Database Security Cheat Sheet

Introduction

This cheat sheet provides guidance on securely configuring and using the SQL and NoSQL databases. It is intended to be used by application developers when they are responsible for managing the databases, in the absence of a dedicated database administrator (DBA). For details about protecting against SQL Injection attacks, see the SQL Injection Prevention Cheat Sheet.

Connecting to the Database

The backend database used by the application should be isolated as much as possible, in order to prevent malicious or undesirable users from being able to connect to it. Exactly how this is achieved will depend on the system and network architecture. The following options could be used to protect it:

- Disabling network (TCP) access and requiring all access is over a local socket file or named pipe.
- Configuring the database to only bind on localhost.
- Restricting access to the network port to specific hosts with firewall rules.
- Placing the database server in a separate DMZ isolated from the application server.

Similar protection should be implemented to protect any web-based management tools used with the database, such as phpMyAdmin.

When an application is running on an untrusted system (such as a thick-client), it should always connect to the backend through an API that can enforce appropriate access control and restrictions. Direct connections should **never** be made from a thick client to the backend database.

Transport Layer Protection

Most databases will allow unencrypted network connections in their default configurations. Although some will encrypt the initial authentication (such as Microsoft SQL Server), the rest of the traffic will be unencrypted, meaning that all kinds of sensitive information will be sent across the network in clear text. The following steps should be taken to prevent unencrypted traffic:

- Configure the database to only allow encrypted connections.
- Install a trusted digital certificate on the server.
- Configure the client application to connect using TLSv1.2+ with modern ciphers (e.g, AES-GCM or ChaCha20).
- Configure the client application to verify that the digital certificate is correct.

The Transport Layer Protection and TLS Cipher String Cheat Sheets contain further guidance on securely configuring TLS.

Authentication

The database should be configured to always require authentication, including connections from the local server. Database accounts should be:

- Protected with strong and unique passwords.
- Used by a single application or service.
- Configured with the minimum permissions required as discussed in the permissions section below.

As with any system that has its own user accounts, the usual account management processes should be followed, including:

- Regular reviews of the accounts to ensure that they are still required.
- Regular reviews of permissions.
- Removing user accounts when an application is decommissioned.
- Changing the passwords when staff leave, or there is reason to believe that they may have been compromised.

For Microsoft SQL Server, consider the use of Windows or Integrated-Authentication, which uses existing Windows accounts rather than SQL Server accounts. This also removes the requirement to store credentials in the application, as it will connect using the credentials of the Windows user it is running under. The Windows Native Authentication Plugins provides similar functionality for MySQL.

Storing Database Credentials

Database credentials should never be stored in the application source code, especially if they are unencrypted. Instead, they should be stored in a configuration file that:

- Is outside of the webroot.
- Has appropriate permissions so that it can only be read by the required user(s).
- Is not checked into source code repositories.

Where possible, these credentials should also be encrypted or otherwise protected using built-in functionality, such as the `web.config` encryption available in ASP.NET.

Permissions

The permissions assigned to database user accounts should be based on the principle of least privilege (i.e, the accounts should only have the minimal permissions required for the application to function). This can be applied at a number of increasingly granular levels depending on the functionality available in the database. The following steps should be applicable to all environments:

- Do not use the built-in `root`, `sa` or `SYS` accounts.

- Do not grant the account administrative rights over the database instance.
- Only allow the account to connect from allowed hosts.
  - This would often be `localhost` or the address of the application server.
- Only grant the account access to the specific databases it needs.
  - Development, UAT and Production environments should all use separate databases and accounts.
- Only grant the required permissions on the databases.
  - Most applications would only need `SELECT`, `UPDATE` and `DELETE` permissions.
  - The account should not be the owner of the database as this can lead to privilege escalation vulnerabilities.
- Avoid using database links or linked servers.
  - Where they are required, use an account that has been granted access to only the minimum databases, tables, and system privileges required.

For more security-critical applications, it is possible to apply permissions at more granular levels, including:

- Table-level permissions.
- Column-level permissions.
- Row-level permissions
- Blocking access to the underlying tables, and requiring all access through restricted views.

Database Configuration and Hardening

The underlying operating system for the database server should be hardened in the same way as any other server, based on a secure baseline such as the CIS Benchmarks or the Microsoft Security Baselines.

The database application should also be properly configured and hardened. The following principles should apply to any database application and platform:

- Install any required security updates and patches.
- Configure the database services to run under a low privileged user account.
- Remove any default accounts and databases.
- Store transaction logs on a separate disk to the main database files.
- Configure a regular backup of the database.
  - Ensure that the backups are protected with appropriate permissions, and ideally encrypted.

The following sections gives some further recommendations for specific database software, in addition to the more general recommendations given above.

Microsoft SQL Server

- Disable `xp_cmdshell`, `xp_dirtree` and other stored procedures that are not required.
- Disable Common Language Runtime (CLR) execution.
- Disable the SQL Browser service.
- Disable Mixed Mode Authentication unless it is required.
- Ensure that the sample Northwind and AdventureWorks databases have been removed.
- See Microsoft's articles on securing SQL Server.

MySQL and MariaDB

- Run the `mysql_secure_installation` script to remove the default databases and accounts.
- Disable the FILE privilege for all users to prevent them reading or writing files.
- See the Oracle MySQL and MariaDB hardening guides.

PostgreSQL

- See the PostgreSQL Server Setup and Operation documentation and the older Security documentation.

MongoDB

- See the MongoDB security checklist.

Redis

- See the Redis security guide.