PostgreSQL Security Guide: Best Practices and Implementation

Authentication Methods

Password Authentication

```
# postgresql.confpassword_encryption = scram-sha-256  # More secure than MD5# pg_hba.conf#
TYPE  DATABASE    USER    ADDRESS         METHOD
host    all       all    192.168.1.0/24   scram-sha-256
```

Client Certificate Authentication

```
# Generate server certificate
openssl req -new -x509 -days 365 -nodes -text -out server.crt \  -keyout server.key -subj "/CN=dbhost.yourdomain.com"
# postgresql.conf
ssl = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
```

LDAP Integration

```
# pg_hba.conf
host all all 0.0.0.0/0 ldap ldapserver=ldap.example.com ldapprefix="cn=" ldapsuffix=", dc=example, dc=com"
```

Access Control

```
Role-Based Access Control (RBAC)

-- Create roles with specific privileges

CREATE ROLE readonly LOGIN PASSWORD 'secure_password';

GRANT CONNECT ON DATABASE your_database TO readonly;

GRANT USAGE ON SCHEMA public TO readonly;

GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly;
```

```
-- Create admin role

CREATE ROLE db_admin LOGIN PASSWORD 'admin_password';

GRANT ALL PRIVILEGES ON DATABASE your_database TO db_admin;

-- Create application role

CREATE ROLE app_user LOGIN PASSWORD 'app_password';

GRANT CONNECT ON DATABASE your_database TO app_user;

GRANT USAGE, CREATE ON SCHEMA public TO app_user;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO app_user;
```

Row-Level Security (RLS)

```
-- Enable RLS on a table
ALTER TABLE customer_data ENABLE ROW LEVEL SECURITY;

-- Create policy for accessing own data
CREATE POLICY customer_data_access ON customer_data
   FOR ALL
   TO authenticated_users
   USING (user_id = current_user_id());

-- Create policy for admin access
CREATE POLICY admin_access ON customer_data
   FOR ALL
   TO admin_role
   USING (true);
```

Encryption

Data at Rest

```
-- Enable encryption for specific columns

CREATE EXTENSION pgcrypto;



-- Create table with encrypted columns

CREATE TABLE sensitive_data (

   id SERIAL PRIMARY KEY,

   plain_text TEXT,

   encrypted_text TEXT GENERATED ALWAYS AS (
    encode(
        pgp_sym_encrypt(
            plain_text::text,
            current_setting('app.encryption_key')
        ),
        'base64'
    )
  ) STORED
);


-- Set encryption key
ALTER SYSTEM SET app.encryption_key = 'your-secure-key';
```

SSL/TLS Configuration

```
# postgresql.conf
ssl = onssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'ssl_ciphers = 'HIGH:!aNULL:!MD5'
```

Network Security

Firewall Configuration

```
# pg_hba.conf# Allow specific IP rangeshost    all       all       10.0.0.0/8        scram-sha-256
host    all       all       172.16.0.0/12     scram-sha-256
host    all       all       192.168.0.0/16    scram-sha-256
# Block all other connections
host    all       all       0.0.0.0/0         reject
```

Connection Settings

```
# postgresql.conf
listen_addresses = 'localhost'        # Only listen on localhost
max_connections = 100                 # Limit concurrent connections
authentication_timeout = 1min         # Timeout for authentication
```

Auditing and Monitoring

Audit Logging

```
-- Enable audit logging

CREATE EXTENSION pgaudit;

-- Configure audit logging in postgresql.conf

pgaudit.log = 'write,ddl'

pgaudit.log_catalog = on

pgaudit.log_client = on

pgaudit.log_level = log

pgaudit.log_statement = on

-- Create audit log table

CREATE TABLE audit_log (

   id SERIAL PRIMARY KEY,

   timestamp TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

   user_name TEXT,

   database_name TEXT,

   action TEXT,

   object_type TEXT,

   object_name TEXT,

   query TEXT

);
```

Security Monitoring

```
-- Monitor failed login attempts
SELECT
   application_name,
   client_addr,
   count(*) as failed_attempts
FROM pg_stat_activity
WHERE state = 'active'
AND query LIKE '%failed%login%'
GROUP BY application_name, client_addr;

-- Monitor user activities
SELECT
   usename,
   client_addr,
    count(*) as activity_count,
    max(backend_start) as last_connection
FROM pg_stat_activity
GROUP BY usename, client_addr
ORDER BY activity_count DESC;
```

Security Best Practices

1. Password Policies

```
-- Create password check function

CREATE OR REPLACE FUNCTION check_password_strength(username TEXT, password TEXT)

RETURNS BOOLEAN AS $$

BEGIN

    -- Check password length

    IF length(password) < 12 THEN

        RAISE EXCEPTION 'Password must be at least 12 characters';

    END IF;

    -- Check for complexity

    IF NOT (password ~ '[A-Z]' AND

            password ~ '[a-z]' AND

            password ~ '[0-9]' AND

            password ~ '[^a-zA-Z0-9]') THEN

        RAISE EXCEPTION 'Password must contain uppercase, lowercase, numbers, and special characters';

    END IF;

-- Prevent username in password

    IF password ILIKE '%' || username || '%' THEN

        RAISE EXCEPTION 'Password cannot contain username';

    END IF;

    RETURN true;

END;

$$ LANGUAGE plpgsql;
```

2. Regular Security Updates

```
# Keep PostgreSQL updated

sudo apt update

sudo apt upgrade postgresql

# Check for security advisories

https://www.postgresql.org/support/security/
```

3. Backup Encryption

```
# Encrypt backups using GPG
pg_dump dbname | gpg -c > backup.sql.gpg


# Decrypt backups
gpg -d backup.sql.gpg | psql dbname
```

Security Checklist

Authentication

- Use strong password encryption (SCRAM-SHA-256)
- Implement client certificate authentication
- Configure LDAP integration if needed

Access Control

- Implement role-based access control
- Enable row-level security where needed
- Regular permission audits

Encryption

- Enable SSL/TLS

- Encrypt sensitive columns
- Secure connection strings

.

Network Security

.

- Configure firewall rules
- Limit listening addresses
- Set connection timeouts

.

Monitoring

.

- Enable audit logging
- Monitor failed login attempts
- Regular security scans