MySQL 8.0 Secure Deployment Guide

Abstract

This is the MySQL 8.0 Secure Deployment Guide. It documents procedures for deploying a Linux-generic binary distribution of MySQL Enterprise Edition Server with features for implementing and managing the security of a MySQL installation. The deployment described in this guide is performed on Oracle Linux.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2020-09-08 (revision: 67269)

Table of Contents

Preface and Legal Notices v
1 Introduction 1
2 Downloading the MySQL for Linux Generic Binary Package 3
3 Verifying Package Integrity 5
4 Installing the MySQL Binary Package 7
5 Post Installation Setup 9
6 Installing the MySQL Password Validation Component 15
7 Installing MySQL Enterprise Audit 19
8 Installing MySQL Enterprise Firewall 21
9 Installing Connection Control Plugins 23
10 Block Encryption Mode Configuration 27
11 Enabling Authentication 29
12 Configuring MySQL to Use Secure Connections 33
13 Creating User Accounts 37
14 Connecting to the Server 41
A Transparent Data Encryption (TDE) and MySQL Keyring 45
B Data Masking and De-Identification 47
C FIPS Support 49
D SQL Roles and Dynamic Privileges 51
E Installation Directory and File Permissions 53
F Deployment Configuration File 55

Preface and Legal Notices

This is the *MySQL 8.0 Secure Deployment Guide*. It documents procedures for deploying a Linux-generic binary distribution of MySQL Enterprise Edition Server with features for implementing and managing the security of a MySQL installation. The deployment described in this guide is performed on Oracle Linux.

Legal Notices

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Chapter 1 Introduction

The *MySQL 8.0 Secure Deployment Guide* documents procedures for deploying a Linux-generic binary distribution of MySQL Enterprise Edition Server with features for implementing and managing the security of your MySQL installation. The deployment is performed on Oracle Linux.

The deployment is specific to MySQL Enterprise Edition. Features required by the deployment, such as MySQL Enterprise Audit, MySQL Enterprise Firewall, and auto-generation of SSL certificates and keys, are only available with MySQL Enterprise Edition.

Deployment of the MySQL Enterprise Transparent Data Encryption (TDE) feature, which protects critical data by enabling data-at-rest encryption, is not covered in this guide. For more information, see Appendix A, *Transparent Data Encryption (TDE) and MySQL Keyring*.

Enabling FIPS (Federal Information Processing Standards) mode, which imposes conditions on cryptographic operations such as restrictions on acceptable encryption algorithms or requirements for longer key lengths, is not covered in this guide. For more information, see Appendix C, *FIPS Support*.

Enabling the MySQL Enterprise Data Masking and De-Identification extension, which can be used to mask sensitive data, is not covered in this guide. For more information, see Appendix B, *Data Masking and De-Identification*.

The deployment of other MySQL products such as MySQL Workbench, MySQL NDB Cluster, MySQL Shell, and MySQL Connectors is not covered in this guide.

This guide adheres to the following principles which form the basis of a secure MySQL deployment:

- Always use the latest MySQL release, which has the latest security features and patches.
- Always practice the principle of least privilege, which requires that users, processes, programs, and other system components only have access to information and resources that are required for their legitimate purpose.

A secure deployment also requires implementation of security policies that protect the entire server host (not just the MySQL server) against all types of applicable attacks. Such polices include but are not limited to using a firewall, securing operating system access, and employing enhanced security modules such as SELinux and AppArmour. These types of server host security measures are not covered in this guide.

For more information about security topics related to MySQL server and related applications, see Security.

Chapter 2 Downloading the MySQL for Linux Generic Binary Package

To download the latest `MySQL Enterprise Edition for Linux x86-64` generic binary package, perform the following steps. If you already have the latest package, you can skip this procedure.

1. Sign into Oracle Software Delivery Cloud. Create an account if you do not have one.
2. On the *Oracle Software Delivery Cloud* page:
   - Select **Release** from the drop-down list.
   - Type **`MySQL Server`** in the text box.
   - Select **MySQL Server 8.0.*xx*** from list to add it to your shopping cart, where *xx* is the latest MySQL 8.0 Server release.
   - Click **Selected Software** next to the shopping cart icon.
   - Select **Linux x86-64** from the **Platform / Languages** drop-down list.
   - Click **Continue**.
3. Review the *Oracle Standard Terms and Restrictions* and select the check box indicating that you have reviewed and accept the terms of the Commercial License, Special Programs License, and/or Trial License. Click **Continue**.
4. Deselect all MySQL Enterprise Edition packages except for `MySQL Commercial Server 8.0.xx TAR for Generic Linux x86 (64bit)`. Click **Download**.

   If your browser is not compatible with the download manager, click on the zip file name to start the download. The zip file name is similar to `VXXXXXX-XX.zip`, where *XXXXXX-XX* is a numerical identifier.
5. When the zip file download completes, extract the contents of the zip file to a location of your choice. The extracted files include those shown in the following table:

Table 2.1 MySQL Package and Signature Files for Source files

File Type	File Name
README file	`README.txt`
MD5 signature file	`mysql-commercial-8.0.xx-linux-glibc2.12-x86_64.tar.xz.md5`
GPG signature file	`mysql-commercial-8.0.xx-linux-glibc2.12-x86_64.tar.xz.asc`
Distribution file	`mysql-commercial-8.0.xx-linux-glibc2.12-x86_64.tar.xz`

Chapter 3 Verifying Package Integrity

After downloading the MySQL package and before attempting to install it, ensure that the package is intact and has not been tampered with. There are two methods of integrity checking for MySQL Linux Generic Binary packages: *MD5 Checksum* and *Signature Checking Using GnuPG*. *Signature Checking Using GnuPG* is used in this deployment.

MySQL signs its downloadable packages with `GnuPG` (GNU Privacy Guard). Most Linux distributions ship with `GnuPG` installed by default. Otherwise, see http://www.gnupg.org/ for more information about `GnuPG` and how to obtain and install it.

1. To verify the signature of your MySQL download package, obtain a copy of the MySQL public GPG build key, which you can download from http://pgp.mit.edu/. The key name is `mysql-build@oss.oracle.com`.

   a. In your browser, navigate to http://pgp.mit.edu/.

   b. In the **Search String** field, enter the key name, `mysql-build@oss.oracle.com`, and click **Do the search!**

      This search result is returned:

      ```
      Type bits/keyID     Date       User ID
      ----------------------------------------------------------------------------------------
      pub  1024D/5072E1F5 2003-02-03 MySQL Release Engineering <mysql-build@oss.oracle.com>
                                     MySQL Package signing key (www.mysql.com) <build@mysql.com>
      ```

      2003-02-03 is the initial creation date for the MySQL Package Signing Key.

   c. Click on the `keyID` link, copy the key, and save it to a file named `mysql_pubkey.asc`, for example.

      Alternatively, you can copy and paste the key directly from the MySQL Reference Manual. See Signature Checking Using GnuPG.

2. To import the build key into your personal public GPG keyring, use the `gpg --import` command. For example, if you saved the key to a file named `mysql_pubkey.asc`, the import command looks like this:

   ```
   shell> gpg --import mysql_pubkey.asc
   gpg: key 5072E1F5: public key "MySQL Release Engineering
   <mysql-build@oss.oracle.com>" imported
   gpg: Total number processed: 1
   gpg:               imported: 1
   gpg: no ultimately trusted keys found
   ```

3. After importing the public build key, place the public build key file that you created in the same directory as the `.asc` signature file that was included in the MySQL download package.

   The signature file has the same name as the distribution file with an `.asc` extension, as shown in the following table.

   Table 3.1 MySQL Package and Signature Files for Source files

File Type	File Name
Distribution file	`mysql-commercial-8.0.xx-linux-glibc2.12-x86_64.tar.xz`
Signature file	`mysql-commercial-8.0.xx-linux-glibc2.12-x86_64.tar.xz.asc`

4. Run this command to verify the signature for the distribution file:

   ```
   shell> gpg --verify mysql-commercial-8.0.xx-linux-glibc2.12-x86_64.tar.xz.asc
   ```

If the downloaded package is valid, the verification returns a "Good signature" message similar to:

```
shell> gpg --verify mysql-commercial-8.0.xx-linux-glibc2.12-x86_64.tar.xz.asc
gpg: Signature made Tue 01 Feb 2011 02:38:30 AM CST using DSA key ID 5072E1F5
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
```

The `Good signature` message indicates that the file signature is valid, but you might also see warnings:

```
shell> gpg --verify mysql-commercial-8.0.xx-linux-glibc2.12-x86_64.tar.xz.asc
gpg: Signature made Wed 23 Jan 2013 02:25:45 AM PST using DSA key ID 5072E1F5
gpg: checking the trustdb
gpg: no ultimately trusted keys found
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: A4A9 4068 76FC BD3C 4567  70C8 8C71 8D3B 5072 E1F5
```

To encounter warnings is normal, as they depend on your setup and configuration. Here are explanations for the warnings:

- *gpg: no ultimately trusted keys found*: This means that the specific key is not "ultimately trusted" by you or your web of trust, which is okay for the purposes of verifying file signatures.
- *WARNING: This key is not certified with a trusted signature! There is no indication that the signature belongs to the owner.*: This refers to your level of trust in your belief that you possess our real public key. This is a personal decision. Ideally, a MySQL developer would hand you the key in person, but more commonly, you downloaded it. Was the download tampered with? Probably not, but this decision is up to you. Setting up a web of trust is one method for trusting them.

If the GPG signatures do not match, try to download the respective package one more time.

For additional information about GnuPG signature checking, see Signature Checking Using GnuPG.

Chapter 4 Installing the MySQL Binary Package

This section covers installation prerequisites, creating the `mysql` user and group, and unpacking the distribution.

- Installation Prerequisites
- Creating the mysql User and Group
- Unpacking the Distribution

Installation Prerequisites

- The installation must be performed as an operating system `root` user, as the installation process involves creating a user, a group, directories, and assigning ownership and permissions. Installed MySQL binaries are owned by the operating system `root` user.

  > **Note**
  >
  > Unless otherwise indicated, procedures in this guide are performed as the operating system `root` user.

- MySQL has a dependency on the `libaio` library. Data directory initialization and subsequent server startup steps fail if this library is not installed locally. If necessary, install it using the appropriate package manager. For example, on Yum-based systems:

```
shell> yum search libaio  # search for info
shell> yum install libaio # install library
```

- **Oracle Linux 8** does not install the file `/lib64/libtinfo.so.5` by default, which is required by the MySQL client `bin/mysql` for `mysql-VERSION-linux-glibc2.12-x86_64.tar.xz` packages. To work around this issue, install the `ncurses-compat-libs` package:

```
shell> yum install ncurses-compat-libs
```

Creating the mysql User and Group

The `mysql` user owns the MySQL data directory. It is also used to run the `mysqld` server process, as defined in the systemd `mysqld.service` file (see Starting the Server using systemd). The `mysql` user has read and write access to anything in the MySQL data directory. It does not have the ability to log into MySQL. It only exists for ownership purposes.

The `mysql` group is the database administrator group. Users in this group have read and write access to anything in the MySQL data directory, and execute access on any packaged MySQL binary.

This command adds the `mysql` group.

```
shell> groupadd -g 27 -o -r mysql
```

The groupadd `-g 27` and `-o` options assign a non-unique group ID (GID). The `-r` option makes the group a system group.

This command adds the `mysql` user:

```
shell> useradd -M -N -g mysql -o -r -d datadir -s /bin/false -c "MySQL Server" -u 27 mysql
```

- The `-M` option prevents the creation of a user home directory.
- The `-N` option indicates that the user should be added to the group specified by the `-g` option.
- The `-o` and `-u 27` options assign a non-unique user ID (UID).

- The `-r` and `-s /bin/false` options create a user without login permissions to the server host. The `mysql` user is required only for ownership purposes, not login purposes.
- The `-d` option specifies the user login directory, which is set to the expected MySQL data directory path. The expected data directory path in this deployment is `/usr/local/mysql/data`.
- The `-c` option specifies a comment describing the account.

Unpacking the Distribution

To extract the binary files from the verified *MySQL Linux Generic Binary* download package:

1. Change location to the directory under which you want to unpack the MySQL distribution. In this deployment, the distribution is unpackaged by `root` under `/usr/local`.

   ```
   shell> cd /usr/local
   ```

2. Unpack the MySQL distribution, which creates the installation directory. Any modern `tar` program can uncompress and unpack the distribution with this command:

   ```
   shell> tar xvf /path/to/mysql-commercial-8.0.xx-linux-glibc2.12-x86_64.tar.xz
   ```

   The `tar` command creates a directory named `mysql-VERSION-OS`. In this case, the directory is named `mysql-commercial-8.0.xx-linux-glibc2.12-x86_64`, where `xx` is the latest release.

3. Create a relative symbolic link to the installation directory created by `tar`:

   ```
   shell> cd /usr/local
   shell> ln -s mysql-commercial-8.0.xx-linux-glibc2.12-x86_64 mysql
   ```

   The `ln` command makes a symbolic link to the installation directory. This enables you to refer more easily to it as `/usr/local/mysql`.

   > **Note**
   >
   > To avoid typing the path name of client programs when working with MySQL, add the `/usr/local/mysql/bin` directory to your `PATH` variable:
   >
   > ```
   > shell> export PATH=/usr/local/mysql/bin:$PATH
   > ```

Unpacking the distribution creates the directories shown in the following table. The directories are located in the MySQL installation directory, which is `/usr/local/mysql`:

Table 4.1 MySQL Linux Generic Binary Distribution Directories

Directory	Contents of Directory
`bin`	`mysqld` server; client and utility programs
`docs`	MySQL manual in Info format
`man`	Unix manual pages
`include`	Include (header) files
`lib`	Libraries
`share`	Miscellaneous files, including error messages, sample configuration files, SQL for database installation
`support-files`	Miscellaneous support files related to managing multiple server processes, automatic startup configuration, and log rotation.

Also included in the MySQL installation directory are the `README` and `LICENSE` files. There is no data directory. It is created later when the data directory is initialized.

Chapter 5 Post Installation Setup

Post-installation setup involves creating a safe directory for import and export operations, configuring server startup options, initializing the data directory, starting MySQL using systemd, resetting the password for the MySQL `root@localhost` user account, and running a few tests to ensure that the server is working.

- Creating a Safe Directory For Import and Export Operations
- Configuring Server Startup Options
- Initializing the Data Directory
- Starting the Server using systemd
- Resetting the MySQL root Account Password
- Testing the Server

Creating a Safe Directory For Import and Export Operations

MySQL users with the `FILE` privilege have permission to read and write files on the server host using `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements, and the `LOAD_FILE()` function. By default, a user who has the `FILE` privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.) The `FILE` privilege also enables the user to create new files in any directory where the MySQL server has write access. This includes the server data directory containing the files that implement the privilege tables.

To limit the scope of the `FILE` privilege, create a directory that users with the `FILE` privilege can safely use for import and export operations. In this deployment, the directory created is named `mysql-files` and is located under the data directory. In a later step, when server startup options are configured, the `secure_file_priv` option is set to the `mysql-files` directory.

```
shell> cd /usr/local/mysql
shell> mkdir mysql-files
shell> chown mysql:mysql mysql-files
shell> chmod 750 mysql-files
```

Configuring Server Startup Options

Specify options that the MySQL server should use at startup by placing them in a MySQL configuration file. If you do not do this, the server starts with its default settings (see Server Configuration Defaults).

> **Note**
>
> Certain `InnoDB` options can only be configured before initializing the data directory. Among these options are `innodb_data_home_dir`, `innodb_data_file_path`, `innodb_log_file_size`, `innodb_log_group_home_dir`, and `innodb_page_size`. If you do not want to use default values for these options, set your own values in the MySQL configuration file before initializing the data directory. This deployment uses default `InnoDB` configuration settings. For more information, see InnoDB Startup Configuration.

1. To create a MySQL configuration file, issue these commands as root:

```
shell> cd /etc
shell> touch my.cnf
shell> chown root:root my.cnf
```

```
shell> chmod 644 my.cnf
```

> **Note**
>
> If there is an existing `my.cnf` configuration file in the same location that belongs to another MySQL instance, use a different name for your configuration file.

2. Under a `[mysqld]` group entry, set the `datadir`, `socket`, `port`, `log-error` options for the instance. If there are other MySQL installations on the host, ensure that the values for these options are unique to this instance. This deployment uses the default values.

```
[mysqld]
datadir=/usr/local/mysql/data
socket=/tmp/mysql.sock
port=3306
log-error=/usr/local/mysql/data/localhost.localdomain.err
```

> **Important**
>
> The location of the MySQL data directory is critically important to the security of a MySQL installation. In addition to user data, the data directory contains data dictionary and system tables, which store sensitive information about database objects, users, privileges, and so on. Following the principle of least privilege, system user access to the data directory should be as restrictive as possible. The size of the file system where the data directory resides should also be considered. Ensure that the file system can accommodate the anticipated size of your data. The deployment described in this guide places the data directory in the default location (`/usr/local/mysql/data`), and access to the directory is limited to the `mysql` operating system user account.

3. Set the `user` option to ensure that the server is started as the unprivileged `mysql` user account. For security reasons, it is important to avoid running the server as the operating system `root` user.

```
user=mysql
```

4. If you intend to permit import and export operations, set the `secure_file_priv` system variable to the path of the `mysql-files` directory that you created previously. This option limits file import and export operations, such as those performed by the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function, to the specified directory. If you do not intend to permit import or export operations, set `secure_file_priv` to `NULL`, which disables import and export operations entirely. `NULL` is the default setting.

```
secure_file_priv=/usr/local/mysql/mysql-files
```

5. To avoid potential security issues with the `LOCAL` version of `LOAD DATA`, ensure that `local_infile` is disabled, which it is by default.

```
local_infile=OFF
```

   For more information, see Security Considerations for LOAD DATA LOCAL.

After completing the steps above, the configuration file should contain these settings, assuming you have not added others:

```
[mysqld]
datadir=/usr/local/mysql/data
socket=/tmp/mysql.sock
port=3306
log-error=/usr/local/mysql/data/localhost.localdomain.err
user=mysql
secure_file_priv=/usr/local/mysql/mysql-files
local_infile=OFF
```

Initializing the Data Directory

After installing MySQL, you must initialize the data directory, which includes the `mysql` system database and its tables, including grant tables, server-side help tables, and time zone tables. Initialization also creates the `root@localhost` superuser account and the `InnoDB` system tablespace and related data structures required to manage `InnoDB` tables.

To initialize the data directory:

1. Change location to the top-level directory of the MySQL installation, create the data directory, and grant ownership to the `mysql` user.

```
shell> cd /usr/local/mysql
shell> mkdir data
shell> chmod 750 data
shell> chown mysql:mysql data
```

> **Note**
>
> Data directory ownership is assigned to the `mysql` user, but most of the MySQL installation remains owned by `root`. Other exceptions are the error log file, the `mysql-files` directory, the pid file, and socket file, to which the `mysql` user must have write access. Files and resources that the `mysql` user requires read access to include configuration files (e.g., `/etc/my.cnf`) and the MySQL binaries (`/usr/local/mysql/bin`).

2. Initialize the data directory.

```
shell> cd /usr/local/mysql
shell> bin/mysqld --defaults-file=/etc/my.cnf --initialize
```

Initialization output is printed to the error log (`/usr/local/mysql/data/localhost.localdomain.err`) and appears similar to the output shown below. The output includes an initial random password for the `root@localhost` account. The password is required later to reset the `root@localhost` password.

```
2018-05-02T17:47:49.806563Z 0 [System] [MY-013169] [Server]
/usr/local/mysql-commercial-8.0.11-linux-glibc2.12-x86_64/bin/mysqld (mysqld 8.0.11-commercial)
initializing of server in progress as process 16039
2018-05-02T17:47:54.083010Z 5 [Note] [MY-010454] [Server]
A temporary password is generated for root@localhost: uZmx9ihSd2;.
2018-05-02T17:47:56.225881Z 0 [System] [MY-013170] [Server]
/usr/local/mysql-commercial-8.0.11-linux-glibc2.12-x86_64/bin/mysqld (mysqld 8.0.11-commercial)
initializing of server has completed
```

> **Note**
>
> Data directory initialization creates time zone tables in the `mysql` database but does not populate them. To do so, refer to the instructions in MySQL Server Time Zone Support.

For more information about data directory initialization, see Initializing the Data Directory.

Starting the Server using systemd

This section describes how to start the server with systemd and how to enable automatic restart of the MySQL server when the host is rebooted.

systemd provides manual server management using the `systemctl` command:

```
systemctl {start|stop|restart|status} mysqld
```

To configure the MySQL installation to work with systemd:

1. Add a systemd service unit configuration file with details about the MySQL service. The file is named `mysqld.service` and is placed in `/usr/lib/systemd/system`.

```
shell> cd /usr/lib/systemd/system
shell> touch mysqld.service
shell> chmod 644 mysqld.service
```

   Add this configuration information to the `mysqld.service` file:

```
[Unit]
Description=MySQL Server
Documentation=man:mysqld(8)
Documentation=http://dev.mysql.com/doc/refman/en/using-systemd.html
After=network.target
After=syslog.target

[Install]
WantedBy=multi-user.target

[Service]
User=mysql
Group=mysql

# Have mysqld write its state to the systemd notify socket
Type=notify

# Disable service start and stop timeout logic of systemd for mysqld service.
TimeoutSec=0

# Start main service
ExecStart=/usr/local/mysql/bin/mysqld --defaults-file=/etc/my.cnf $MYSQLD_OPTS

# Use this to switch malloc implementation
EnvironmentFile=-/etc/sysconfig/mysql

# Sets open_files_limit
LimitNOFILE = 10000

Restart=on-failure

RestartPreventExitStatus=1

# Set environment variable MYSQLD_PARENT_PID. This is required for restart.
Environment=MYSQLD_PARENT_PID=1

PrivateTmp=false
```

2. Enable the `mysqld` service to automatically start at reboot.

```
shell> systemctl enable mysqld.service
Created symlink from /etc/systemd/system/multi-user.target.wants/mysqld.service
to /usr/lib/systemd/system/mysqld.service.
```

3. To ensure the systemd configuration works, start the `mysqld` service manually using `systemctl`.

```
shell> systemctl start mysqld
```

4. Check the status of the `mysqld` service. The output should appear similar to the following, which shows that the `mysqld` service was started successfully.

```
shell> systemctl status mysqld
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2018-05-02 18:18:05 ADT; 5s ago
     Docs: man:mysqld(8)
           http://dev.mysql.com/doc/refman/en/using-systemd.html
 Main PID: 19520 (mysqld)
   Status: "SERVER_OPERATING"
   CGroup: /system.slice/mysqld.service
           └─19520 /usr/local/mysql/bin/mysqld --defaults-file=/etc/my.cnf
```

```
May 02 18:18:04 localhost.localdomain systemd[1]: Starting MySQL Server...
May 02 18:18:05 localhost.localdomain systemd[1]: Started MySQL Server.
```

5. To verify that systemd automatically starts MySQL when the system is rebooted, restart your system and check the status of the `mysqld` service again.

```
shell> systemctl status mysqld
```

> **Note**
>
> systemd has its own log file which can be accessed using `journalctl`. To view mysqld-related log messages, use `journalctl -u mysqld`. Some messages, such as MySQL startup messages, may be printed to the systemd log.

For more information about systemd, see Managing MySQL Server with systemd.

Resetting the MySQL root Account Password

This procedure assumes that the MySQL server is running. You can check server status by issuing:

```
shell> systemctl status mysqld
```

When the data directory was initialized, a random initial password was generated for the MySQL `root` account (`root@localhost`) and marked as expired. Perform these steps to set a new password:

1. Using the `mysql` client, connect to the server as `root@localhost` using the random password that the server generated during the initialization sequence:

```
shell> cd /usr/local/mysql
shell> bin/mysql -u root -p
Enter password: (enter the random root password here)
```

2. After connecting, assign a new `root@localhost` password. Use a strong password. Ideally, the password should conform to the password policy that you will define using the `validate_password` component, which is enabled in a later step. (See Chapter 6, *Installing the MySQL Password Validation Component*.)

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'password';
```

   Alternatively, you can generate a random password using the `RANDOM PASSWORD` option. For more information, see Random Password Generation.

Testing the Server

Now that MySQL is installed and initialized, and the MySQL `root` user password is reset, perform a couple of simple tests to verify that the server works.

1. Use `mysqlshow` to verify that you can retrieve information from the server.

```
shell> cd /usr/local/mysql
shell> bin/mysqlshow -u root -p
Enter password: (enter root password here)
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

2. Use `mysqladmin` to view MySQL server version information.

```
shell> cd /usr/local/mysql
```

```
shell> bin/mysqladmin -u root -p version
Enter password: (enter root password here)
```

The output should be similar to that shown here:

```
bin/mysqladmin  Ver 8.0.19-commercial for linux-glibc2.12 on x86_64
(MySQL Enterprise Server - Commercial)
Copyright (c) 2000, 2020, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Server version  8.0.19-commercial
Protocol version 10
Connection  Localhost via UNIX socket
UNIX socket  /tmp/mysql.sock
Uptime:   11 min 7 sec

Threads: 3  Questions: 8  Slow queries: 0  Opens: 146  Flush tables: 3
Open tables: 63  Queries per second avg: 0.011
```

For additional tests, see Testing the Server.

Chapter 6 Installing the MySQL Password Validation Component

The `validate_password` component serves to test user-specified passwords and improve security. The component exposes a set of system variables that enable you to define a password policy.

The component implements two capabilities:

- In statements that assign a password supplied as a cleartext value, the component checks the password against the current password policy and rejects the password if it is weak. This affects the `ALTER USER`, `CREATE USER`, and `SET PASSWORD` statements.
- The `VALIDATE_PASSWORD_STRENGTH()` SQL function assesses the strength of potential passwords. The function takes a password argument and returns an integer from 0 (weak) to 100 (strong).

The `validate_password` component provides three levels of password checking: `LOW`, `MEDIUM`, and `STRONG`. The default is `MEDIUM`; controlled by the `validate_password.policy` system variable. The policies implement increasingly strict password tests.

- The `LOW` policy tests password length only. Passwords must be at least 8 characters long. To change this length, modify `validate_password.length`.
- The `MEDIUM` policy adds the conditions that passwords must contain at least 1 numeric character, 1 lowercase character, 1 uppercase character, and 1 special (nonalphanumeric) character. To change these values, modify `validate_password.number_count`, `validate_password.mixed_case_count`, and `validate_password.special_char_count`.
- The `STRONG` policy adds the condition that password substrings of length 4 or longer must not match words in the dictionary file, if one has been specified. To specify the dictionary file, modify `validate_password.dictionary_file`.

In addition, the `validate_password` component can reject passwords that match the user name part of the effective user account for the current session, either forward or in reverse. To provide control over this capability, `validate_password` exposes a `validate_password.check_user_name` system variable, which is enabled by default.

To install and configure the password validation component:

1. Ensure that the `validate_password` component library file is located in the MySQL plugin directory.

   ```
   shell> cd /path/to/mysql/lib/plugin/
   shell> ls component_v*
   component_validate_password.so
   ```

   Ensure that the `plugin_dir` is set to the server the MySQL plugin directory.

   ```
   mysql> SELECT @@plugin_dir;
   +-----------------------------------------+
   | @@plugin_dir                            |
   +-----------------------------------------+
   | /path/to/mysql/lib/plugin/              |
   +-----------------------------------------+
   ```

   Install the `validate_password` component using the `INSTALL COMPONENT` statement:

   ```
   mysql> INSTALL COMPONENT 'file://component_validate_password';
   ```

   Component installation is a one-time operation that need not be done per server startup. `INSTALL COMPONENT` loads the component, and also registers it in the `mysql.component` system table to cause it to be loaded during subsequent server startups.

2. Add these options under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`) so that you can adjust them as necessary. The default values are used in this deployment.

```
validate_password.policy=1
validate_password.length=8
validate_password.number_count=1
validate_password.mixed_case_count=1
validate_password.special_char_count=1
validate_password.check_user_name=1
```

   - `validate_password.policy=1`

     The password policy enforced by `validate_password`. A value of 1 is `MEDIUM`. By default, the `MEDIUM` policy specifies that passwords must be at least 8 characters long, contain at least 1 numeric character, 1 lowercase character, 1 uppercase character, and 1 special (nonalphanumeric) character. 1 (`MEDIUM`) is the default setting.

   - `validate_password.length=8`

     The minimum number of characters that `validate_password` requires passwords to have.

   - `validate_password.number_count=1`

     The minimum number of lowercase and uppercase characters that `validate_password` requires passwords to have if the password policy is `MEDIUM` or stronger.

   - `validate_password.mixed_case_count=1`

     The minimum number of numeric (digit) characters that `validate_password` requires passwords to have if the password policy is `MEDIUM` or stronger.

   - `validate_password.special_char_count=1`

     The minimum number of nonalphanumeric characters that `validate_password` requires passwords to have if the password policy is `MEDIUM` or stronger.

   - `validate_password.check_user_name=1`

     Rejects passwords that match the user name part of the effective user account for the current session, either forward or in reverse.

     > **Note**
     >
     > `validate_password.dictionary_file` is not used in this deployment. By default, this variable has an empty value and dictionary checks are not performed. For the dictionary file to be used during password checking, the password policy must be set to 2 (STRONG); see the description of the `validate_password.policy` system variable for more information.

3. To verify component installation, query the `mysql.component` table:

```
shell> cd /usr/local/mysql
shell> bin/mysqladmin -u root -p version
Enter password: (enter root password here)
```

```
mysql> SELECT * FROM mysql.component;
+--------------+--------------------+-----------------------------------+
| component_id | component_group_id | component_urn                     |
+--------------+--------------------+-----------------------------------+
|            1 |                  1 | file://component_validate_password |
+--------------+--------------------+-----------------------------------+
```

4. To verify that the password validation component works, attempt to create a user with a non-compliant password:

```
mysql> CREATE USER 'bob.smith'@'localhost' IDENTIFIED BY 'abc';
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
```

> **Note**
>
> The policy that the `validate_password` component implements has no effect on generated passwords. The purpose of a `validate_password` policy is to help *humans* create better passwords.

For more information about the `validate_password` component, see The Password Validation Component.

Chapter 7 Installing MySQL Enterprise Audit

MySQL Enterprise Audit enables standard, policy-based monitoring and logging of connection and query activity, providing an auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables the MySQL server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

After you install the plugin, it writes an audit log file. By default, the file is named `audit.log` and is located in the data directory.

To install MySQL Enterprise Audit:

1. Run the `audit_log_filter_linux_install.sql` script located in the `share` directory of your MySQL installation.

```
shell> cd /usr/local/mysql
shell> bin/mysql -u root -p < /usr/local/mysql/share/audit_log_filter_linux_install.sql
Enter password: (enter root password here)
Result
OK
```

2. Verify the plugin installation by logging in as root and examining the `INFORMATION_SCHEMA.PLUGINS` table or using the `SHOW PLUGINS` statement.

```
shell> cd /usr/local/mysql
shell> bin/mysqladmin -u root -p version
Enter password: (enter root password here)
```

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'audit%';
+-------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+-------------+---------------+
| audit_log   | ACTIVE        |
+-------------+---------------+
```

3. To prevent the plugin from being removed at runtime, add the `--audit-log` option under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`) with a setting of `FORCE_PLUS_PERMANENT`.

```
audit-log=FORCE_PLUS_PERMANENT
```

4. Restart the server to apply the configuration change:

```
shell> systemctl restart mysqld
```

5. By default, rule-based audit log filtering logs no auditable events for any users. To produce log-everything behavior with rule-based filtering, create a simple filter to enable logging and assign it to the default account:

```
mysql> SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
+--------------------------------------------------------------------------+
| audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }') |
+--------------------------------------------------------------------------+
| OK                                                                       |
+--------------------------------------------------------------------------+

mysql> SELECT audit_log_filter_set_user('%', 'log_all');
+--------------------------------------------------------------------------+
| audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }') |
+--------------------------------------------------------------------------+
| OK                                                                       |
```

```
+----------------------------------------------------------------------------+
```

The filter assigned to `%` is used for connections from any account that has no explicitly assigned filter (which initially is true for all accounts).

6. To verify that events are being logged, issue a statement such as `SHOW DATABASES` and check the `audit.log` file contents for the log event.

```
shell> cd /usr/local/mysql
shell> bin/mysqladmin -u root -p version
Enter password: (enter root password here)
```

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

The `SHOW DATABASES` statement run as `root@localhost` writes a log event to `audit.log` similar to the following:

```
<AUDIT_RECORD>
<TIMESTAMP>2018-04-20T11:28:02 UTC</TIMESTAMP>
<RECORD_ID>7_2018-04-20T11:27:29</RECORD_ID>
<NAME>Query</NAME>
<CONNECTION_ID>9</CONNECTION_ID>
<STATUS>0</STATUS>
<STATUS_CODE>0</STATUS_CODE>
<USER>root[root] @ localhost []</USER>
<OS_LOGIN/>
<HOST>localhost</HOST>
<IP/>
<COMMAND_CLASS>show_databases</COMMAND_CLASS>
<SQLTEXT>SHOW DATABASES</SQLTEXT>
</AUDIT_RECORD>
```

> **Note**
>
> Contents of the audit log file may contain sensitive information, such as the text of SQL statements. For security reasons, the file should be written to a directory accessible only to the MySQL server and users with a legitimate reason to view the log. The default audit log file is named `audit.log` and is located in the data directory. In this deployment, the data directory is owned by the `mysql` user. This location of the audit log file can be changed at server startup using the `audit_log_file` system variable.
>
> Optionally, audit log files may also be encrypted. Encryption is based on a user-defined password. To use this feature, the MySQL keyring must be enabled. Audit logging uses the MySQL keyring for password storage. For more information, see Encrypting Audit Log Files.

For more information about configuring MySQL Enterprise Audit, see MySQL Enterprise Audit.

Chapter 8 Installing MySQL Enterprise Firewall

MySQL Enterprise Firewall is an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against allowlists of accepted statement patterns. This helps harden MySQL against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics.

Each MySQL account registered with the firewall has its own statement allowlist, enabling protection to be tailored per account. For a given account, the firewall can operate in recording, protecting, or detecting mode, for training in the accepted statement patterns, active protection against unacceptable statements, or passive detection of unacceptable statements.

MySQL Enterprise Firewall installation is a one-time operation that involves running a script located in the `share` directory of your MySQL installation.

To install MySQL Enterprise Firewall:

1. Run the `linux_install_firewall.sql` script that is located in the `/usr/local/mysql/share` directory.

   The installation script creates stored procedures in the default database, so choose a database to use. Then run the script as follows, naming the chosen database on the command line. This deployment uses the `mysql` database.

   ```
   shell> cd /usr/local/mysql
   shell> bin/mysql -u root -p mysql < /usr/local/mysql/share/linux_install_firewall.sql
   Enter password: (enter root password here)
   ```

2. To enable the firewall, enable the `mysql_firewall_mode` system variable. By default, this variable is enabled when the firewall is installed. To configure the firewall state explicitly, add it under the `[mysqld]` option group in the MySQL configuration file:

   ```
   mysql_firewall_mode=ON
   ```

3. Restart MySQL server to apply the new configuration settings.

   ```
   shell> systemctl restart mysqld
   ```

4. To verify that MySQL Enterprise Firewall is enabled, connect to the server and execute this statement:

   ```
   shell> cd /usr/local/mysql
   shell> bin/mysql -u root -p
   Enter password: (enter the root password here)
   ```

   ```
   mysql> SHOW GLOBAL VARIABLES LIKE 'mysql_firewall_mode';
   +---------------------+-------+
   | Variable_name       | Value |
   +---------------------+-------+
   | mysql_firewall_mode | ON    |
   +---------------------+-------+
   ```

MySQL Enterprise Firewall is now enabled an ready for use. For information about registering accounts with the firewall and configuring operational modes, see Using MySQL Enterprise Firewall. An example is provided that demonstrates how to register an account with the firewall, use the firewall to learn acceptable statements for the account, and protect the account against execution of unacceptable statements.

Chapter 9 Installing Connection Control Plugins

The connection-control plugin library enables administrators to introduce an increasing delay in server response to connection attempts after a configurable number of consecutive failed attempts. This capability provides a deterrent that slows down brute force attacks against MySQL user accounts. The plugin library contains two plugins:

- `CONNECTION_CONTROL` checks incoming connection attempts and adds a delay to server responses as necessary. This plugin also exposes system variables that enable its operation to be configured and a status variable that provides rudimentary monitoring information.
- `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` implements an `INFORMATION_SCHEMA` table that exposes more detailed monitoring information for failed connection attempts.

To install the connection-control plugins:

1. Add these options under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

   ```
   plugin-load-add=connection_control.so
   connection-control=FORCE_PLUS_PERMANENT
   connection-control-failed-login-attempts=FORCE_PLUS_PERMANENT
   ```

   - `plugin-load-add=connection_control.so`

     Loads the `connection_control.so` library each time the server is started.

   - `connection_control=FORCE_PLUS_PERMANENT`

     Prevents the server from running without the `CONNECTION_CONTROL` plugin, and server startup fails if the plugin does not initialize successfully.

   - `connection-control-failed-login-attempts=FORCE_PLUS_PERMANENT`

     Prevents the server from running without the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` plugin, and server startup fails if the plugin does not initialize successfully.

2. To verify plugin installation, restart the server and examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement:

   ```
   shell> systemctl restart mysqld
   ```

   ```
   shell> cd /usr/local/mysql
   shell> bin/mysqladmin -u root -p version
   Enter password: (enter root password here)
   ```

   ```
   mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
          FROM INFORMATION_SCHEMA.PLUGINS
          WHERE PLUGIN_NAME LIKE 'connection%';
   +---------------------------------------+---------------+
   | PLUGIN_NAME                           | PLUGIN_STATUS |
   +---------------------------------------+---------------+
   | CONNECTION_CONTROL                    | ACTIVE        |
   | CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS | ACTIVE        |
   +---------------------------------------+---------------+
   ```

- Configuring Connection Delays
- Monitoring Failed Connection Attempts

Configuring Connection Delays

Configure the server response delay for failed connection attempts using these server parameters:

- `connection_control_failed_connections_threshold`

  The number of consecutive failed connection attempts permitted to accounts before the server adds a delay for subsequent connection attempts.

- `connection_control_min_connection_delay`

  The minimum delay in milliseconds for connection failures above the threshold.

- `connection_control_max_connection_delay`

  The maximum delay in milliseconds for connection failures above the threshold.

Add these options under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`) so that you can adjust them later as necessary. The default values are used in this deployment.

```
connection_control_failed_connections_threshold=3
connection_control_min_connection_delay=1000
connection_control_max_connection_delay=2147483647
```

For more information about server response delay configuration, see Connection-Control Plugin Installation.

Monitoring Failed Connection Attempts

Failed connection attempts can be monitored using these information sources:

- The `Connection_control_delay_generated` status variable indicates the number of times the server added a delay to its response to a failed connection attempt. This status variable does not count attempts that occur before reaching the threshold defined by the `connection_control_failed_connections_threshold` system variable.
- The `INFORMATION_SCHEMA.CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table, enabled by the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` plugin, provides information about the current number of consecutive failed connection attempts per account (user/host combination). This counts all failed attempts, regardless of whether they were delayed.

To test the connection-control plugin and view monitoring data:

1. Open a terminal and connect to the server as root:

   ```
   shell> cd /usr/local/mysql
   shell> bin/mysql -u root -p
   Enter password: (enter the root password here)
   ```

2. Open a second terminal and perform *four* connection attempts as root, specifying an incorrect password each time. There should be a small but noticeable delay on the fourth connection attempt.

   ```
   shell> cd /usr/local/mysql
   shell> bin/mysql -u root -p
   Enter password: (enter incorrect password here)
   ```

3. In the first terminal, issue this statement to view `Connection_control_delay_generated` status variable data. Connection attempts that exceed the `connection_control_failed_connections_threshold` threshold value of 3 are counted.

   ```
   mysql> SHOW STATUS LIKE 'Connection_control_delay_generated';
   +-----------------------------------+-------+
   | Variable_name                     | Value |
   +-----------------------------------+-------+
   | Connection_control_delay_generated | 1     |
   +-----------------------------------+-------+
   ```

4. In the first terminal, issue this statement to view `INFORMATION_SCHEMA.CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` data. All four failed connection attempts are counted.

```
mysql> SELECT FAILED_ATTEMPTS FROM INFORMATION_SCHEMA.CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS;
+-----------------+
| FAILED_ATTEMPTS |
+-----------------+
|               4 |
+-----------------+
```

Chapter 10 Block Encryption Mode Configuration

If you use the `AES_ENCRYPT()` encryption function, a block encryption mode with a `CBC` mode value and key length of 256 is recommended.

The `block_encryption_mode` variable controls the block encryption mode. The default setting is `aes-128-ecb`. Set this option to `aes-256-cbc`, for example, under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

```
block_encryption_mode=aes-256-cbc
```

When using the `AES_ENCRYPT()` function, an initialization vector (the `key_str` value) must be supplied. This value is required for decryption and should be managed carefully.

For more information about `block_encryption_mode` configuration, see the `AES_DECRYPT()` function description. For information about how block modes work, see Block cipher mode of operation.

Chapter 11 Enabling Authentication

When a client connects to the MySQL server, the server uses the user name provided by the client and the client host to select the account row from the `mysql.user` table. The server authenticates the client, determining from the account row which authentication plugin applies to the client. The server invokes that plugin to authenticate the user, and the plugin returns a status to the server indicating whether the user is permitted to connect.

This deployment uses the `caching_sha2_password` and `auth_socket` authentication plugins for user authentication.

Caching SHA-2 Authentication

In MySQL 8.0, `caching_sha2_password` is the default authentication plugin rather than `mysql_native_password`, which was the default in MySQL 5.7.

The server-side `caching_sha2_password` plugin is built into the server and it does not need to be loaded explicitly. Therefore, no server-side configuration is required to use it.

The client-side plugin is built into the `libmysqlclient` library (MySQL 8.0.4 and higher) and is available to any program linked against `libmysqlclient`. For a list of compatible clients and connectors, see caching_sha2_password-Compatible Clients and Connectors.

The `caching_sha2_password` plugin uses a SHA-2 algorithm that provides 256-bit password encryption. Passwords are salted with random data before SHA-256 transformations are applied. The resulting hashed values are stored in the `mysql.user` table. Using a salt helps defend against dictionary attacks on stored password hash values.

The `caching_sha2_password` plugin requires a secure connection (made using TLS credentials, a Unix socket file, or shared memory) or an unencrypted connection that supports password exchange using an RSA key pair. However, the performance cost associated with a secure connection is mitigated by the caching capability of the plugin. Once a hashed password is cached in memory, authentication can be performed over an unencrypted channel using a SHA256-based challenge-response mechanism, which means faster authentication for users that have connected previously.

> **Note**
>
> Changing a password, renaming a user, and `FLUSH PRIVILEGES` operations invalidate cached password hash values. When a cached password hash value is invalidated, a secure connection is required again for password exchange.

User accounts created later in this deployment use `caching_sha2_password` authentication. See Chapter 13, *Creating User Accounts*. TLS and RSA key pair connection methods are demonstrated in Chapter 14, *Connecting to the Server*.

For additional information about the `caching_sha2_password` plugin, see Caching SHA-2 Pluggable Authentication.

Socket Peer-Credential Authentication

This section describes how to enable the server-side `auth_socket` authentication plugin, which authenticates clients that connect to the MySQL server from the local host through the Unix socket file. `auth_socket` authentication is well suited to server administration user accounts for which access must be tightly restricted.

The `auth_socket` plugin checks whether the socket user name matches the MySQL user name specified by the client program to the server. If the names do not match, the plugin also checks whether the socket user name matches the name specified in the `authentication_string` column of the `mysql.user` table row. If a match is found, the plugin permits the connection.

For example, suppose that a MySQL account is created for a user named `valerie` who is to be authenticated by the `auth_socket` plugin for connections from the local host through the socket file:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
```

If a user on the local host with a login name of `stefanie` invokes `mysql` with the option `--user=valerie` to connect through the socket file, the server uses `auth_socket` to authenticate the client. The plugin determines that the `--user` option value (`valerie`) differs from the client user's name (`stephanie`) and refuses the connection. If a user named `valerie` tries the same thing, the plugin finds that the user name and the MySQL user name are both `valerie` and permits the connection. However, the plugin refuses the connection even for `valerie` if the connection is made using a different protocol, such as TCP/IP.

Users authenticated by the `auth_socket` need not specify a password when connecting to the server. However, users authenticated by the `auth_socket` plugin are restricted from connecting remotely; they can only connect from the local host through the Unix socket file.

To install the server-side `auth_socket` plugin:

1. Add these options under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

   ```
   plugin-load-add=auth_socket.so
   auth_socket=FORCE_PLUS_PERMANENT
   ```

   - `plugin-load-add=auth_socket.so`

     Loads the `auth_socket.so` plugin library each time the server is started.

   - `auth_socket=FORCE_PLUS_PERMANENT`

     Prevents the server from running without the `auth_socket` plugin, and server startup fails if the plugin does not initialize successfully.

2. To verify plugin installation, restart the server and examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement:

   ```
   shell> systemctl restart mysqld
   ```

   ```
   shell> cd /usr/local/mysql
   shell> bin/mysqladmin -u root -p version
   Enter password: (enter root password here)
   ```

   ```
   mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
          FROM INFORMATION_SCHEMA.PLUGINS
          WHERE PLUGIN_NAME LIKE '%socket%';
   +-------------------+---------------+
   | PLUGIN_NAME       | PLUGIN_STATUS |
   +-------------------+---------------+
   | auth_socket       | ACTIVE        |
   +-------------------+---------------+
   ```

3. Optionally, modify the MySQL root user account to use the `auth_socket` plugin for authentication:

   ```
   mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH auth_socket;
   ```

4. To verify that the `root@localhost` account is using the `auth_socket` plugin, issue this query:

   ```
   mysql> SELECT user, plugin FROM mysql.user WHERE user IN ('root')\G
   *************************** 1. row ***************************
     user: root
   plugin: auth_socket
   ```

5. To verify that the `auth_socket` plugin works, log in to the MySQL server host as the operating system root user and then connect to the MySQL server locally as the MySQL root user. You should be able to connect without specifying a password.

```
shell> cd /usr/local/mysql
shell> bin/mysql -u root
```

For more information about the `auth_socket` plugin, see Socket Peer-Credential Pluggable Authentication.

Chapter 12 Configuring MySQL to Use Secure Connections

This section describes configuring the server for secure connections and distributing client certificate and keys files.

- Configuring the Server for Secure Connections
- Distributing Client Certificate and Key Files

Configuring the Server for Secure Connections

1. MySQL requires certificate and key files to enable secure connections. By default, MySQL servers that are compiled using OpenSSL generate these files in the data directory at startup if they are not present. (MySQL Enterprise Edition is compiled using OpenSSL.) The only requirement is that the `--ssl` option is enabled, which it is by default, and no other `--ssl-*` options are specified.

   a. Check the data directory of the MySQL installation to verify that server and client certificate and key files were generated:

   ```
   shell> cd /usr/local/mysql/data
   shell> ls *.pem
   ca-key.pem  client-cert.pem  private_key.pem  server-cert.pem
   ca.pem      client-key.pem   public_key.pem   server-key.pem
   ```

   > **Important**
   >
   > Generation of certificate files by MySQL helps lower the barrier to using TLS. However, these certificates are self-signed, which is not very secure. After you gain experience using the files generated by MySQL, consider obtaining a CA certificate from a registered certificate authority.

   b. These options identify the certificate and key files the server uses when establishing a secure connection:

   - `ssl-ca=ca.pem`

     Identifies the Certificate Authority (CA) certificate.

   - `ssl-cert=server-cert.pem`

     Identifies the server public key certificate.

   - `ssl-key=server-key.pem`

     Identifies the server private key.

   To configure these options explicitly, add them under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

   ```
   ssl_ca=ca.pem
   ssl_cert=server-cert.pem
   ssl_key=server-key.pem
   ```

2. By default, the MySQL server accepts TCP/IP connections from MySQL user accounts on all server host IPv6 and IPv4 interfaces. You can make this configuration more restrictive by setting the `bind_address` configuration option to a specific IPv4 or IPv6 address so that the server only accepts TCP/IP connections on that address.

   For example, to have the MySQL server only accept connections on a specific IPv4 address, add an entry similar to this under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

```
bind_address=192.0.2.24
```

In this case, clients can connect to the server using `--host=192.0.2.24`. Connections on other server host addresses are not permitted.

For more information about `bind_address` configuration, see Server Command Options.

3. The `tls_version` option defines protocols permitted by the server for encrypted connections. To ensure that clients connect to the server using TLSv1.3, which provides greater security than earlier TLS versions, set `tls_version` to TLSv1.3. When compiled using OpenSSL 1.1.1 or higher, MySQL supports the TLSv1, TLSv1.1, TLSv1.2, and TLSv1.3 protocols.

   ```
   tls_version=TLSv1.3
   ```

   With this setting, only clients that support TLSv1.3 are able to establish an encrypted connection to the server. Support for the TLSv1.3 protocol is available as of MySQL 8.0.16 for MySQL servers and clients compiled with OpenSSL 1.1.1 or higher.

4. To further harden your deployment, you can use the `tls_ciphersuites` variable to limit the ciphersuites that the server permits for encrypted connections that use TLSv1.3. For example, to permit a single ciphersuite, add an entry similar to this under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

   ```
   tls_ciphersuites=TLS_AES_128_GCM_SHA256
   ```

   To specify more than one ciphersuite, separate ciphersuite names with colons.

   You can determine which ciphersuites a given server supports by establishing an encrypted connection to the server and issuing the following statement to check the value of the `Ssl_cipher_list` status variable:

   ```
   mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher_list'\G
   ```

   For encrypted connections that use TLSv1.3, OpenSSL 1.1.1 and higher supports the following ciphersuites, the first three of which are enabled by default:

   ```
   TLS_AES_128_GCM_SHA256
   TLS_AES_256_GCM_SHA384
   TLS_CHACHA20_POLY1305_SHA256
   TLS_AES_128_CCM_SHA256
   TLS_AES_128_CCM_8_SHA256
   ```

   > **Note**
   >
   > Ciphersuite support was added in MySQL 8.0.16 for encrypted connections that use TLSv1.3. If you use TLSv1.2 or lower, use the `--ssl-cipher` option to configure a specific cipher instead of using `tls_ciphersuites`.
   >
   > Optionally, you can use the `--tls-ciphersuites` option to limit the ciphersuites that client programs permit for encrypted connections that use TLSv1.3.

   For more information about ciphers and ciphersuites, see Encrypted Connection TLS Protocols and Ciphers.

5. Optionally, to require that all clients connect to the server securely, you can enable the `require_secure_transport` option. When this option is enabled, the server only permits TCP/IP connections that use TLS, or that use a socket file (on Unix) or shared memory (on Windows).

Connections that use insecure transport are prohibited, including unencrypted connections that use RSA key pair-based password exchange.

The `require_secure_transport` option is not used in this deployment so that RSA key pair-based password exchange over an unencrypted connection can be demonstrated. (See Using RSA Key Pair-Based Password Exchange Over an Unencrypted Connection.)

> **Note**
>
> Enabling `require_secure_transport` prevents TCP/IP connections that do not use TLS. Requiring all TCP/IP connections to use TLS may impact performance due to associated network and CPU costs.

6. Restart the server to apply the configuration changes:

```
shell> systemctl restart mysqld
```

Distributing Client Certificate and Key Files

Client certificate and key files are created in the MySQL data directory by default. Permissions for the data directory enable access only to the `mysql` account that runs the MySQL server, so client programs cannot use files located there. To make the files available to clients, either distribute the files to client hosts or place them on a mounted partition that is accessible to clients. The files should reside in a directory that is readable (but not writable) by the client. Use a secure channel when distributing the files to ensure they are not tampered with during transit.

The client certificate and key files to distribute include:

- `ca.pem` (CA certificate)
- `client-cert.pem` (Client certificate)
- `client-key.pem` (Client private key)

The `ca.pem`, `client-cert.pem`, and `client-key.pem` files are used later to establish an encrypted connection to the server.

Optionally, also distribute the RSA public key file (`public_key.pem`). For OpenSSL-compiled `mysql` clients that authenticate using the `sha256_password` plugin, this file is used for RSA key pair-based password exchange with the server over an unencrypted connection.

The location of the files on the client host or mounted partition is required later when connecting to the server.

Chapter 13 Creating User Accounts

This section describes how to create user accounts. It demonstrates configuring global password policies, using security-related `CREATE USER` options, granting user privileges, and verifying user privileges and authentication.

Two user accounts are created: `user1` and `user2`. The `user1` account is defined with an SSL/TLS option that requires an encrypted connection. The `user2` account is defined without an SSL/TLS option (`REQUIRE NONE`) so that it can be used to demonstrate RSA key pair-based password exchange with the server over an unencrypted connection.

1. Define global password history, reuse, expiration, and verification-required policies:

   a. A global password history policy is defined using the `password_history` system variable. The default setting is 0, which means that there is no restriction. To require a specified number of password changes before the same password can be reused, add an entry similar to this under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

   ```
   password_history=12
   ```

   A setting of `12` means that a minimum of 12 password changes must occur before a password can be reused.

   b. A global password reuse policy is defined using the `password_reuse_interval` system variable. The default setting is 0, which means that there is no restriction. To require that a specified number of days pass before the same password can be reused, add an entry similar to this under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

   ```
   password_reuse_interval=1095
   ```

   A setting of `1095` means that a minimum of 1095 days must pass before a password can be reused.

   c. A global automatic password expiration policy is defined using the `default_password_lifetime` system variable. The default setting is 0, which disables automatic password expiration. To have passwords automatically expire after a specified number of days, add an entry similar to this under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

   ```
   default_password_lifetime=120
   ```

   A setting of `120` means that the lifetime of a password is 120 days, after which it automatically expires.

   d. A global password verification-required policy is defined using the `password_require_current` system variable. The default setting is 0, which means that password changes do not require specifying the current password. To require that password changes specify the current password, add the following entry under the `[mysqld]` option group in the MySQL configuration file (`/etc/my.cnf`):

   ```
   password_require_current=1
   ```

   Restart the server to apply the configuration changes:

   ```
   shell> systemctl restart mysqld
   ```

2. Log in as root.

   ```
   shell> cd /usr/local/mysql
   shell> bin/mysql -u root -p
   Enter password: (enter root password here)
   ```

3. Create the user accounts. The following statements create user accounts named `'user1'@'203.0.113.11'` and `'user1'@'203.0.113.12'`, where *`203.0.113.11`* and *`203.0.113.12`* are the IP addresses of the client hosts. The statements include security-related options for enabling authentication, defining SSL/TSL requirements, generating a random password, limiting server resource usage, and managing password expiration.

```
mysql> CREATE USER 'user1'@'203.0.113.11' IDENTIFIED WITH caching_sha2_password BY
       RANDOM PASSWORD REQUIRE X509 WITH MAX_USER_CONNECTIONS 3 PASSWORD HISTORY DEFAULT
       PASSWORD REUSE INTERVAL DEFAULT PASSWORD EXPIRE DEFAULT PASSWORD REQUIRE CURRENT DEFAULT
       FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME UNBOUNDED;
+-------+-------------+----------------------+
| user  | host        | generated password   |
+-------+-------------+----------------------+
| user1 | 203.0.113.11 | e6<]aR3he*XPg3o6ML<7 |
+-------+-------------+----------------------+
```

```
mysql> CREATE USER 'user2'@'203.0.113.12' IDENTIFIED WITH caching_sha2_password BY
       RANDOM PASSWORD REQUIRE NONE WITH MAX_USER_CONNECTIONS 3 PASSWORD HISTORY DEFAULT
       PASSWORD REUSE INTERVAL DEFAULT PASSWORD EXPIRE DEFAULT PASSWORD REQUIRE CURRENT DEFAULT
       FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME UNBOUNDED;
+-------+-------------+----------------------+
| user  | host        | generated password   |
+-------+-------------+----------------------+
| user2 | 203.0.113.12 | VT@jNXB3@CvVB>/vMbke |
+-------+-------------+----------------------+
```

`CREATE USER` statement options:

- `IDENTIFIED WITH caching_sha2_password BY RANDOM PASSWORD`

  Sets the account authentication plugin to `sha256_password`, generates a random password that is passed as a cleartext value to the plugin for hashing, and stores the result in the `mysql.user` account row. The cleartext random password is also returned in a row of a result set (as shown above) to make it available to the user executing the statement.

  > **Note**
  >
  > The `RANDOM PASSWORD` option is used as an alternative to an administrator-specified literal password. By default, generated random passwords have a length of 20 characters. This length is controlled by the `generated_random_password_length` system variable, which has a range from 5 to 255. The default length is used in this deployment.
  >
  > If an administrator-specified literal password is specified instead of the `RANDOM PASSWORD` option, the literal *`password`* value must conform to the password policy enabled by the `validate_password` component. (See Chapter 6, *Installing the MySQL Password Validation Component*.) The policy that the `validate_password` component implements has no effect on generated passwords. The purpose of a `validate_password` policy is to help *humans* create better passwords.

  For more information, see CREATE USER Authentication Options.

- `REQUIRE X509`

  This SSL/TLS option is only used for the `user1` account.

  MySQL can check X509 certificate attributes in addition to the usual authentication that is based on the user name and credentials. Available SSL/TSL options include `SSL`, `X509`, `ISSUER`, `SUBJECT`, and `CIPHER`. The `CREATE USER` statement for `user1` uses the `X509` option, which requires that clients present a valid certificate, but the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the

CA certificates. Use of `X509` certificates always implies encryption, so it is unnecessary to also specify the `SSL` option.

For more information, see CREATE USER SSL/TLS Options.

- `REQUIRE NONE`

  Indicates that the account has no `TLS` or `X509` requirements. Unencrypted connections are permitted if the user name and password are valid. Encrypted connections can be used, at the client's option, if the client has the proper certificate and key files. `NONE` is the default if no SSL-related `REQUIRE` options are specified.

  For more information, see CREATE USER SSL/TLS Options.

- `MAX_USER_CONNECTIONS 3`

  Restricts the maximum number of simultaneous connections to the server by the account. If the number is 0 (the default), the server determines the number of simultaneous connections for the account from the global value of the `max_user_connections` system variable. `MAX_USER_CONNECTIONS 3` means that the account can have a maximum of 3 simultaneous connections to the server.

  Other resource-limiting options not used here include `MAX_QUERIES_PER_HOUR`, `MAX_UPDATES_PER_HOUR`, and `MAX_CONNECTIONS_PER_HOUR`. For more information, see CREATE USER Resource-Limit Options.

- `PASSWORD HISTORY DEFAULT`

  Applies the global password history policy defined by the `password_history` system variable. In an earlier step, `password_history` was set to `12` to require that 12 password changes occur before the same password can be reused.

- `PASSWORD REUSE INTERVAL DEFAULT`

  Applies the global password reuse policy defined by the `password_reuse_interval` system variable. In an earlier step, `password_reuse_interval` was set to `1095` to require that 1095 days pass before the same password can be reused.

- `PASSWORD EXPIRE DEFAULT`

  Applies the global automatic password expiration policy defined by the `default_password_lifetime` system variable. In an earlier step, `default_password_lifetime` was set to 120 so that passwords automatically expire every 120 days.

  Other password expiration options include `PASSWORD EXPIRE`, `PASSWORD EXPIRE INTERVAL`, and `PASSWORD EXPIRE NEVER`. For more information, see CREATE USER Password-Management Options.

- `PASSWORD REQUIRE CURRENT DEFAULT`

  Causes the account to defer to the global password verification-required policy defined by the `password_require_current` system variable. In an earlier step, `password_require_current` was enabled to require that password changes must specify the current password.

  Verification of the current password occurs when a user changes a password using the `ALTER USER` or `SET PASSWORD` statement. For more information, see Password Verification-Required Policy.

- `FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME UNBOUNDED`

  The `FAILED_LOGIN_ATTEMPTS` option defines how many consecutive incorrect passwords cause the account to become locked. The `PASSWORD_LOCK_TIME` option defines how long the account is locked after too many consecutive login attempts provide an incorrect password. `PASSWORD_LOCK_TIME` can be set to a number of days or to `UNBOUNDED`, which specifies that the duration of locked state is unbounded and does not end until the account is unlocked.

  For more information, including the conditions under which unlocking occurs, see Failed-Login Tracking and Temporary Account Locking.

4. Grant user privileges. The following statements grant the `SHOW DATABASES` privilege to the `user1` and `user2` accounts:

```
mysql> GRANT SHOW DATABASES ON *.* TO 'user1'@'203.0.113.11';
```

```
mysql> GRANT SHOW DATABASES ON *.* TO 'user2'@'203.0.113.12';
```

> **Note**
>
> The privileges granted to a MySQL account determine which operations the account can perform. Following the principle of least privilege, a MySQL account should only be granted privileges required for its legitimate purposes. To facilitate effective privilege management, MySQL 8.0 provides two new privilege-related features: *MySQL Roles* and *Dynamic Privileges*. For information about these features, see Appendix D, *SQL Roles and Dynamic Privileges*.

5. To verify the privileges granted to the user accounts, issue a `SHOW GRANTS` statement. For example:

```
mysql> SHOW GRANTS FOR 'user1'@'203.0.113.11';
+-----------------------------------------------------------------------------------+
| Grants for user1@203.0.113.11                                                     |
+-----------------------------------------------------------------------------------+
| GRANT SHOW DATABASES ON *.* TO 'user1'@'203.0.113.11'                             |
+-----------------------------------------------------------------------------------+
```

6. To verify that the accounts are using the expected authentication plugin, issue this query:

```
mysql> SELECT user, plugin FROM mysql.user WHERE user LIKE ('user%')\G
*************************** 1. row ***************************
  user: user1
plugin: caching_sha2_password
*************************** 2. row ***************************
  user: user2
plugin: caching_sha2_password
```

Chapter 14 Connecting to the Server

This section describes two connection methods. The first method uses the TLS (Transport Layer Security) protocol to establish an encrypted connection. The second method uses RSA key pair-based password exchange over an unencrypted connection.

The following procedures assume that `mysql` clients are available on remote client hosts. The procedures also assume that you have distributed client certificate and key files to the remote clients as described in Distributing Client Certificate and Key Files.

The user accounts created previously are used to connect to the server. See Chapter 13, *Creating User Accounts*.

Using an Encrypted Connection

MySQL client programs attempt to establish an encrypted connection if the server supports encrypted connections. In this deployment, the `--ssl` option is enabled for the server, which means encrypted connections are supported.

1. Using the `mysql` client program, establish a connection for the `user1@203.0.113.11` account that you created previously. The `user1@203.0.113.11` account was created with the `REQUIRE X509` option, which requires that the user presents a valid certificate.

   ```
   shell> cd /usr/local/mysql
   shell> bin/mysql  --user=user1 -p --host=192.0.2.24 --ssl-mode=VERIFY_CA
          --ssl-ca=/path/to/ca.pem --ssl-cert=/path/to/client-cert.pem
          --ssl-key=/path/to/client-key.pem
   ```

   - The `--host` option specifies the host where the MySQL server is running.
   - The `--ssl-mode=VERIFY_CA` option ensures that an encrypted connection is established and verifies the TLS certificate against the configured Certificate Authority (CA) certificates; it ensures that client and server trust a common CA and thus are likely communicating with the correct party.

     > **Note**
     >
     > Ideally, `--ssl-mode` should be set to `VERIFY_IDENTITY`. This option is like `VERIFY_CA` but it additionally requires that the server certificate matches the host to which the connection is attempted, which means that the server certificate must be signed by a valid Certificate Authority (CA) and have your server host as the Common Name (CN). The MySQL-generated certificates used in this deployment do not support this mode.

   - The `--ssl-ca`, `--ssl-cert`, and `--ssl-key` options define the path to the distributed client certificate and key files, as described in Distributing Client Certificate and Key Files.

2. After connecting successfully, verify that the current connection uses encryption by checking the value of the `Ssl_cipher` status variable. If the value is empty, the connection is not encrypted. Otherwise, the connection is encrypted and the value indicates the encryption cipher or ciphersuite. For example:

   ```
   mysql> SHOW STATUS LIKE 'Ssl_cipher';
   +---------------+------------------------+
   | Variable_name | Value                  |
   +---------------+------------------------+
   | Ssl_cipher    | TLS_AES_128_GCM_SHA256 |
   +---------------+------------------------+
   ```

3. To view the TLS version and the cipher or ciphersuite for all connections, query the `Sys` schema `session_ssl_status` view as the MySQL root user:

```
shell> cd /usr/local/mysql
shell> bin/mysql -u root -p
Enter password: (enter the root password here)
```

```
mysql> SELECT * FROM sys.session_ssl_status;
+-----------+-------------+-----------------------+--------------------+
| thread_id | ssl_version | ssl_cipher            | ssl_sessions_reused |
+-----------+-------------+-----------------------+--------------------+
|        51 |             |                       | 0                  |
|        52 | TLSv1.3     | TLS_AES_128_GCM_SHA256 | 0                  |
+-----------+-------------+-----------------------+--------------------+
```

For more information about encrypted connections, see Configuring MySQL to Use Encrypted Connections.

Using RSA Key Pair-Based Password Exchange Over an Unencrypted Connection

Clients that authenticate using the `caching_sha2_password` plugin can connect to the server over an unencrypted connection using RSA key pair-based password exchange. (Both the client and server must be compiled using OpenSSL.)

To support RSA encryption, the server generates RSA public and private key files in the data directory:

```
shell> cd /usr/local/mysql/data
shell> ls *_key.pem
private_key.pem  public_key.pem
```

By default, the server also exposes variables for defining the RSA private key and public key paths:

- `caching_sha2_password_private_key_path`

  Defines the path name of the RSA private key file for the `caching_sha2_password` authentication plugin.

- `caching_sha2_password_public_key_path`

  Defines the path name of the RSA public key file for the `caching_sha2_password` authentication plugin.

If the RSA public key and private key files are located in the MySQL data directory and are named `private_key.pem` and `public_key.pem`, as they are in this deployment, the `caching_sha2_password_private_key_path` and `caching_sha2_password_private_key_path` options are configured by default.

When a client that uses the `caching_sha2_password` plugin attempts an unencrypted connection, the `caching_sha2_password` plugin sends the RSA public key to the client, but the key transfer can be avoided if the RSA public key is distributed to the client host and its location is defined using the `--server-public-key-path` option when establishing a connection. Avoiding the key transfer saves a round trip in the client/server protocol. This option is used in the instructions that follow. For information about distributing key files, see Distributing Client Certificate and Key Files.

To establish an unencrypted connection that uses RSA key pair-based password exchange, use the `mysql` client program and the `user2@203.0.113.12` account that you created previously. The `user2@203.0.113.12` account was created without SSL/TLS options to permit the account to establish an unencrypted connection to the server.

```
shell> cd /usr/local/mysql
shell> bin/mysql  --user=user2 -p --ssl-mode=DISABLED --host=192.0.2.24
       --server-public-key-path=/path/to/public_key.pem
```

- The `--host` option specifies the host where the MySQL server is running.

- The `--ssl-mode=DISABLE` option ensures that the connection is unencrypted.
- The `--server-public-key-path` option defines the path name to the file on the client host (`public_key.pem`) that contains the same RSA public key used by the server.

Appendix A Transparent Data Encryption (TDE) and MySQL Keyring

MySQL Server supports Transparent Data Encryption (TDE), which protects critical data by enabling data-at-rest encryption. Data-at-rest encryption is supported by the MySQL Keyring feature, which provides plugin-based support for key management solutions such as:

- Oracle Key Vault
- Gemalto SafeNet KeySecure Appliance
- Thales Vormetric Key Management Server
- Fornetix Key Orchestration
- Amazon Web Services Key Management Service
- Hashicorp Vault

For information about the MySQL Keyring feature and supported plugins, see The MySQL Keyring.

After a keyring plugin is installed and configured, encryption can be enabled for:

- File-per-table tablespaces
- General tablespaces
- The `mysql` system tablespace
- Redo logs
- Undo logs

For more information, see InnoDB Data-at-Rest Encryption.

Encryption is also supported for:

- Binary log files and relay log files. See Encrypting Binary Log Files and Relay Log Files.
- Audit log files. See Encrypting Audit Log Files.
- Backups. See Encryption for Backups, and Working with Encrypted InnoDB Tablespaces.

Appendix B Data Masking and De-Identification

As of MySQL 8.0.13, MySQL Enterprise Edition provides data masking and de-identification capabilities, which permit:

- Transforming existing data to mask it and remove identifying characteristics, such as changing all digits of a credit card number but the last four to `'X'` characters.
- Generating random data, such as email addresses and payment card numbers.

The way that applications use these capabilities depends on the purpose for which the data will be used and who will access it:

- Applications that use sensitive data may protect it by performing data masking and permitting use of partially masked data for client identification.
- Applications that require properly formatted data, but not necessarily the original data, can synthesize sample data.

MySQL Enterprise Data Masking and De-Identification is implemented as a plugin library file that contains these components:

- A server-side plugin named `data_masking`.
- A set of user-defined functions (UDFs) that provide an SQL-level API for performing masking and de-identification operations.

MySQL Enterprise Data Masking and De-Identification can help application developers satisfy privacy requirements that are core to regulatory compliance.

For more information about the components of MySQL Enterprise Data Masking and De-Identification, and how to install and use them, see MySQL Enterprise Data Masking and De-Identification.

Appendix C FIPS Support

MySQL 8.0 supports FIPS (Federal Information Processing Standards) mode, if compiled using OpenSSL, and a FIPS-enabled OpenSSL library and FIPS Object Module are available at runtime. FIPS mode imposes conditions on cryptographic operations such as restrictions on acceptable encryption algorithms or requirements for longer key lengths. The `ssl_fips_mode` system variable enables control of FIPS mode on the server side. The `--ssl-fips-mode` client option enables control of FIPS mode on the client side. Both options are disabled by default.

FIPS mode has been tested for MySQL on EL7, but may work on other systems.

For more information about this security feature, see FIPS Support in the *MySQL Reference Manual*.

Appendix D SQL Roles and Dynamic Privileges

Privilege management is an important aspect of managing the security of a MySQL installation. Following the principle of least privilege, a MySQL account should only be granted privileges required to for its legitimate purposes. To facilitate effective privilege management, MySQL 8.0 provides two new privilege-related features: *MySQL Roles* and *Dynamic Privileges*.

- MySQL roles are named collections of privileges. A user account can be granted roles, which grants to the account the privileges associated with each role. This enables assignment of sets of privileges to accounts and provides a convenient alternative to granting individual privileges, both for conceptualizing desired privilege assignments and implementing them.

  For more information about using roles to manage account privileges, see Using Roles.

- Dynamic privileges enable DBAs to begin migrating away from the `SUPER` privilege. Many operations covered by `SUPER` are associated with a dynamic privilege of more limited scope. Operations that previously required the `SUPER` privilege can be permitted to an account by granting the associated dynamic privilege rather than `SUPER`. For example, a user who must be able to modify global system variables can be granted `SYSTEM_VARIABLES_ADMIN` rather than `SUPER`.

  This change improves security by enabling DBAs to avoid granting `SUPER` and tailor user privileges more closely to the operations permitted. The `SUPER` privilege is deprecated and will be removed in a future version of MySQL.

  For more information about this feature, see Static Versus Dynamic Privileges. That discussion includes instructions for migrating accounts away from `SUPER` to dynamic privileges.

Appendix E Installation Directory and File Permissions

The following table shows directory and file permissions for the generic binary distribution installation of *MySQL Enterprise Edition for Linux x86-64* on *Oracle Linux* that is described in this guide.

As a general rule, distributed files and resources should follow the principal of least privilege, which requires that users, processes, programs, and other system components only have access to information and resources that are required for their legitimate purpose.

As indicated previously, most of the MySQL installation can be owned by `root`. The exceptions are the data directory, the error log file, the `mysql-files` directory, the pid file, and the socket file, to which the `mysql` user must have write access. Files and resources that the `mysql` user requires read access to include configuration files (`/etc/my.cnf`) and the MySQL binaries (`/usr/local/mysql/bin`).

Table E.1 MySQL Linux Generic Binary Installation Directory and File Permissions

File or Resource	Location	Owner	Directory Permissions	File Permissions
Client and utility programs directory	/usr/local/mysql/bin	root	drwxr-xr-x	
mysqld server	/usr/local/mysql/bin	root	drwxr-xr-x	-rwxr-xr-x
MySQL configuration file	/etc/my.cnf	root	drwxr-xr-x	-rw-r--r--
Data directory	/usr/local/mysql/data	mysql	drwxr-x---	
Error log file	/usr/local/mysql/data/host_name.err	mysql	drwxr-x---	-rw-------
`secure_file_priv` directory	/usr/local/mysql/mysql-files	mysql	drwxr-x---	
mysqld systemd service file	/usr/lib/systemd/system/mysqld.service	root	drwxr-xr-x	-rw-r--r--
systemd tmpfiles configuration file	/usr/lib/tmpfiles.d/mysql.conf	root	drwxr-xr-x	-rw-r--r--
pid file	/usr/local/mysql/data/mysqld.pid	mysql	drwxr-x---	-rw-r-----
socket file	/tmp/mysql.sock	mysql	drwxrwxrwt	srwxrwxrwx
Unix manual pages directory	/usr/local/mysql/man	root	drwxr-xr-x	
Include Header files directory	/usr/local/mysql/include	root	drwxr-xr-x	
Libraries directory	/usr/local/mysql/lib	root	drwxr-xr-x	
Miscellaneous support files directory	/usr/local/mysql/support-files	root	drwxr-xr-x	
Miscellaneous files directory	/usr/local/mysql/share	root	drwxr-xr-x	

Appendix F Deployment Configuration File

Upon the completion of the deployment described in this guide, the MySQL configuration file (`/etc/my.cnf`) contains these configuration settings:

```
[mysqld]
datadir=/usr/local/mysql/data
socket=/tmp/mysql.sock
port=3306
log-error=/usr/local/mysql/data/localhost.localdomain.err
user=mysql
secure_file_priv=/usr/local/mysql/mysql-files
local_infile=OFF
validate_password.policy=1
validate_password.length=8
validate_password.number_count=1
validate_password.mixed_case_count=1
validate_password.special_char_count=1
validate_password.check_user_name=1
audit-log=FORCE_PLUS_PERMANENT
mysql_firewall_mode=ON
plugin-load-add=connection_control.so
connection-control=FORCE_PLUS_PERMANENT
connection-control-failed-login-attempts=FORCE_PLUS_PERMANENT
connection_control_failed_connections_threshold=3
connection_control_min_connection_delay=1000
connection_control_max_connection_delay=2147483647
block_encryption_mode=aes-256-cbc
default_authentication_plugin=caching_sha2_password
plugin-load-add=auth_socket.so
auth_socket=FORCE_PLUS_PERMANENT
ssl_ca=ca.pem
ssl_cert=client-cert.pem
ssl_key=client-key.pem
bind_address=192.0.2.24
tls_version=TLSv1.3
tls_ciphersuite=TLS_AES_128_GCM_SHA256
default_password_lifetime=120
password_history=12
password_reuse_interval=1095
generated_random_password_length=20
password_require_current=1
```