WHITEPAPER

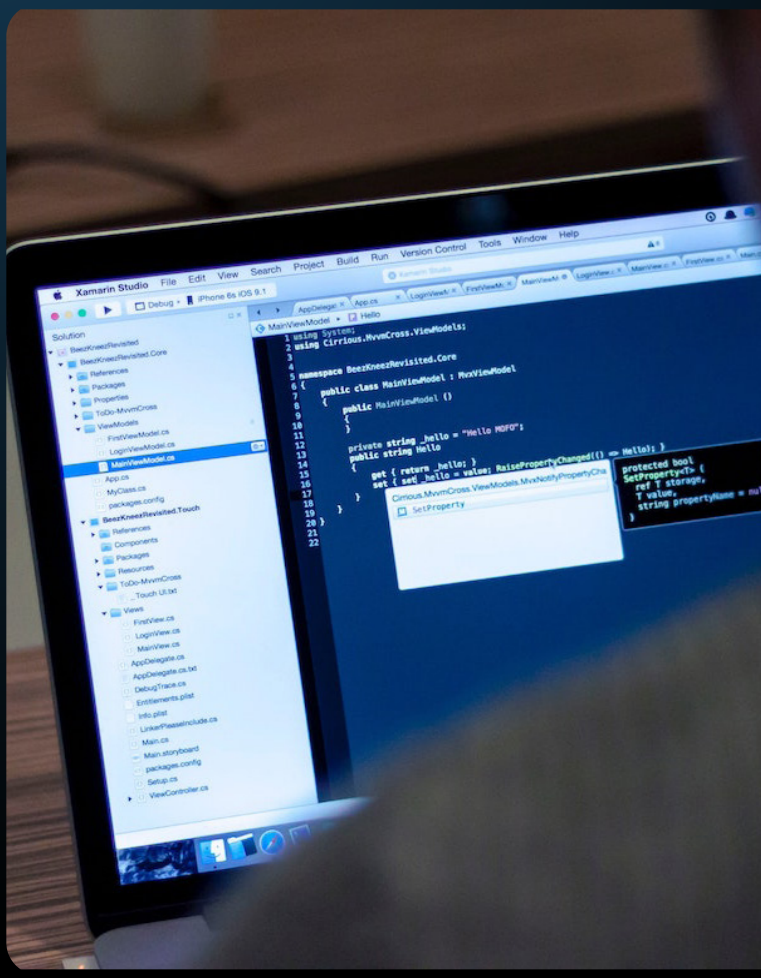
Security Best Practices for PostgreSQL

Contents

1

Executive Summary

This white paper presents a framework and a series of recommendations to secure and protect a PostgreSQL database. We discuss a layered security model that addresses physical security, network security, host access control, database access management, and data security. While all of these aspects are equally important, the paper focuses on PostgreSQL specific aspects of securing the database and the data. For our discussion of the specific security aspects relating to the database and the data managed in the database, we use an AAA (Authentication, Authorization, and Auditing) approach common to computer and network security.



Most of the recommendations in this paper are applicable to PostgreSQL (the community edition) and to EDB Postgres Advanced Server (Advanced Server), the enterprise-class, feature-rich commercial distribution of PostgreSQL from EnterpriseDB® (EDB). Advanced Server provides additional relevant security enhancements, such as Password Profiles, Auditing, Data Redaction, and SQL Server Injection Protection that are not available in the same form in PostgreSQL.

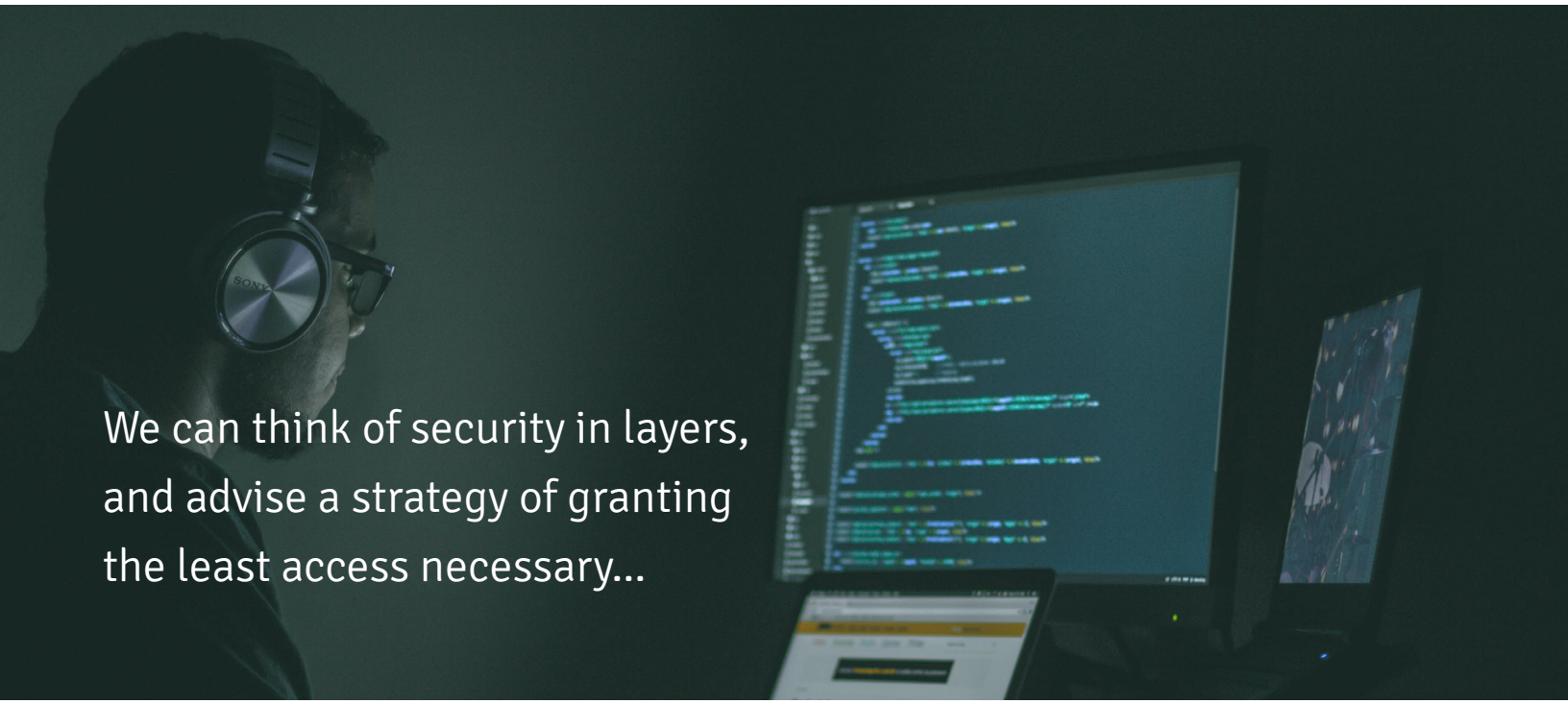
This document has been updated for PostgreSQL 12 and EDB Postgres Advanced Server 12.

2

Introduction

We can think of security in layers, and advise a strategy of granting the least access necessary for any job or role, blocking unnecessary access at the earliest opportunity.

- First is to secure physical access to the host
- Next is to limit access to your corporate network in general
- Next is to limit access to the database host
- Next is to limit access to the database application
- Next is to limit access to the data contained within
- Next is to secure the data stored within



General Recommendations

- **Keep** your operating system and your database patched. EDB's support subscriptions provide timely notifications of security updates and appropriate patches for Postgres. There are a variety of tools available for monitoring for operating system upgrades that can integrate with package management systems such as yum/dnf or apt.
- **Don't put** a postmaster port on the internet, unless it is truly vital to your business. Firewall this port appropriately; if that's not possible, make a read-only standby database available on the port, instead of a read-write master. Network port forwarding with auditing of all connections is a valid alternative.
- **Isolate the database** port from other network traffic through subnetting or other techniques.
- **Grant users** the minimum access they require to do their work, nothing more; reserve the use of superuser accounts for tasks or roles where it is absolutely required.
- **Restrict access** to configuration files (postgresql.conf and pg_hba.conf) and log files (pg_log) to administrators.
- **Disallow host system login** by the database superuser roles (postgres on PostgreSQL, enterprisedb on EDB Postgres Advanced Server). Enable superuser access only as required, in exceptional circumstances.
- **Provide each user** with their own login; shared credentials are not a recommended practice and they make auditing more complicated. Alternatively, use the edb_audit_tag capability (available in EDB Postgres Advanced Server only) to allow applications to add more audit information to sessions resulting from application-level connections.
- **Don't rely** solely on your front-end application to prevent unauthorized access to your database; integrate database security with enterprise level authentication and authorization models, such as LDAP/AD or Kerberos.
- **Keep backups**, and have a tested recovery plan. No matter how well you secure your system, it is still possible for an intruder to get in and delete or modify your data. Ensure your backups are kept securely as well, to prevent unauthorised access.

It may be helpful to think of security in terms of the AAA model developed for network and computer security. AAA stands for Authentication, Authorization, and Auditing.

- **Authentication:** verify that the user is who they claim to be.
- **Authorization:** verify that the user is allowed access.
- **Auditing (or Accounting):** record all database activity, including the user name and the time in the log files.

Not all features fit neatly into these categories, but the AAA model offers a useful framework for this discussion.

3

Applying PostgreSQL Security Features to the AAA Framework.

The following sections provide detailed outlines of how to add PostgreSQL security features to the AAA Framework.

3.1 Authentication

The pg_hba.conf (PostgreSQL host-based access) file restricts access based on user name, database, and source IP (if the user is connecting via TCP/IP). Authentication methods are assigned in this file as well. The authentication method (or methods) you choose depends on your use case.

Kerberos/GSSAPI — PostgreSQL supports GSSAPI with Kerberos authentication according to RFC 1964. GSSAPI provides automatic authentication (single sign-on) for systems that support it. The authentication itself is secure, but data sent over the database connection is unencrypted unless GSS or SSL encryption is in use.

SSPI — Use this if you are on a Windows system and would like to implement Single Sign-On (SSO) authentication.

LDAP should only be used if Kerberos (which includes both SSPI and GSSAPI) are out of the question. LDAP is less secure because passwords are forwarded to the LDAP server, and it can easily be set up in an insecure way.

LDAP and RADIUS — LDAP and RADIUS are useful in situations where you have large numbers of users and need to manage passwords from a central location. This centralization has the advantage of keeping your pg_hba.conf file small and more manageable, and gives your users a "unified password experience" across your infrastructure. Both LDAP and RADIUS require solid infrastructure, as you are relying on the service and connectivity to that service to access your database.

RADIUS — should not be used because it has weak encryption, using md5 hashing for credentials.

Cert — TLS certificate authentication (sometimes referred to as SSL) can be used for encryption of the traffic on the wire and for authentication. Certificates are often used in machine-to-machine communication.

md5 — md5 stores username and password information in the database, which may be a suitable alternative if you have a very small number of users. Scram is highly preferred over md5 as the passwords are securely hashed.

Scram — If you have a very small number of trusted users, you may want to use scram-sha-256 authentication. Scram is highly preferred over md5 as the passwords are securely hashed.

Reject — Use this method to reject specific users, connections to specific databases, and/or specific source IPs.

Trust — trust authentication should only be used in exceptional circumstances, if at all, as it allows a matching client to connect to the server with no further authentication.

It's imperative that you have a full understanding of the ramifications of each authentication method. See the PostgreSQL documentation for a more detailed study of these and other authentication methods.

As mentioned in the Introduction, access to the pg_hba.conf file should be restricted to administrators. Try to keep this file properly pruned; larger, more complicated files are harder to maintain and more likely to contain incorrect or outdated entries. Review this file periodically for unnecessary entries.

3.2 Password Profiles

Starting with version 9.5, Advanced Server supports Oracle-compatible password profiles when using MD5 or SCRAM authentication. A password profile is a named set of password attributes that allow a DBA to easily manage a group of roles that share comparable authentication requirements. Each profile can be associated with one or more users. When a user connects to the server, the server enforces the profile that is associated with the login role.

See Section 2.3 "Profile Management" of EDB's Database Compatibility for Oracle® Developer's Guide for more information, available here.

Profiles can be used to:

- Specify the number of allowable failed login attempts.
- Lock an account due to excessive failed login attempts.
- Mark a password for expiration.
- Define a grace period after a password expiration.
- Define rules for password complexity.
- Define rules that limit password reuse.

3.3 Authorization

Once the user has been properly authenticated, you must grant permissions to view data and perform work in the database. As previously advised, grant only those privileges required for a user to perform a job and disallow shared (group) login credentials. Manage users and groups in PostgreSQL via role assignments. A role may refer to an individual user or a group of users. In Postgres, roles are created at the cluster (database server) level. This means roles are applied to all databases defined for the cluster/database server; it is very important to limit role permissions appropriately. Permissions can be applied to database objects (tables, views, functions, etc), to rows inside of tables, and to redaction policies.

3.3.1 – Access to database objects

Assigned privileges and caveats are outlined in the PostgreSQL CREATE ROLE documentation:

- Revoke CREATE privileges from all users and grant them back to trusted users only.
- Don't allow the use of functions or triggers written in untrusted procedural languages.
- SECURITY DEFINER functions allow users to run functions at an elevated privilege level in a controlled way, but a carelessly written function can inadvertently reduce security. Review the documentation (section Writing Security Definer Functions Safely of CREATE FUNCTION) for more details.
- Database objects should be owned by a secure role, ideally one with very restricted access to the database (e.g. from a Unix Domain Socket only), and not by a role that an application user can connect with. This minimizes the chance that an attacker can modify or drop objects. Whilst this is preferred from a security perspective, it may be problematic with application frameworks that manage the schema themselves - such functionality should be implemented with caution.

Be aware that when log_statement is set to 'ddl' or higher, changing a role's password via the ALTER ROLE command will result in password exposure in the logs, except in EDB Postgres Advanced Server 11 and higher, where the edb_filter_log.redact_password_command instructs the server to redact stored passwords from the log file. Click here for more details.

When authentication information (e.g., usernames and passwords) is stored in a table, use of statement logging can expose that information, even if the table is nominally secure. Similarly, if sensitive information is used in queries (for example any kind of personally identifiable information as a key); those parameters can be exposed by statement logging.

3.3.2 – Views

Access to views can be controlled as described above (they are database objects), and views can in turn be used to limit visibility of data to certain groups of users by creating a VIEW of a table and limiting permissions for that VIEW. PostgreSQL versions 9.2 and higher provide the option to CREATE VIEW WITH (security_barrier), if extra precaution is deemed necessary to avoid possible security issues such as the one described by Robert Haas.

3.3.3 – Row Level Security

PostgreSQL introduced Row Level Security (RLS) in version 9.5. RLS allows fine-grained access to table rows based on the current user role. This includes SELECT, UPDATE, DELETE, and INSERT operations. See more information here.

EDB Postgres Advanced Server includes an Oracle-compatible implementation of this mechanism in its DBMS_RLS package, which includes Oracle compatible implementations of ADD_POLICY, DROP_POLICY, and UPDATE_POLICY. For more information, click here.

3.3.4 – Data Redaction

Data redaction - the ability to hide some data elements or selectively obfuscate data for certain groups of users is another technique to manage access to data. EDB Postgres Advanced Server introduced data redaction in version 11.

Data redaction is a policy-based tool that works with PostgreSQL roles to grant or revoke read access to certain data elements. For example, one group of users sees social security numbers as XXX-XX-1235, whereas data admin role members see the full detail. Here is additional information about data redaction.

Constant	Type	Value	Description
NONE	INTEGER	0	No redaction, zero effect on the result of a query against table.
FULL	INTEGER	1	Full redaction, redacts full values of the column data.
PARTIAL	INTEGER	2	Partial redaction, redacts a portion of the column data.
RANDOM	INTEGER	4	Random redaction, each query results in a different random value depending on the datatype of the column.
REGEXP	INTEGER	5	Regular Expression based redaction, searches for the pattern of data to redact.
CUSTOM	INTEGER	99	Custom redaction type.

3.13.1 Using DBMS_REDACT Constants and Function Parameters

3.4 Auditing

Advanced Server provides the capability to produce audit reports. Database auditing allows database administrators, auditors, and operators to track and analyze database activities in support of complex auditing requirements. These audited activities include database access and usage along with data creation, change, or deletion. The auditing system is based on configuration parameters defined in the configuration file.

We recommend that you audit, (listed by increasing the level of scrutiny):

- User connections
- DDL changes
- Data changes
- Data views

Highly detailed levels of scrutiny can result in a lot of log messages; log only at the level you need. With Postgres, you can adjust logging levels on a per-user and per-database basis. Review your audit logs frequently for anomalous behavior. Establish a chain of custody for your logs.

Keep in mind that a high logging level, combined with storage of passwords in the database, can result in passwords being displayed in the logs. EDB Postgres Advanced Server has introduced the edb_filter_log.redact_password_commands extension in version 11 to instruct the server to redact stored passwords from the audit log file.

Here, see more information on Advanced Server's audit log capability.

> Advanced Server allows database and security administrators, auditors, and operators to track and analyze database activities using the EDB Audit Logging functionality.

3.5 Data Encryption

PostgreSQL offers encryption at several levels, and provides flexibility in protecting data from disclosure due to database server theft, unscrupulous administrators, and insecure networks:

- User connections
- DDL changes
- Data changes
- Data views

You can read more about these options in the PostgreSQL documentation.

If you are concerned about data being sniffed during transfer between a client and the database, enable SSL in the postgresql.conf file unless you can be certain that data sniffing is not a risk. While SSL encryption can add some overhead and certificate management can be tricky, in general this is a recommended practice.

You can also encrypt data within the database, or at the filesystem level (one or the other). See more about Transparent Data Encryption on EDB's blog. With this encryption option, the data is decrypted as it is read from the filesystem, so DBAs can view data; it's imperative to have roles and privileges locked down. Other options include the use of Thales Vormetric Transparent Encryption (VTE).

Use the pgcrypto contrib module to encrypt data on a per-column basis. There are a few drawbacks to this method:

- There's a potential performance hit, depending on the size of the table.
- The encrypted fields can't be searched or indexed.
- The encryption must be applied at table creation time, requiring advanced planning.
- Encryption key management can also be complex.

Additionally, your application must handle the encryption/decryption so that each exchange with the database remains encrypted to prevent an unscrupulous DBA from viewing data.

3.6 SQL Injection Attacks

A SQL injection attack is an attempt to compromise a database by running SQL statements that provide clues to the attacker as to the content, structure, or security of the database. Preventing a SQL injection attack is normally the responsibility of the application developer. Database administrators typically have little or no control over the potential threat.

The standard method to prevent SQL injection attacks in PostgreSQL is to use parameterized queries. If you are using EDB Postgres Advanced Server, we recommend you use the SQL/Protect module to protect against SQL injection attacks. SQL/Protect provides a layer of security in addition to the normal database security policies by examining incoming queries for common SQL profiles. SQL/Protect gives control back to the database administrator by alerting the administrator to potentially dangerous queries and by blocking these queries. For more information, click here.

```
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
```

4.1.2 Configuring SQL/Protect

4

Further Reading

- EDB Security Technical Implementation Guidelines (STIG) for PostgreSQL on Windows and Linux
- Blog: How to Secure PostgreSQL: Security Hardening Best Practices & Tips
- Blog: Managing Roles with Password Profiles: Part 1
- Blog: Managing Roles with Password Profiles: Part 2
- Blog: Managing Roles with Password Profiles: Part 3

About EDB

PostgreSQL is increasingly the database of choice for organizations looking to boost innovation and accelerate business. EDB's enterprise-class software extends PostgreSQL, helping our customers get the most out of it both on premises and in the cloud. And our 24x7 global support, professional services, and training help our customers control risk, manage costs, and scale efficiently.

With 16 offices worldwide, EDB serves over 4,000 customers, including leading financial services, government, media and communications, and information technology organizations. To learn about PostgreSQL for people, teams, and enterprises, visit EDBpostgres.com.

WHITEPAPER

Security Best Practices for PostgreSQL

EnterpriseDB Corporation
34 Crosby Drive
Suite 201
Bedford, MA 01730

EDB™

EDBPOSTGRES.COM