EDB™

Security Best Practices for Postgres

2023 Update

POWER TO POSTGRES

CONTENTS

Executive Summary

This document presents a framework and a series of recommendations to secure and protect a Postgres database. We discuss a layered security model that addresses physical security, network security, host access control, database access management and data encryption. While all of these aspects are equally important, the document focuses on Postgres-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 document are applicable to PostgreSQL (the Community edition) and to EDB Postgres™ Advanced Server (EPAS), the enterprise-class, feature-rich commercial distribution of Postgres from EnterpriseDB® (EDB™). EPAS provides additional relevant security enhancements such as Transparent Data Encryption, password profiles, auditing, data redaction and server-based SQL injection protection that are not available in the same form in PostgreSQL.

EDB also provides our **Standard Plan**, which offers open source Postgres with enterprise-grade tools for high availability, migration, monitoring and backup and recovery. This solution helps businesses strengthen and extend PostgreSQL with enhanced security, resiliency, reliability and optimization. For those looking to protect their data on business-critical applications via transparent dataencryption, the **EDB Standard Plan** is ideal.

2. Introduction

Introduction

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

- Secure physical access to the host
- Limit access to the database application
- Limit access to your corporate network in general
- Limit access to the data contained within the database
- Limit access to the database host
- Secure the data stored within the database

In this document, we will discuss the last three items: limiting access to the database, limiting access to the data and securing the data. While physical, network and host system security are extremely important to the security of your data, they are beyond the scope of this document.

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 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 database ports from other traffic through subnetting or other techniques.
- Grant users the minimum access they require to do their work; 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 for community Postgres, enterprisedb on EDB Postgres Advanced Server). Enable superuser access only as required, in exceptional circumstances.
- Provide users 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 add more audit information to sessions resulting from application-level connections.
- Do not 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 access, delete and modify your data. Ensure your backups are kept securely to prevent unauthorized 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 security features such as Data Encryption (Section 3.5) fit exactly into these categories, but the AAA model offers a useful framework for this discussion.

3. Overcoming PostgreSQL HA limitations with EDB Postgres Distributed

Postgres Security Features – AAA Framework

3.1 Authentications

The Postgres host-based access file (pg_hba.conf) 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 and/or methods you choose depend on your use case.

GSSAPI—Postgres 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 option if you are on a Windows-based system and would like to implement Single Sign-On (SSO) authentication.

LDAP/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.

LDAP should only be used if Kerberos (which includes both SSPI and GSSAPI) is 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. 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 Postgres **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.

3.1.1 Password Profiles

Since Postgres version 9.5, EPAS 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.

Password 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.

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

3.2 Authorization

Once the user has been properly authenticated, you must grant permission 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 Postgres 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.

3.2.1 Database Object Access

Assigned privileges and caveats are outlined in the Postgres documentation:

- Revoke CREATE privileges from all users and grant them back to trusted users only.
- Do not allow functions or triggers written in untrusted procedural languages.
- SECURITY DEFINER functions allow users to run tasks 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. While this is preferred from a security perspective, it may be problematic with application frameworks that manage the schema themselves - this 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. See more information **here**.

When authentication information (e.g., usernames and passwords) are stored in a table, the 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.

3.2.2 View Access

Access to views can be controlled as described above (they are database objects), and views can in turn be used to limit the visibility of data to certain groups of users by creating a VIEW of a table and limiting permissions for that VIEW. Postgres versions 9.2 and higher provide the option to CREATE VIEW WITH (security_barrier). If extra precaution is necessary to avoid possible security issues, see more information **here**.

3.2.3 Row Level Security

Row Level Security (RLS) has been available since Postgres 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 provides for Oracle-compatible implementations of ADD_POLICY, DROP_POLICY and UPDATE_POLICY. See more information **here**.

3.2.4 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 Postgres 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. See more information **here**.

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.

Using DBMS_REDACT Constants and Function Parameters

3.3 Auditing

EPAS 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 the 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.

For more information about EPAS' audit log capability, click **here**.

4. Data Encryption

Data Encryption

Postgres offers various encryption options at several different levels and provides flexibility in protecting data from disclosure due to database server theft, unscrupulous administrators and insecure networks.

Some of these encryption options include:

- Transparent Data Encryption
- Full Disk Encryption
- File System Encryption
- Column Level Encryption
- Password Storage Encryption
- Data Partition Encryption
- Network-based Password Encryption
- Network-based Data Encryption
- Client-side Encryption

Some of these encryption options are highlighted below.

4.1 Transparent Data Encryption (TDE)

One of the building blocks of database encryption is TDE. TDE offers encryption at the file level which solves the problem of protecting data at rest, encrypting databases both on the hard drive and consequently on backup media. Enterprises typically employ TDE to solve compliance issues such as PCI DSS which require the protection of data at rest. TDE is an optional feature supported by version 15 of EDB Postgres Advanced Server, EDB Postgres Extended Server with high availability and the **EDB Standard Plan**.

In addition, TDE also helps protect private and confidential information by encrypting it so that it cannot be read by anyone without the authority to see it. It uses key management to control who has access to what data. Keys are essentially decoder rings, which allow for encrypted data to be unlocked with a unique key.

Community Postgres does not have TDE, making it a difficult choice for regulated industries and governmental agencies that require PCI compliance. The introduction of TDE to Postgres makes it a much more viable option for these types of customers.

What is encrypted with TDE?

TDE encrypts the following:
Files underlying tables, sequences and indexes, including TOAST tables and system catalogs—including all forks. These files are known as data files.
Write-ahead log (WAL) files
Temporary files for query processing and database system operation

The following items are not encrypted:
Metadata internal to operating the database system that doesn't contain user data, such as the transaction status (for example, pg_subtrans and pg_xact).
The file names and file system structure are in the data directory. That means that the overall size of the database system, the number of databases, the number of tables, their relative sizes, as well as file system metadata such as last access time are all visible without decryption.
Data in foreign tables, server diagnostic logs, configuration files, etc.

How does TDE work?

TDE prevents unauthorized viewing of data in operating system files on the database server and on backup storage. Data becomes unintelligible for unauthorized users if it's stolen or misplaced.

Data encryption and decryption are managed by the database and do not require application changes or updated client drivers.

EDB Postgres Advanced Server and EDB Postgres Extended Server provide hooks to key management that's external to the database. These hooks allow for simple passphrase encrypt/decrypt or integration with enterprise key management solutions.

For more information on Securing the data encryption key, **click here**.

4.2 Full Disk Encryption

Full disk or partition encryption is one of the best ways of protecting your data. This method not only protects each file, however, also protects the temporary storage that may contain parts of these files. Full disk encryption protects all files so you do not have to worry about selecting what you want to protect and possibly missing a file.

There are different encryption options available bundled with most operating systems, commercially and as open-source products. Among the most common options are **FileVault**, which is included with Apple macOS, **BitLocker** for use with Microsoft Windows and Linux Unified Key Setup **LUKS** on Linux systems.

Encrypted volumes are also available on all the major cloud providers for protecting your data. For example, Amazon's Elastic Block Service (EBS) provides an option for creating encrypted volumes, which can use a default key or one provided through their key management system. It's worth noting that Amazon does of course have access to both your keys and the physical devices on which the volumes are provisioned, but they go to lengths to ensure that there is a separation of duties between the staff that may have access to the keys and staff that may have access to the hardware.

4.3 File System Encryption

File system encryption, often called file/directory encryption, is where individual files or directories are encrypted by the file system itself. There is stackable cryptographic file system encryption available which users can utilize in their environment.

File system encryption gives the following advantages:

1. Flexible file-based key management, so that each file can be and usually is encrypted with a separate encryption key.
2. Individual management of encrypted files e.g. Incremental backups of the individual changed files even in encrypted form, rather than backup of the entire encrypted volume.
3. Access control can be enforced through the use of public-key cryptography, and the fact that cryptographic keys are only held in memory while the file that is decrypted by them is held open.

When using file system encryption, we typically encrypt the volumes that are used to store the database and write-ahead log, or often the entire system. These types of encryption are transparent to the database server and require no configuration in Postgres.

It is important to note that file system or full disk encryption in Postgres provides protection against different attack vectors. The operating system may make use of a password or key management system very early on in the boot phase to ensure that keys are kept externally, but once a server with file system encryption is booted and running with filesystems mounted, all the data is accessible in the same way as it would be on a machine without encryption. This gives us protection against physical attacks on non-running hardware; for example, a stolen hard disk. File system or full disk encryption does not protect against attacks on a system that is up and running, nor do they enable us to control the visibility of data in the database for different users.

5. Conclusion

Conclusion

Techniques for securing and minimizing access to sensitive data in Postgres require careful planning and design, but can significantly improve the overall security of your data. Postgres offers a rich set of tools to secure the database and control access to the data. The AAA (Authentication, Authorization, and Auditing) framework provides a common method on how to leverage and most effectively utilize system tools. EDB Postgres Advanced Server adds additional capabilities in terms of Password Profiles, Data Redaction, and SQL Server Injection Protection.

We've also covered how determining user authentication mechanisms to authenticate different connection attempts is critical to securing your Postgres deployment. Roles are an important part of security in Postgres, and by configuring and securing them properly, we can use them to minimize the risk to your database servers using the principle of least privilege.

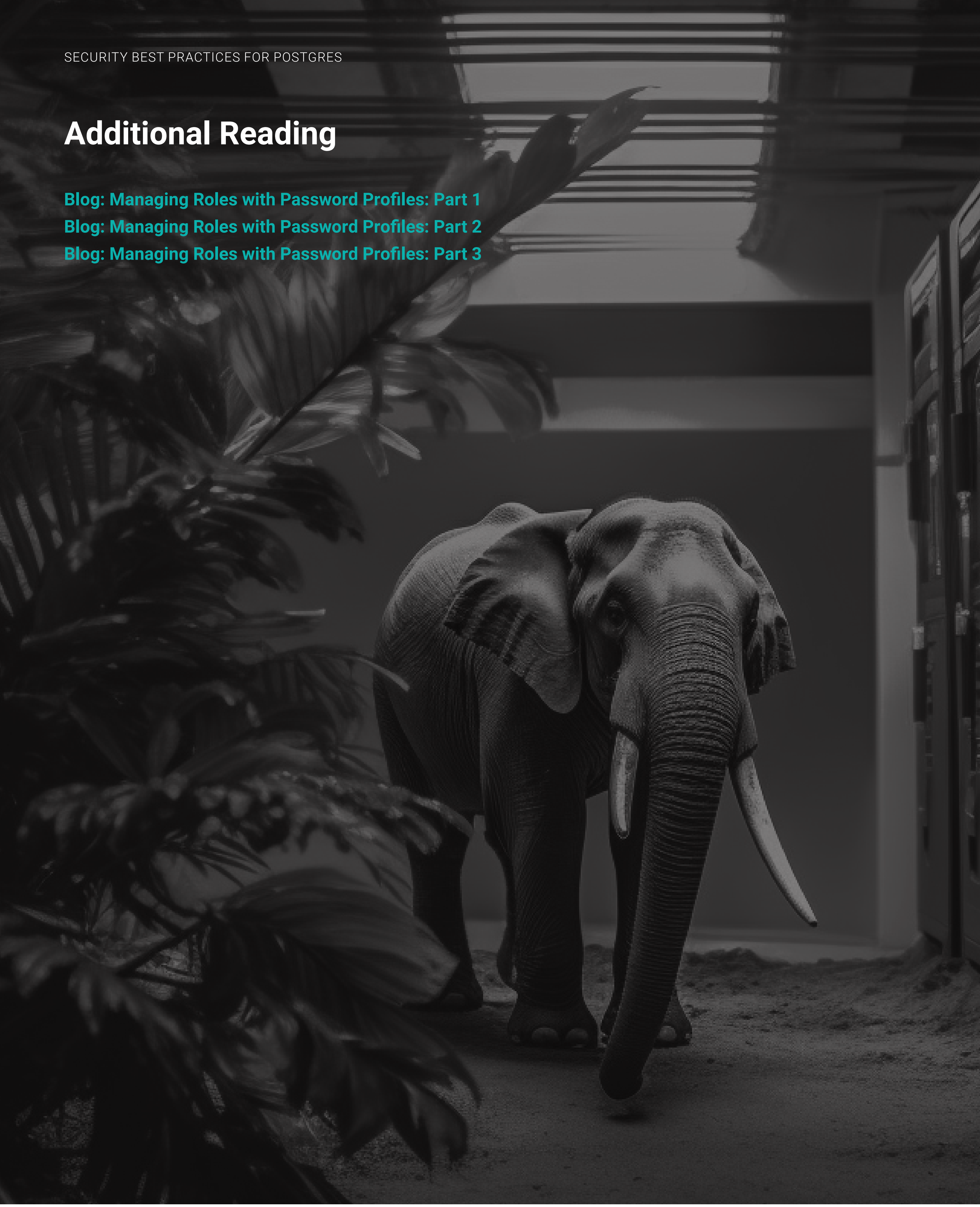
Lastly, we explored how encryption plays a big role in the overall security posture of your Postgres deployments and encompasses several different levels of encryption options, such as Transparent Data Encryption, File System Encryption and Full Disk Encryption, to name a few—and the benefits they bring. Hopefully, this overview is helpful for reviewing the overall security of your deployments. But, do remember that each deployment scenario is unique and the suggestions made here are not a "one size fits all" solution.

Use Postgres—get stuff done!

6. Additional Reading

Additional Reading

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

EDB provides enterprise-class software and services that enable businesses and governments to harness the full power of Postgres, the world's leading open source database. With offices worldwide, EDB serves more than 1,500 customers, including leading financial services, government, media and communications and information technology organizations. As one of the leading contributors to the vibrant and fast-growing Postgres community, EDB is committed to driving technology innovation. With deep database expertise, EDB ensures extreme high availability, reliability, security, 24x7 global support and advanced professional services, both on premises and in the cloud. This empowers enterprises to control risk, manage costs and scale efficiently. For more information, visit **www.enterprisedb.com**.

EDB™

Security Best Practices for Postgres

2023 Update