

Camada 1: Autenticação de Conexão

Esta camada responde à pergunta mais básica: "Você tem permissão para sequer bater à nossa porta?". O controle é feito exclusivamente pelo arquivo `pg_hba.conf` (Host-Based Authentication), localizado no diretório de dados do seu cluster PostgreSQL.

Como o PostgreSQL Utiliza o pg_hba.conf

O processo é metódico e inflexível:

1. **Tentativa de Conexão**: Um cliente (seja o psql, DBeaver, uma aplicação Java, etc.) tenta estabelecer uma conexão, informando o host, a porta, o nome do usuário (role) e o banco de dados de destino.
2. **Verificação do Processo Backend**: O processo Postmaster no servidor aceita a conexão TCP e cria um processo filho (backend) para servi-la. Este processo filho é o responsável por autenticar o usuário.
3. **Leitura Sequencial**: O processo backend lê o arquivo `pg_hba.conf` de cima para baixo, linha por linha.
4. **A Primeira Regra Vence**: Ele para na primeira linha que corresponde ao tipo de conexão (local ou rede), ao banco de dados solicitado, ao nome do usuário e ao endereço IP de origem. Qualquer regra subsequente que também poderia corresponder é ignorada.
5. **Aplicação do Método**: O método de autenticação especificado na linha correspondente é então executado. Se a autenticação for bem-sucedida, o acesso é concedido. Se falhar, a conexão é encerrada.
6. **Rejeição Padrão**: Se o processo backend ler o arquivo inteiro e nenhuma linha corresponder à tentativa de conexão, o acesso é negado.

Sintaxe e Regras

Cada linha significativa no `pg_hba.conf` é uma regra com colunas fixas:

```
TIPO BANCO_DE_DADOS USUÁRIO ENDEREÇO MÉTODO [OPÇÕES]
```

TIPO (Como você está se conectando?)

- `local`: Conexões via sockets de domínio Unix. Usado apenas para conexões na mesma máquina, sem passar pela pilha de rede. É o método mais comum e seguro para scripts e administradores locais.
- `host`: Conexões via TCP/IP. Cobre todas as conexões de rede, incluindo as de localhost (ex: 127.0.0.1).
- `hostssl`: Conexão TCP/IP que exige que a comunicação seja criptografada com SSL/TLS. A conexão será rejeitada se o cliente não negociar SSL.
- `hostnossl`: O oposto. Rejeita a conexão se ela estiver usando SSL.

BANCO_DE_DADOS e USUÁRIO (Para onde e quem?)

- `all`: Corresponde a qualquer banco de dados ou usuário.
- `sameuser`: Corresponde se o nome do banco de dados for o mesmo do nome do usuário.
- `samerole`: Corresponde se o usuário for membro do role com o mesmo nome do banco de dados.
- `replication`: Uma palavra-chave especial para conexões de replicação.
- `dvdrental`: Um nome específico.

- `@nome_do_arquivo`: Inclui uma lista de nomes a partir de um arquivo externo, útil para gerenciar muitos usuários ou bancos.

ENDEREÇO (De onde você vem?)

- Um endereço IP em notação CIDR (ex: 192.168.1.10/32 para um único host, 192.168.1.0/24 para uma sub-rede).
- `all`: Corresponde a qualquer endereço IP.
- `localhost`: Abreviação para 127.0.0.1/32 e ::1/128.
- `samenet`: Corresponde a qualquer endereço na mesma sub-rede do próprio servidor.

MÉTODO (Como você vai provar sua identidade?)

- `scram-sha-256`: Usa um mecanismo de desafio-resposta que nunca envia a senha pela rede. O cliente prova que conhece a senha sem revelá-la. É a escolha preferencial para o PostgreSQL 13 em diante.
- `md5`: O padrão antigo. Envia a senha pela rede, mas hasheada com MD5. Vulnerável a ataques se a conexão não for protegida com SSL.
- `password`: Envia a senha em texto claro. **NUNCA USE** a menos que a conexão seja obrigatoriamente `hostssl`.
- `peer`: Apenas para conexões local. Obtém o nome do usuário do sistema operacional a partir do kernel e o compara com o nome do usuário do banco de dados solicitado. Seguro para administradores logados na máquina do servidor.
- `trust`: Confia cegamente no usuário. **EXTREMAMENTE PERIGOSO**. Permite a conexão sem qualquer verificação de senha. Use apenas em ambientes de desenvolvimento totalmente isolados e controlados.
- `reject`: Nega explicitamente a conexão. Útil como a última regra "catch-all" para garantir que nada passe despercebido.
- `ldap, pam, cert, gss`: Métodos avançados para integração com sistemas de autenticação corporativos (LDAP, Active Directory, Kerberos) ou autenticação baseada em certificados de cliente SSL.

Exemplo Detalhado de um pg_hba.conf para o dvdrental

```
# ============================================================================
# ARQUIVO DE CONFIGURAÇÃO DE AUTENTICAÇÃO DO POSTGRESQL - pg_hba.conf
#
# A ordem é crucial. A primeira regra que corresponder será usada.
# ============================================================================

# TIPO      BANCO        USUÁRIO         ENDEREÇO            MÉTODO

# -- Regras Locais (Administração no Servidor) --
# Permite que qualquer usuário do sistema se conecte a qualquer banco
# se o nome do usuário do SO for o mesmo do usuário do PG (ex: sudo -u postgres psql)
local   all             all                                     peer

# -- Regras de Rede --
# Requer SSL e senha SCRAM para o superusuário 'postgres' de qualquer lugar.
# Medida de segurança para administração remota.
hostssl all             postgres        0.0.0.0/0           scram-sha-256
hostssl all             postgres        ::/0                scram-sha-256

# Servidor de aplicação principal tem acesso total ao banco dvdrental.
# O IP é fixo para máxima segurança.
hostssl dvdrental       app_dvdrental   192.168.1.100/32    scram-sha-256

# A equipe de BI, em sua própria sub-rede, pode se conectar para relatórios.
# Também exigimos SSL para proteger os dados em trânsito.
hostssl dvdrental       all             192.168.10.0/24     scram-sha-256

# Rejeita explicitamente conexões de um servidor legado que foi desativado.
host    all             all             10.0.5.50/32        reject

# Nega todas as outras tentativas de conexão TCP/IP que não usam SSL.
host    all             all             0.0.0.0/0           reject
host    all             all             ::/0                reject
```

Camada 2: Roles e Seus Atributos (A Identidade e seus

Poderes Inerentes)

Uma vez dentro do portão, o guarda precisa saber quem você é. No PostgreSQL, essa identidade é o Role. Um role é a entidade única para autorização. Ele pode ter atributos que lhe conferem poderes especiais e inerentes, independentemente das permissões granulares.

Formas de Definição de Roles

```
        CREATE ROLE nome_do_role [WITH OPÇÕES];
        É o comando fundamental e mais completo.

        CREATE USER nome_do_usuario [WITH OPÇÕES];
        É um atalho (alias) para CREATE ROLE nome_do_role WITH LOGIN;. A intenção é criar um role que
pode fazer login.

        CREATE GROUP nome_do_grupo [WITH OPÇÕES];
        É um atalho para CREATE ROLE nome_do_role WITH NOLOGIN;. A intenção é criar um role para ser
usado como um grupo de permissões.
```

Atributos Exaustivos de um Role

Cada um desses atributos pode ser definido no `CREATE ROLE` ou modificado depois com `ALTER ROLE`:

- `LOGIN / NOLOGIN`: Define se o role pode iniciar conexões com o banco
- `SUPERUSER / NOSUPERUSER`: Concede poderes de superusuário (ignora verificações de permissão)
- `CREATEDB / NOCREATEDB`: Permite criar novos bancos de dados
- `CREATEROLE / NOCREATEROLE`: Permite gerenciar outros roles (exceto superusuários)
- `INHERIT / NOINHERIT`: Controla herança de permissões (padrão: INHERIT)
- `REPLICATION / NOREPLICATION`: Permite iniciar conexões de replicação
- `BYPASSRLS / NOBYPASSRLS`: Ignora políticas de segurança a nível de linha (RLS)
- `CONNECTION LIMIT N`: Define conexões concorrentes (padrão: -1 = sem limite)
- `PASSWORD 'senha' / PASSWORD NULL`: Define/remove senha do role
- `VALID UNTIL 'timestamp'`: Define data de expiração para senha

Exemplos Detalhados de Criação de Roles

```
        -- 1. Criando um role de aplicação
        CREATE ROLE app_dvdrental WITH
          NOLOGIN           -- O role em si não faz login
          PASSWORD NULL;    -- Não precisa de senha

        -- 2. Criando um usuário de aplicação que fará parte do role acima
        CREATE ROLE app_user WITH
          LOGIN
          PASSWORD 'uma_senha_muito_longa_e_complexa_gerada_aleatoriamente'
          CONNECTION LIMIT 10; -- Limita o pool de conexões da aplicação

        -- 3. Criando um grupo para administradores de banco de dados júnior
        CREATE ROLE junior_dbas WITH
          NOLOGIN;

        -- 4. Criando um DBA júnior que pode criar bancos e roles, mas não é superusuário
        CREATE ROLE maria_dba WITH
          LOGIN
          PASSWORD 'outra_senha_forte'
          CREATEROLE
          CREATEDB;

        -- 5. Criando um role de auditoria que não pode fazer login, mas precisa fazer bypass no RLS
para ver todos os dados
        CREATE ROLE auditor WITH
          NOLOGIN
          BYPASSRLS;
```

Camada 3: Filiação a Roles (A Estrutura de Equipes)

Aqui organizamos nossas identidades em uma hierarquia. Em vez de dar uma chave para cada sala a cada funcionário, damos a chave da "Sala de Finanças" para o grupo "Equipe de Finanças", e então colocamos os funcionários nesse grupo.

Práticas Comuns na Indústria (Modelo de 3 Camadas)

Grandes empresas evitam o caos gerenciando permissões através de uma hierarquia de roles. A abordagem mais escalável é um modelo de 3 camadas:

1. **Roles de Identidade (Pessoas/Aplicações)**: São roles com LOGIN. Representam uma entidade real (ex: ana_silva, app_dvdrental_service). Regra de Ouro: Estes roles nunca recebem permissões GRANT diretamente. Eles apenas recebem filiação a outros roles.
2. **Roles de Acesso (Permissões)**: São roles NOLOGIN com nomes que descrevem uma permissão específica (ex: permission_read_customers, permission_write_payments). É aqui que os comandos GRANT SELECT, INSERT... são aplicados. Eles agrupam um conjunto coeso de privilégios.
3. **Roles Funcionais (Cargos/Equipes)**: São roles NOLOGIN que representam uma função de negócio (ex: job_analyst, job_finance_manager). Estes roles não recebem GRANTs diretos. Em vez disso, eles recebem filiação aos roles de acesso.

O Fluxo de Trabalho

1. **Configuração Inicial**: O DBA define os roles de acesso e funcionais, e conecta os dois.

```
GRANT permission_read_payments TO job_finance_manager;
GRANT permission_read_customers TO job_analyst;
```

2. **Contratação de Novo Analista (Júlio César)**:

```
CREATE ROLE julio_cesar WITH LOGIN PASSWORD '...';
GRANT job_analyst TO julio_cesar;
```

3. **Promoção para Gerente de Finanças**:

```
REVOKE job_analyst FROM julio_cesar;
GRANT job_finance_manager TO julio_cesar;
```

Vantagens

- Escalável, auditável e alinhado ao negócio
- Você nunca precisa mexer nas permissões de baixo nível (GRANT SELECT...) no dia a dia
- A gestão de acesso se resume a gerenciar filiações

Exemplo Prático no dvdrental

```
-- Camada 2: Roles de Acesso (Permissões)
CREATE ROLE dvd_read_permissions NOLOGIN;
CREATE ROLE dvd_payment_write_permissions NOLOGIN;

-- Camada 3: Roles Funcionais (Cargos)
CREATE ROLE analysts NOLOGIN;
CREATE ROLE cashiers NOLOGIN;

-- Conectando Camada 2 e 3
GRANT dvd_read_permissions TO analysts;
GRANT dvd_read_permissions TO cashiers;  -- Caixas também podem ler
GRANT dvd_payment_write_permissions TO cashiers; -- Apenas caixas podem registrar pagamentos

-- Camada 1: Roles de Identidade (Pessoas)
CREATE ROLE ana_silva WITH LOGIN PASSWORD '...';
CREATE ROLE pedro_caixa WITH LOGIN PASSWORD '...';

-- Atribuindo as pessoas às suas funções
GRANT analysts TO ana_silva;
GRANT cashiers TO pedro_caixa;
```

Camada 4: Permissões em Objetos

Com a identidade e a equipe definidas, finalmente respondemos: "O que, exatamente, você pode fazer?".

Exemplo no dvdrental

Cenário: A equipe de analysts (onde está ana_silva) precisa analisar a performance dos filmes

(quantas vezes foram alugados, popularidade por categoria) mas não pode, em hipótese alguma, ver informações pessoais dos clientes ou dados financeiros detalhados.

1. Criar um Schema Dedicado

Sempre isole ambientes de trabalho. Isso previne poluição do schema public e cria uma barreira de segurança clara.

```
CREATE SCHEMA analytics;
```

2. Conceder Permissão de USAGE ao Grupo Funcional

```
-- Usamos o role funcional 'analysts'
GRANT USAGE ON SCHEMA analytics TO analysts;
```

3. Conceder Permissões de Leitura nos Dados Base

Os analistas precisam ler os dados originais para poderem criar seus relatórios. Concedemos isso ao role de permissão.

```
-- Usamos o role de permissão 'dvd_read_permissions'
GRANT USAGE ON SCHEMA public TO dvd_read_permissions;
GRANT SELECT ON public.film, public.film_category, public.category, public.inventory,
public.rental
TO dvd_read_permissions;
```

4. Criar uma VIEW Segura e Agregada no Schema Dedicado

O DBA cria uma view que pré-processa e expõe apenas os dados necessários.

```
CREATE VIEW analytics.vw_film_performance AS
SELECT
    f.title,
    c.name AS category,
    f.rental_rate,
    count(r.rental_id) AS total_rentals
FROM public.film f
JOIN public.film_category fc ON f.film_id = fc.film_id
JOIN public.category c ON fc.category_id = c.category_id
JOIN public.inventory i ON f.film_id = i.film_id
LEFT JOIN public.rental r ON i.inventory_id = r.inventory_id
GROUP BY f.film_id, c.name -- Agrupamos por ID para performance
ORDER BY total_rentals DESC;
```

5. Conceder Permissão de SELECT na VIEW

A permissão de acesso final é na view, não nas tabelas base.

```
GRANT SELECT ON analytics.vw_film_performance TO dvd_read_permissions;
```

6. Automatizar Permissões Futuras

O que acontece se um DBA sênior criar uma nova tabela de sumarização no schema analytics? Para que os analistas a acessem automaticamente, definimos privilégios padrão.

```
-- Qualquer tabela/view nova no schema 'analytics' será automaticamente legível pelo grupo.
ALTER DEFAULT PRIVILEGES IN SCHEMA analytics
GRANT SELECT ON TABLES TO dvd_read_permissions;
```

Princípios Gerais: As Regras de Ouro do DBA

Princípio do Menor Privilégio

Conceda o mínimo absoluto de permissões que um role precisa para fazer seu trabalho. \

- Limita drasticamente o dano em caso de uma conta comprometida (seja por ataque de SQL Injection ou credenciais vazadas).
  Exemplo Prático:

```
-- RUIM:
GRANT SELECT ON customer TO analysts;
```

```
-- CORRETO:
CREATE VIEW vw_customer_region AS ...;
GRANT SELECT ON vw_customer_region TO analysts;
```

Nunca Use SUPERUSER para Aplicações

O role que sua aplicação usa para se conectar ao banco NUNCA deve ser SUPERUSER.

- Um ataque de SQL Injection em uma aplicação rodando como superusuário pode permitir que o atacante leia arquivos do sistema de arquivos do servidor (COPY ... FROM PROGRAM 'cat /etc/passwd'), execute código arbitrário ou apague todo o cluster de dados.
  Exemplo Prático:

```
-- PERIGOSO:
CREATE ROLE app_super WITH LOGIN SUPERUSER;

-- SEGURO:
CREATE ROLE app_normal WITH LOGIN;
GRANT INSERT, UPDATE, SELECT ON tabelas TO grupo_app;
```

Use Roles como Grupos

Adote o modelo de 3 camadas (Identidade, Acesso, Função) descrito acima.

- Gerenciar as permissões de 500 usuários individualmente é impossível. Gerenciar a filiação de 500 usuários a 10 grupos funcionais é trivial.
  Exemplo Prático: O modelo implementado na Camada 3 é o exemplo perfeito.

Schemas e Views são Suas Fronteiras de Segurança

Não coloque todos os objetos no schema public. Use schemas para separar domínios de dados (ex: finance, analytics, hr) e use views para expor subconjuntos seguros desses dados.

- Fornece isolamento lógico e de segurança. Reduz a "superfície de ataque".
  Exemplo Prático: O tutorial da Camada 4, onde criamos o schema analytics e a vw_film_performance.

Exija SCRAM e SSL para Tudo

No pg_hba.conf, use hostssl em vez de host e scram-sha-256 como método para todas as conexões de rede.

- O SSL criptografa todo o tráfego entre o cliente e o servidor. O SCRAM garante que a senha em si nunca trafegue.
  Exemplo Prático: A configuração de pg_hba.conf mostrada na Camada 1 é o exemplo a ser seguido.

Audite Suas Permissões Regularmente

Não confie que as permissões estão corretas; verifique. Periodicamente, revise quem tem acesso a quê.

- "Desvios de permissão" acontecem. Acessos temporários que se tornam permanentes, roles de funcionários que saíram da empresa que não foram removidos.
  Exemplo Prático: Use os comandos do psql e queries no catálogo do sistema:

```
\du: Lista todos os roles e suas filiações a grupos.
\z ou \dp: Lista as permissões de tabelas e views no schema atual.
SELECT * FROM information_schema.role_table_grants WHERE grantee = 'analysts';
```

Separe a Propriedade dos Objetos do Uso

Os objetos do banco (tabelas, views) devem ser propriedade de um role NOLOGIN (ex: schema_owner). A aplicação se conecta com um role diferente que recebe permissões GRANT.

- O proprietário de um objeto pode fazer DROP nele. Se sua aplicação se conecta como

proprietária, um ataque de SQL Injection pode apagar tabelas.
Exemplo Prático:

```
CREATE ROLE dvd_owner NOLOGIN;
ALTER TABLE public.film OWNER TO dvd_owner;
-- A aplicação se conecta como:
GRANT SELECT, INSERT ON public.film TO app_dvdrental;
```