```
| USER()          | CURRENT_USER()          | @@proxy_user |
+-----------------+-------------------------+--------------+
| basil@localhost | front_office@localhost | ''@'%'       |
+-----------------+-------------------------+--------------+
```

This demonstrates that `basil` uses the privileges granted to the proxied `front_office` MySQL account, and that proxying occurs through the default proxy user account.

LDAP Authentication Group Preference and Mapping Specification

As described in LDAP Authentication with Proxying, basic LDAP authentication proxying works by the principle that the plugin uses the first group name returned by the LDAP server as the MySQL proxied user account name. This simple capability does not enable specifying any preference about which group name to use if the LDAP server returns multiple group names, or specifying any name other than the group name as the proxied user name.

As of MySQL 8.0.14, for MySQL accounts that use LDAP authentication, the authentication string can specify the following information to enable greater proxying flexibility:

- A list of groups in preference order, such that the plugin uses the first group name in the list that matches a group returned by the LDAP server.
- A mapping from group names to proxied user names, such that a group name when matched can provide a specified name to use as the proxied user. This provides an alternative to using the group name as the proxied user.

Consider the following MySQL proxy account definition:

```
CREATE USER ''@'%'
  IDENTIFIED WITH authentication_ldap_sasl
  AS '+ou=People,dc=example,dc=com#grp1=usera,grp2,grp3=userc';
```

The authentication string has a user DN suffix `ou=People,dc=example,dc=com` prefixed by the `+` character. Thus, as described in LDAP Authentication User DN Suffixes, the full user DN is constructed from the user DN suffix as specified, plus the client user name as the `uid` attribute.

The remaining part of the authentication string begins with `#`, which signifies the beginning of group preference and mapping information. This part of the authentication string lists group names in the order `grp1`, `grp2`, `grp3`. The LDAP plugin compares that list with the set of group names returned by the LDAP server, looking in list order for a match against the returned names. The plugin uses the first match, or if there is no match, authentication fails.

Suppose that the LDAP server returns groups `grp3`, `grp2`, and `grp7`. The LDAP plugin uses `grp2` because it is the first group in the authentication string that matches, even though it is not the first group returned by the LDAP server. If the LDAP server returns `grp4`, `grp2`, and `grp1`, the plugin uses `grp1` even though `grp2` also matches. `grp1` has a precedence higher than `grp2` because it is listed earlier in the authentication string.

Assuming that the plugin finds a group name match, it performs mapping from that group name to the MySQL proxied user name, if there is one. For the example proxy account, mapping occurs as follows:

- If the matching group name is `grp1` or `grp3`, those are associated in the authentication string with user names `usera` and `userc`, respectively. The plugin uses the corresponding associated user name as the proxied user name.
- If the matching group name is `grp2`, there is no associated user name in the authentication string. The plugin uses `grp2` as the proxied user name.

If the LDAP server returns a group in DN format, the LDAP plugin parses the group DN to extract the group name from it.

To specify LDAP group preference and mapping information, these principles apply:

- Begin the group preference and mapping part of the authentication string with a # prefix character.
- The group preference and mapping specification is a list of one or more items, separated by commas. Each item has the form *group_name=user_name* or *group_name*. Items should be listed in group name preference order. For a group name selected by the plugin as a match from set of group names returned by the LDAP server, the two syntaxes differ in effect as follows:
  - For an item specified as *group_name=user_name* (with a user name), the group name maps to the user name, which is used as the MySQL proxied user name.
  - For an item specified as *group_name* (with no user name), the group name is used as the MySQL proxied user name.
- To quote a group or user name that contains special characters such as space, surround it by double quote (") characters. For example, if an item has group and user names of `my group name` and `my user name`, it must be written in a group mapping using quotes:

  ```
  "my group name"="my user name"
  ```

  If an item has group and user names of `my_group_name` and `my_user_name` (which contain no special characters), it may but need not be written using quotes. Any of the following are valid:

  ```
  my_group_name=my_user_name
  my_group_name="my_user_name"
  "my_group_name"=my_user_name
  "my_group_name"="my_user_name"
  ```

- To escape a character, precede it by a backslash (\). This is useful particularly to include a literal double quote or backslash, which are otherwise not included literally.
- A user DN need not be present in the authentication string, but if present, it must precede the group preference and mapping part. A user DN can be given as a full user DN, or as a user DN suffix with a + prefix character. (See LDAP Authentication User DN Suffixes.)

LDAP Authentication User DN Suffixes

LDAP authentication plugins permit the authentication string that provides user DN information to begin with a + prefix character:

- In the absence of a + character, the authentication string value is treated as is without modification.
- If the authentication string begins with +, the plugin constructs the full user DN value from the user name sent by the client, together with the DN specified in the authentication string (with the + removed). In the constructed DN, the client user name becomes the value of the attribute that specifies LDAP user names. This is `uid` by default; to change the attribute, modify the appropriate system variable (`authentication_ldap_simple_user_search_attr` or `authentication_ldap_sasl_user_search_attr`). The authentication string is stored as given in the `mysql.user` system table, with the full user DN constructed on the fly before authentication.

This account authentication string does not have + at the beginning, so it is taken as the full user DN:

```
CREATE USER 'baldwin'
  IDENTIFIED WITH authentication_ldap_simple
  AS 'uid=admin,ou=People,dc=example,dc=com';
```

The client connects with the user name specified in the account (`baldwin`). In this case, that name is not used because the authentication string has no prefix and thus fully specifies the user DN.

This account authentication string does have + at the beginning, so it is taken as just part of the user DN:

```
CREATE USER 'accounting'
  IDENTIFIED WITH authentication_ldap_simple
```

```
AS '+ou=People,dc=example,dc=com';
```

The client connects with the user name specified in the account (`accounting`), which in this case is used as the `uid` attribute together with the authentication string to construct the user DN: `uid=accounting,ou=People,dc=example,dc=com`

The accounts in the preceding examples have a nonempty user name, so the client always connects to the MySQL server using the same name as specified in the account definition. If an account has an empty user name, such as the default anonymous `''@'%'` proxy account described in LDAP Authentication with Proxying, clients might connect to the MySQL server with varying user names. But the principle is the same: If the authentication string begins with `+`, the plugin uses the user name sent by the client together with the authentication string to construct the user DN.

LDAP Authentication Methods

The LDAP authentication plugins use a configurable authentication method. The appropriate system variable and available method choices are plugin-specific:

- For the `authentication_ldap_simple` plugin: Set the `authentication_ldap_simple_auth_method_name` system variable to configure the method. The permitted choices are `SIMPLE` and `AD-FOREST`.
- For the `authentication_ldap_sasl` plugin: Set the `authentication_ldap_sasl_auth_method_name` system variable to configure the method. The permitted choices are `SCRAM-SHA-1`, `SCRAM-SHA-256`, and `GSSAPI`. (To determine which SASL LDAP methods are actually available on the host system, check the value of the `Authentication_ldap_sasl_supported_methods` status variable.)

See the system variable descriptions for information about each permitted method. Also, depending on the method, additional configuration may be needed, as described in the following sections.

The GSSAPI/Kerberos Authentication Method

Generic Security Service Application Program Interface (GSSAPI) is a security abstraction interface. Kerberos is an instance of a specific security protocol that can be used through that abstract interface. Using GSSAPI, applications authenticate to Kerberos to obtain service credentials, then use those credentials in turn to enable secure access to other services.

One such service is LDAP, which is used by the client-side and server-side SASL LDAP authentication plugins. When the `authentication_ldap_sasl_auth_method_name` system variable is set to `GSSAPI`, these plugins use the GSSAPI/Kerberos authentication method. In this case, the plugins communicate securely using Kerberos without using LDAP messages directly. The server-side plugin then communicates with the LDAP server to interpret LDAP authentication messages and retrieve LDAP groups.

GSSAPI/Kerberos is supported as an LDAP authentication method for MySQL servers and clients on Linux. It is useful in Linux environments where applications have access to LDAP through Microsoft Active Directory, which has Kerberos enabled by default.

The following discussion provides information about the configuration requirements for using the GSSAPI method. Familiarity is assumed with Kerberos concepts and operation. The following list briefly defines several common Kerberos terms. You may also find the Glossary section of RFC 4120 helpful.

- Principal: A named entity, such as a user or server.
- KDC: The key distribution center, comprising the AS and TGS:
  - AS: The authentication server; provides the initial ticket-granting ticket needed to obtain additional tickets.

  - TGS: The ticket-granting server; provides additional tickets to Kerberos clients that possess a valid TGT.
- TGT: The ticket-granting ticket; presented to the TGS to obtain service tickets for service access.

LDAP authentication using Kerberos requires both a KDC server and an LDAP server. This requirement can be satisfied in different ways:

- Active Directory includes both servers, with Kerberos authentication enabled by default in the Active Directory LDAP server.
- OpenLDAP provides an LDAP server, but a separate KDC server may be needed, with additional Kerberos setup required.

Kerberos must also be available on the client host. A client contacts the AS using a password to obtain a TGT. The client then uses the TGT to obtain access from the TGS to other services, such as LDAP.

The following sections discuss the configuration steps to use GSSAPI/Kerberos for SASL LDAP authentication in MySQL:

- Verify Kerberos and LDAP Availability
- Configure the Server-Side SASL LDAP Authentication Plugin for GSSAPI/Kerberos
- Create a MySQL Account That Uses GSSAPI/Kerberos for LDAP Authentication
- Use the MySQL Account to Connect to the MySQL Server
- Client Configuration Parameters for LDAP Authentication

Verify Kerberos and LDAP Availability

The following example shows how to test availability of Kerberos in Active Directory. The example makes these assumptions:

- Active Directory is running on the host named `ldap_auth.example.com` with IP address `198.51.100.10`.
- MySQL-related Kerberos authentication and LDAP lookups use the `MYSQL.LOCAL` domain.
- A principal named `bredon@MYSQL.LOCAL` is registered with the KDC. (In later discussion, this principal name is also associated with the MySQL account that authenticates to the MySQL server using GSSAPI/Kerberos.)

With those assumptions satisfied, follow this procedure:

1. Verify that the Kerberos library is installed and configured correctly in the operating system. For example, to configure a `MYSQL.LOCAL` domain for use during MySQL authentication, the `/etc/krb5.conf` Kerberos configuration file should contain something like this:

   ```
   [realms]
     MYSQL.LOCAL = {
       kdc = ldap_auth.example.com
       admin_server = ldap_auth.example.com
       default_domain = MYSQL.LOCAL
     }
   ```

2. You may need to add an entry to `/etc/hosts` for the server host:

   ```
   198.51.100.10 ldap_auth ldap_auth.example.com
   ```

3. Check whether Kerberos authentication works correctly:

a. Use `kinit` to authenticate to Kerberos:

```
$> kinit bredon@MYSQL.LOCAL
Password for bredon@MYSQL.LOCAL: (enter password here)
```

The command authenticates for the Kerberos principal named `bredon@MYSQL.LOCAL`. Enter the principal's password when the command prompts for it. The KDC returns a TGT that is cached on the client side for use by other Kerberos-aware applications.

b. Use `klist` to check whether the TGT was obtained correctly. The output should be similar to this:

```
$> klist
Ticket cache: FILE:/tmp/krb5cc_244306
Default principal: bredon@MYSQL.LOCAL

Valid starting       Expires              Service principal
03/23/2021 08:18:33  03/23/2021 18:18:33  krbtgt/MYSQL.LOCAL@MYSQL.LOCAL
```

4. Check whether `ldapsearch` works with the Kerberos TGT using this command, which searches for users in the `MYSQL.LOCAL` domain:

```
ldapsearch -h 198.51.100.10 -Y GSSAPI -b "dc=MYSQL,dc=LOCAL"
```

Configure the Server-Side SASL LDAP Authentication Plugin for GSSAPI/Kerberos

Assuming that the LDAP server is accessible through Kerberos as just described, configure the server-side SASL LDAP authentication plugin to use the GSSAPI/Kerberos authentication method. (For general LDAP plugin installation information, see Installing LDAP Pluggable Authentication.) Here is an example of plugin-related settings the server `my.cnf` file might contain:

```
[mysqld]
plugin-load-add=authentication_ldap_sasl.so
authentication_ldap_sasl_auth_method_name="GSSAPI"
authentication_ldap_sasl_server_host=198.51.100.10
authentication_ldap_sasl_server_port=389
authentication_ldap_sasl_bind_root_dn="cn=admin,cn=users,dc=MYSQL,dc=LOCAL"
authentication_ldap_sasl_bind_root_pwd="password"
authentication_ldap_sasl_bind_base_dn="cn=users,dc=MYSQL,dc=LOCAL"
authentication_ldap_sasl_user_search_attr="sAMAccountName"
```

Those option file settings configure the SASL LDAP plugin as follows:

- The `--plugin-load-add` option loads the plugin (adjust the `.so` suffix for your platform as necessary). If you loaded the plugin previously using an `INSTALL PLUGIN` statement, this option is unnecessary.
- `authentication_ldap_sasl_auth_method_name` must be set to `GSSAPI` to use GSSAPI/Kerberos as the SASL LDAP authentication method.
- `authentication_ldap_sasl_server_host` and `authentication_ldap_sasl_server_port` indicate the IP address and port number of the Active Directory server host for authentication.
- `authentication_ldap_sasl_bind_root_dn` and `authentication_ldap_sasl_bind_root_pwd` configure the root DN and password for group search capability. This capability is required, but users may not have privileges to search. In such cases, it is necessary to provide root DN information:
  - In the DN option value, `admin` should be the name of an administrative LDAP account that has privileges to perform user searches.
  - In the password option value, *`password`* should be the `admin` account password.

- `authentication_ldap_sasl_bind_base_dn` indicates the user DN base path, so that searches look for users in the `MYSQL.LOCAL` domain.
- `authentication_ldap_sasl_user_search_attr` specifies a standard Active Directory search attribute, `sAMAccountName`. This attribute is used in searches to match logon names; attribute values are not the same as the user DN values.

Create a MySQL Account That Uses GSSAPI/Kerberos for LDAP Authentication

MySQL authentication using the SASL LDAP authentication plugin with the GSSAPI/Kerberos method is based on a user that is a Kerberos principal. The following discussion uses a principal named `bredon@MYSQL.LOCAL` as this user, which must be registered in several places:

- The Kerberos administrator should register the user name as a Kerberos principal. This name should include a domain name. Clients use the principal name and password to authenticate with Kerberos and obtain a TGT.
- The LDAP administrator should register the user name in an LDAP entry. For example:

```
uid=bredon,dc=MYSQL,dc=LOCAL
```

Note

In Active Directory (which uses Kerberos as the default authentication method), creating a user creates both the Kerberos principal and the LDAP entry.

- The MySQL DBA should create an account that has the Kerberos principal name as the user name and that authenticates using the SASL LDAP plugin.

Assume that the Kerberos principal and LDAP entry have been registered by the appropriate service administrators, and that, as previously described in Installing LDAP Pluggable Authentication, and Configure the Server-Side SASL LDAP Authentication Plugin for GSSAPI/Kerberos, the MySQL server has been started with appropriate configuration settings for the server-side SASL LDAP plugin. The MySQL DBA then creates a MySQL account that corresponds to the Kerberos principal name, including the domain name.

Note

The SASL LDAP plugin uses a constant user DN for Kerberos authentication and ignores any user DN configured from MySQL. This has certain implications:

- For any MySQL account that uses GSSAPI/Kerberos authentication, the authentication string in `CREATE USER` or `ALTER USER` statements should contain no user DN because it has no effect.
- Because the authentication string contains no user DN, it should contain group mapping information, to enable the user to be handled as a proxy user that is mapped onto the desired proxied user. For information about proxying with the LDAP authentication plugin, see LDAP Authentication with Proxying.

The following statements create a proxy user named `bredon@MYSQL.LOCAL` that assumes the privileges of the proxied user named `proxied_krb_usr`. Other GSSAPI/Kerberos users that should have the same privileges can similarly be created as proxy users for the same proxied user.

```
-- create proxy account
CREATE USER 'bredon@MYSQL.LOCAL'
  IDENTIFIED WITH authentication_ldap_sasl
  BY '#krb_grp=proxied_krb_user';

-- create proxied account and grant its privileges;
-- use mysql_no_login plugin to prevent direct login
CREATE USER 'proxied_krb_user'
  IDENTIFIED WITH mysql_no_login;
```

```
GRANT ALL
  ON krb_user_db.*
  TO 'proxied_krb_user';

-- grant to proxy account the
-- PROXY privilege for proxied account
GRANT PROXY
  ON 'proxied_krb_user'
  TO 'bredon@MYSQL.LOCAL';
```

Observe closely the quoting for the proxy account name in the first `CREATE USER` statement and the `GRANT PROXY` statement:

- For most MySQL accounts, the user and host are separate parts of the account name, and thus are quoted separately as `'user_name'@'host_name'`.
- For LDAP Kerberos authentication, the user part of the account name includes the principal domain, so `'bredon@MYSQL.LOCAL'` is quoted as a single value. Because no host part is given, the full MySQL account name uses the default of `'%'` as the host part: `'bredon@MYSQL.LOCAL'@'%'`

Note

When creating an account that authenticates using the `authentication_ldap_sasl` SASL LDAP authentication plugin with the GSSAPI/Kerberos authentication method, the `CREATE USER` statement includes the realm as part of the user name. This differs from creating accounts that use the `authentication_kerberos` Kerberos plugin. For such accounts, the `CREATE USER` statement does not include the realm as part of the user name. Instead, specify the realm as the authentication string in the `BY` clause. See Create a MySQL Account That Uses Kerberos Authentication.

The proxied account uses the `mysql_no_login` authentication plugin to prevent clients from using the account to log in directly to the MySQL server. Instead, it is expected that users who authenticate using LDAP use the `bredon@MYSQL.LOCAL` proxy account. (This assumes that the `mysql_no_login` plugin is installed. For instructions, see Section 6.4.1.9, “No-Login Pluggable Authentication”.) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

Use the MySQL Account to Connect to the MySQL Server

After a MySQL account that authenticates using GSSAPI/Kerberos has been set up, clients can use it to connect to the MySQL server. Kerberos authentication can take place either prior to or at the time of MySQL client program invocation:

- Prior to invoking the MySQL client program, the client user can obtain a TGT from the KDC independently of MySQL. For example, the client user can use `kinit` to authenticate to Kerberos by providing a Kerberos principal name and the principal password:

  ```
  $> kinit bredon@MYSQL.LOCAL
  Password for bredon@MYSQL.LOCAL: (enter password here)
  ```

  The resulting TGT is cached and becomes available for use by other Kerberos-aware applications, such as programs that use the client-side SASL LDAP authentication plugin. In this case, the MySQL client program authenticates to the MySQL server using the TGT, so invoke the client without specifying a user name or password:

  ```
  mysql --default-auth=authentication_ldap_sasl_client
  ```

  As just described, when the TGT is cached, user-name and password options are not needed in the client command. If the command includes them anyway, they are handled as follows:

  - If the command includes a user name, authentication fails if that name does not match the principal name in the TGT.

  - If the command includes a password, the client-side plugin ignores it. Because authentication is based on the TGT, it can succeed *even if the user-provided password is incorrect*. For this reason, the plugin produces a warning if a valid TGT is found that causes a password to be ignored.

- If the Kerberos cache contains no TGT, the client-side SASL LDAP authentication plugin itself can obtain the TGT from the KDC. Invoke the client with options for the name and password of the Kerberos principal associated with the MySQL account (enter the command on a single line, then enter the principal password when prompted):

```
mysql --default-auth=authentication_ldap_sasl_client
  --user=bredon@MYSQL.LOCAL
  --password
```

- If the Kerberos cache contains no TGT and the client command specifies no principal name as the user name, authentication fails.

If you are uncertain whether a TGT exists, you can use `klist` to check.

Authentication occurs as follows:

1. The client uses the TGT to authenticate using Kerberos.
2. The server finds the LDAP entry for the principal and uses it to authenticate the connection for the `bredon@MYSQL.LOCAL` MySQL proxy account.
3. The group mapping information in the proxy account authentication string (`'#krb_grp=proxied_krb_user'`) indicates that the authenticated proxied user should be `proxied_krb_user`.
4. `bredon@MYSQL.LOCAL` is treated as a proxy for `proxied_krb_user`, and the following query returns output as shown:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+------------------------------+--------------------+--------------------------+
| USER()                       | CURRENT_USER()     | @@proxy_user             |
+------------------------------+--------------------+--------------------------+
| bredon@MYSQL.LOCAL@localhost | proxied_krb_user@% | 'bredon@MYSQL.LOCAL'@'%' |
+------------------------------+--------------------+--------------------------+
```

   The `USER()` value indicates the user name used for the client command (`bredon@MYSQL.LOCAL`) and the host from which the client connected (`localhost`).

   The `CURRENT_USER()` value is the full name of the proxied user account, which consists of the `proxied_krb_user` user part and the `%` host part.

   The `@@proxy_user` value indicates the full name of the account used to make the connection to the MySQL server, which consists of the `bredon@MYSQL.LOCAL` user part and the `%` host part.

   This demonstrates that proxying occurs through the `bredon@MYSQL.LOCAL` proxy user account, and that `bredon@MYSQL.LOCAL` assumes the privileges granted to the `proxied_krb_user` proxied user account.

A TGT once obtained is cached on the client side and can be used until it expires without specifying the password again. However the TGT is obtained, the client-side plugin uses it to acquire service tickets and communicate with the server-side plugin.

Note

When the client-side authentication plugin itself obtains the TGT, the client user may not want the TGT to be reused. As described in Client Configuration Parameters for LDAP Authentication, the local `/etc/krb5.conf` file can be used to cause the client-side plugin to destroy the TGT when done with it.

The server-side plugin has no access to the TGT itself or the Kerberos password used to obtain it.

The LDAP authentication plugins have no control over the caching mechanism (storage in a local file, in memory, and so forth), but Kerberos utilities such as `kswitch` may be available for this purpose.

Client Configuration Parameters for LDAP Authentication

The `authentication_ldap_sasl_client` client-side SASL LDAP plugin reads the local `/etc/krb5.conf` file. If this file is missing or inaccessible, an error occurs. Assuming that the file is accessible, it can include an optional `[appdefaults]` section to provide information used by the plugin. Place the information within the `mysql` part of the section. For example:

```
[appdefaults]
  mysql = {
    ldap_server_host = "ldap_host.example.com"
    ldap_destroy_tgt = true
  }
```

The client-side plugin recognizes these parameters in the `mysql` section:

- The `ldap_server_host` value specifies the LDAP server host and can be useful when that host differs from the KDC server host specified in the `[realms]` section. By default, the plugin uses the KDC server host as the LDAP server host.
- The `ldap_destroy_tgt` value indicates whether the client-side plugin destroys the TGT after obtaining and using it. By default, `ldap_destroy_tgt` is `false`, but can be set to `true` to avoid TGT reuse. (This setting applies only to TGTs created by the client-side plugin, not TGTs created by other plugins or externally to MySQL.)

LDAP Search Referral

An LDAP server can be configured to delegate LDAP searches to another LDAP server, a functionality known as LDAP referral. Suppose that the server `a.example.com` holds a `"dc=example,dc=com"` root DN and wishes to delegate searches to another server `b.example.com`. To enable this, `a.example.com` would be configured with a named referral object having these attributes:

```
dn: dc=subtree,dc=example,dc=com
objectClass: referral
objectClass: extensibleObject
dc: subtree
ref: ldap://b.example.com/dc=subtree,dc=example,dc=com
```

An issue with enabling LDAP referral is that searches can fail with LDAP operation errors when the search base DN is the root DN, and referral objects are not set. A MySQL DBA might wish to avoid such referral errors for the LDAP authentication plugins, even though LDAP referral might be set globally in the `ldap.conf` configuration file. To configure on a plugin-specific basis whether the LDAP server should use LDAP referral when communicating with each plugin, set the `authentication_ldap_simple_referral` and `authentication_ldap_sasl_referral` system variables. Setting either variable to `ON` or `OFF` causes the corresponding LDAP authentication plugin to tell the LDAP server whether to use referral during MySQL authentication. Each variable has a plugin-specific effect and does not affect other applications that communicate with the LDAP server. Both variables are `OFF` by default.

6.4.1.8 Kerberos Pluggable Authentication

Note

Kerberos pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition supports an authentication method that enables users to authenticate to MySQL Server using Kerberos, provided that appropriate Kerberos tickets are available or can be obtained.

This authentication method is available in MySQL 8.0.26 and higher, for MySQL servers and clients on Linux. It is useful in Linux environments where applications have access to Microsoft Active Directory, which has Kerberos enabled by default. As of MySQL 8.0.27 (MySQL 8.0.32 for MIT Kerberos), the client-side plugin is supported on Windows as well. The server-side plugin is still supported only on Linux.

Kerberos pluggable authentication provides these capabilities:

- External authentication: Kerberos authentication enables MySQL Server to accept connections from users defined outside the MySQL grant tables who have obtained the proper Kerberos tickets.
- Security: Kerberos uses tickets together with symmetric-key cryptography, enabling authentication without sending passwords over the network. Kerberos authentication supports userless and passwordless scenarios.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable. For installation information, see Installing Kerberos Pluggable Authentication.

Table 6.24 Plugin and Library Names for Kerberos Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`authentication_kerberos`
Client-side plugin	`authentication_kerberos_client`
Library file	`authentication_kerberos.so`, `authentication_kerberos_client.so`

The server-side Kerberos authentication plugin is included only in MySQL Enterprise Edition. It is not included in MySQL community distributions. The client-side plugin is included in all distributions, including community distributions. This enables clients from any distribution to connect to a server that has the server-side plugin loaded.

The following sections provide installation and usage information specific to Kerberos pluggable authentication:

- Prerequisites for Kerberos Pluggable Authentication
- How Kerberos Authentication of MySQL Users Works
- Installing Kerberos Pluggable Authentication
- Using Kerberos Pluggable Authentication
- Kerberos Authentication Debugging

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Prerequisites for Kerberos Pluggable Authentication

To use Kerberos pluggable authentication for MySQL, these prerequisites must be satisfied:

- A Kerberos service must be available for the Kerberos authentication plugins to communicate with.
- Each Kerberos user (principal) to be authenticated by MySQL must be present in the database managed by the KDC server.
- A Kerberos client library must be available on systems where either the server-side or client-side Kerberos authentication plugin is used. In addition, GSSAPI is used as the interface for accessing Kerberos authentication, so a GSSAPI library must be available.

How Kerberos Authentication of MySQL Users Works

This section provides an overview of how MySQL and Kerberos work together to authenticate MySQL users. For examples showing how to set up MySQL accounts to use the Kerberos authentication plugins, see Using Kerberos Pluggable Authentication.

Familiarity is assumed here with Kerberos concepts and operation. The following list briefly defines several common Kerberos terms. You may also find the Glossary section of RFC 4120 helpful.

- Principal: A named entity, such as a user or server. In this discussion, certain principal-related terms occur frequently:
  - SPN: Service principal name; the name of a principal that represents a service.
  - UPN: User principal name; the name of a principal that represents a user.
- KDC: The key distribution center, comprising the AS and TGS:
  - AS: The authentication server; provides the initial ticket-granting ticket needed to obtain additional tickets.
  - TGS: The ticket-granting server; provides additional tickets to Kerberos clients that possess a valid TGT.
- TGT: The ticket-granting ticket; presented to the TGS to obtain service tickets for service access.
- ST: A service ticket; provides access to a service such as that offered by a MySQL server.

Authentication using Kerberos requires a KDC server, for example, as provided by Microsoft Active Directory.

Kerberos authentication in MySQL uses Generic Security Service Application Program Interface (GSSAPI), which is a security abstraction interface. Kerberos is an instance of a specific security protocol that can be used through that abstract interface. Using GSSAPI, applications authenticate to Kerberos to obtain service credentials, then use those credentials in turn to enable secure access to other services.

On Windows, the `authentication_kerberos_client` authentication plugin supports two modes, which the client user can set at runtime or specify in an option file:

- `SSPI` mode: Security Support Provider Interface (SSPI) implements GSSAPI (see Commands for Windows Clients in SSPI Mode). SSPI, while being compatible with GSSAPI at the wire level, only supports the Windows single sign-on scenario and specifically refers to the logged-on user. SSPI is the default mode on most Windows clients.
- `GSSAPI` mode: Supports GSSAPI through the MIT Kerberos library on Windows (see Commands for Windows Clients in GSSAPI Mode).

With the Kerberos authentication plugins, applications and MySQL servers are able to use the Kerberos authentication protocol to mutually authenticate users and MySQL services. This way both the user and the server are able to verify each other's identity. No passwords are sent over the network and Kerberos protocol messages are protected against eavesdropping and replay attacks.

Kerberos authentication follows these steps, where the server-side and client-side parts are performed using the `authentication_kerberos` and `authentication_kerberos_client` authentication plugins, respectively:

1. The MySQL server sends to the client application its service principal name. This SPN must be registered in the Kerberos system, and is configured on the server side using the `authentication_kerberos_service_principal` system variable.
2. Using GSSAPI, the client application creates a Kerberos client-side authentication session and exchanges Kerberos messages with the Kerberos KDC:

- The client obtains a ticket-granting ticket from the authentication server.
- Using the TGT, the client obtains a service ticket for MySQL from the ticket-granting service.

This step can be skipped or partially skipped if the TGT, ST, or both are already cached locally. The client optionally may use a client keytab file to obtain a TGT and ST without supplying a password.

3. Using GSSAPI, the client application presents the MySQL ST to the MySQL server.
4. Using GSSAPI, the MySQL server creates a Kerberos server-side authentication session. The server validates the user identity and the validity of the user request. It authenticates the ST using the service key configured in its service keytab file to determine whether authentication succeeds or fails, and returns the authentication result to the client.

Applications are able to authenticate using a provided user name and password, or using a locally cached TGT or ST (for example, created using `kinit` or similar). This design therefore covers use cases ranging from completely userless and passwordless connections, where Kerberos service tickets are obtained from a locally stored Kerberos cache, to connections where both user name and password are provided and used to obtain a valid Kerberos service ticket from a KDC, to send to the MySQL server.

As indicated in the preceding description, MySQL Kerberos authentication uses two kinds of keytab files:

- On the client host, a client keytab file may be used to obtain a TGT and ST without supplying a password. See Client Configuration Parameters for Kerberos Authentication.
- On the MySQL server host, a server-side service keytab file is used to verify service tickets received by the MySQL server from clients. The keytab file name is configured using the `authentication_kerberos_service_key_tab` system variable.

For information about keytab files, see https://web.mit.edu/kerberos/krb5-latest/doc/basic/keytab_def.html.

Installing Kerberos Pluggable Authentication

This section describes how to install the server-side Kerberos authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

Note

The server-side plugin is supported only on Linux systems. On Windows systems, only the client-side plugin is supported (as of MySQL 8.0.27), which can be used on a Windows system to connect to a Linux server that uses Kerberos authentication.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The server-side plugin library file base name is `authentication_kerberos`. The file name suffix for Unix and Unix-like systems is `.so`.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. Also, specify values for any plugin-provided system variables you wish to configure. The plugin exposes these system variables, enabling its operation to be configured:

- `authentication_kerberos_service_principal`: The MySQL service principal name (SPN). This name is sent to clients that attempt to authenticate using Kerberos. The SPN must be present in the database managed by the KDC server. The default is `mysql/host_name@realm_name`.

- `authentication_kerberos_service_key_tab`: The keytab file for authenticating tickets received from clients. This file must exist and contain a valid key for the SPN or authentication of clients will fail. The default is `mysql.keytab` in the data directory.

For details about all Kerberos authentication system variables, see Section 6.4.1.13, “Pluggable Authentication System Variables”.

To load the plugin and configure it, put lines such as these in your `my.cnf` file, using values for the system variables that are appropriate for your installation:

```
[mysqld]
plugin-load-add=authentication_kerberos.so
authentication_kerberos_service_principal=mysql/krbauth.example.com@MYSQL.LOCAL
authentication_kerberos_service_key_tab=/var/mysql/data/mysql.keytab
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement:

```
INSTALL PLUGIN authentication_kerberos
  SONAME 'authentication_kerberos.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

When you install the plugin at runtime without configuring its system variables in the `my.cnf` file, the system variable `authentication_kerberos_service_key_tab` is set to the default value of `mysql.keytab` in the data directory. The value of this system variable cannot be changed at runtime, so if you need to specify a different file, you need to add the setting to your `my.cnf` file then restart the MySQL server. For example:

```
[mysqld]
authentication_kerberos_service_key_tab=/var/mysql/data/mysql.keytab
```

If the keytab file is not in the correct place or does not contain a valid SPN key, the MySQL server does not validate this, but clients return authentication errors until you fix the issue.

The `authentication_kerberos_service_principal` system variable can be set and persisted at runtime without restarting the server, by using a `SET PERSIST` statement:

```
SET PERSIST authentication_kerberos_service_principal='mysql/krbauth.example.com@MYSQL.LOCAL';
```

`SET PERSIST` sets a value for the running MySQL instance. It also saves the value, causing it to carry over to subsequent server restarts. To change a value for the running MySQL instance without having it carry over to subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME = 'authentication_kerberos';
+-------------------------+---------------+
| PLUGIN_NAME             | PLUGIN_STATUS |
+-------------------------+---------------+
| authentication_kerberos | ACTIVE        |
+-------------------------+---------------+
```

If a plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the Kerberos plugin, see Using Kerberos Pluggable Authentication.

Using Kerberos Pluggable Authentication

This section describes how to enable MySQL accounts to connect to the MySQL server using Kerberos pluggable authentication. It is assumed that the server is running with the server-side plugin enabled, as described in Installing Kerberos Pluggable Authentication, and that the client-side plugin is available on the client host.

- Verify Kerberos Availability
- Create a MySQL Account That Uses Kerberos Authentication
- Use the MySQL Account to Connect to the MySQL Server
- Client Configuration Parameters for Kerberos Authentication

Verify Kerberos Availability

The following example shows how to test availability of Kerberos in Active Directory. The example makes these assumptions:

- Active Directory is running on the host named `krbauth.example.com` with IP address `198.51.100.11`.
- MySQL-related Kerberos authentication uses the `MYSQL.LOCAL` domain, and also uses `MYSQL.LOCAL` as the realm name.
- A principal named `karl@MYSQL.LOCAL` is registered with the KDC. (In later discussion, this principal name is associated with the MySQL account that authenticates to the MySQL server using Kerberos.)

With those assumptions satisfied, follow this procedure:

1. Verify that the Kerberos library is installed and configured correctly in the operating system. For example, to configure a `MYSQL.LOCAL` domain and realm for use during MySQL authentication, the `/etc/krb5.conf` Kerberos configuration file should contain something like this:

   ```
   [realms]
     MYSQL.LOCAL = {
       kdc = krbauth.example.com
       admin_server = krbauth.example.com
       default_domain = MYSQL.LOCAL
     }
   ```

2. You may need to add an entry to `/etc/hosts` for the server host:

   ```
   198.51.100.11 krbauth krbauth.example.com
   ```

3. Check whether Kerberos authentication works correctly:

   a. Use `kinit` to authenticate to Kerberos:

      ```
      $> kinit karl@MYSQL.LOCAL
      Password for karl@MYSQL.LOCAL: (enter password here)
      ```

      The command authenticates for the Kerberos principal named `karl@MYSQL.LOCAL`. Enter the principal's password when the command prompts for it. The KDC returns a TGT that is cached on the client side for use by other Kerberos-aware applications.

   b. Use `klist` to check whether the TGT was obtained correctly. The output should be similar to this:

      ```
      $> klist
      Ticket cache: FILE:/tmp/krb5cc_244306
      Default principal: karl@MYSQL.LOCAL
      ```

```
Valid starting       Expires              Service principal
03/23/2021 08:18:33  03/23/2021 18:18:33  krbtgt/MYSQL.LOCAL@MYSQL.LOCAL
```

Create a MySQL Account That Uses Kerberos Authentication

MySQL authentication using the `authentication_kerberos` authentication plugin is based on a Kerberos user principal name (UPN). The instructions here assume that a MySQL user named `karl` authenticates to MySQL using Kerberos, that the Kerberos realm is named `MYSQL.LOCAL`, and that the user principal name is `karl@MYSQL.LOCAL`. This UPN must be registered in several places:

- The Kerberos administrator should register the user name as a Kerberos principal. This name includes a realm name. Clients use the principal name and password to authenticate with Kerberos and obtain a ticket-granting ticket (TGT).
- The MySQL DBA should create an account that corresponds to the Kerberos principal name and that authenticates using the Kerberos plugin.

Assume that the Kerberos user principal name has been registered by the appropriate service administrator, and that, as previously described in Installing Kerberos Pluggable Authentication, the MySQL server has been started with appropriate configuration settings for the server-side Kerberos plugin. To create a MySQL account that corresponds to a Kerberos UPN of *`user@realm_name`*, the MySQL DBA uses a statement like this:

```
CREATE USER user
  IDENTIFIED WITH authentication_kerberos
  BY 'realm_name';
```

The account named by *`user`* can include or omit the host name part. If the host name is omitted, it defaults to `%` as usual. The *`realm_name`* is stored as the `authentication_string` value for the account in the `mysql.user` system table.

To create a MySQL account that corresponds to the UPN `karl@MYSQL.LOCAL`, use this statement:

```
CREATE USER 'karl'
  IDENTIFIED WITH authentication_kerberos
  BY 'MYSQL.LOCAL';
```

If MySQL must construct the UPN for this account, for example, to obtain or validate tickets (TGTs or STs), it does so by combining the account name (ignoring any host name part) and the realm name. For example, the full account name resulting from the preceding `CREATE USER` statement is `'karl'@'%'`. MySQL constructs the UPN from the user name part `karl` (ignoring the host name part) and the realm name `MYSQL.LOCAL` to produce `karl@MYSQL.LOCAL`.

Note

Observe that when creating an account that authenticates using `authentication_kerberos`, the `CREATE USER` statement does not include the UPN realm as part of the user name. Instead, specify the realm (`MYSQL.LOCAL` in this case) as the authentication string in the `BY` clause. This differs from creating accounts that use the `authentication_ldap_sasl` SASL LDAP authentication plugin with the GSSAPI/Kerberos authentication method. For such accounts, the `CREATE USER` statement does include the UPN realm as part of the user name. See Create a MySQL Account That Uses GSSAPI/Kerberos for LDAP Authentication.

With the account set up, clients can use it to connect to the MySQL server. The procedure depends on whether the client host runs Linux or Windows, as indicated in the following discussion.

Use of `authentication_kerberos` is subject to the restriction that UPNs with the same user part but a different realm part are not supported. For example, you cannot create MySQL accounts that correspond to both these UPNs:

```
kate@MYSQL.LOCAL
```

```
kate@EXAMPLE.COM
```

Both UPNs have a user part of `kate` but differ in the realm part (`MYSQL.LOCAL` versus `EXAMPLE.COM`). This is disallowed.

Use the MySQL Account to Connect to the MySQL Server

After a MySQL account that authenticates using Kerberos has been set up, clients can use it to connect to the MySQL server as follows:

1. Authenticate to Kerberos with the user principal name (UPN) and its password to obtain a ticket-granting ticket (TGT).
2. Use the TGT to obtain a service ticket (ST) for MySQL.
3. Authenticate to the MySQL server by presenting the MySQL ST.

The first step (authenticating to Kerberos) can be performed various ways:

- Prior to connecting to MySQL:
  - On Linux or on Windows in `GSSAPI` mode, invoke `kinit` to obtain the TGT and save it in the Kerberos credentials cache.
  - On Windows in `SSPI` mode, authentication may already have been done at login time, which saves the TGT for the logged-in user in the Windows in-memory cache. `kinit` is not used and there is no Kerberos cache.
- When connecting to MySQL, the client program itself can obtain the TGT, if it can determine the required Kerberos UPN and password:
  - That information can come from sources such as command options or the operating system.
  - On Linux, clients also can use a keytab file or the `/etc/krb5.conf` configuration file. Windows clients in `GSSAPI` mode use a configuration file. Windows clients in `SSPI` mode use neither.

Details of the client commands for connecting to the MySQL server differ for Linux and Windows, so each host type is discussed separately, but these command properties apply regardless of host type:

- Each command shown includes the following options, but each one may be omitted under certain conditions:
  - The `--default-auth` option specifies the name of the client-side authentication plugin (`authentication_kerberos_client`). This option may be omitted when the `--user` option is specified because in that case MySQL can determine the plugin from the user account information sent by MySQL server.
  - The `--plugin-dir` option indicates to the client program the location of the `authentication_kerberos_client` plugin. This option may be omitted if the plugin is installed in the default (compiled-in) location.
- Commands should also include any other options such as `--host` or `--port` that are required to specify which MySQL server to connect to.
- Enter each command on a single line. If the command includes a `--password` option to solicit a password, enter the password of the Kerberos UPN associated with the MySQL user when prompted.

Connection Commands for Linux Clients

On Linux, the appropriate client command for connecting to the MySQL server varies depending on whether the command authenticates using a TGT from the Kerberos cache, or based on command options for the MySQL user name and the UPN password:

- Prior to invoking the MySQL client program, the client user can obtain a TGT from the KDC independently of MySQL. For example, the client user can use `kinit` to authenticate to Kerberos by providing a Kerberos user principal name and the principal password:

  ```
  $> kinit karl@MYSQL.LOCAL
  Password for karl@MYSQL.LOCAL: (enter password here)
  ```

  The resulting TGT for the UPN is cached and becomes available for use by other Kerberos-aware applications, such as programs that use the client-side Kerberos authentication plugin. In this case, invoke the client without specifying a user-name or password option:

  ```
  mysql
    --default-auth=authentication_kerberos_client
    --plugin-dir=path/to/plugin/directory
  ```

  The client-side plugin finds the TGT in the cache, uses it to obtain a MySQL ST, and uses the ST to authenticate to the MySQL server.

  As just described, when the TGT for the UPN is cached, user-name and password options are not needed in the client command. If the command includes them anyway, they are handled as follows:

  - This command includes a user-name option:

    ```
    mysql
      --default-auth=authentication_kerberos_client
      --plugin-dir=path/to/plugin/directory
      --user=karl
    ```

    In this case, authentication fails if the user name specified by the option does not match the user name part of the UPN in the TGT.

  - This command includes a password option, which you enter when prompted:

    ```
    mysql
      --default-auth=authentication_kerberos_client
      --plugin-dir=path/to/plugin/directory
      --password
    ```

    In this case, the client-side plugin ignores the password. Because authentication is based on the TGT, it can succeed *even if the user-provided password is incorrect*. For this reason, the plugin produces a warning if a valid TGT is found that causes a password to be ignored.

- If the Kerberos cache contains no TGT, the client-side Kerberos authentication plugin itself can obtain the TGT from the KDC. Invoke the client with options for the MySQL user name and the password, then enter the UPN password when prompted:

  ```
  mysql --default-auth=authentication_kerberos_client
    --plugin-dir=path/to/plugin/directory
    --user=karl
    --password
  ```

  The client-side Kerberos authentication plugin combines the user name (`karl`) and the realm specified in the user account (`MYSQL.LOCAL`) to construct the UPN (`karl@MYSQL.LOCAL`). The client-side plugin uses the UPN and password to obtain a TGT, uses the TGT to obtain a MySQL ST, and uses the ST to authenticate to the MySQL server.

  Or, suppose that the Kerberos cache contains no TGT and the command specifies a password option but no user-name option:

  ```
  mysql --default-auth=authentication_kerberos_client
    --plugin-dir=path/to/plugin/directory
    --password
  ```

  The client-side Kerberos authentication plugin uses the operating system login name as the MySQL user name. It combines that user name and the realm in the user's MySQL account to construct the

UPN. The client-side plugin uses the UPN and the password to obtain a TGT, uses the TGT to obtain a MySQL ST, and uses the ST to authenticate to the MySQL server.

If you are uncertain whether a TGT exists, you can use `klist` to check.

Note

When the client-side Kerberos authentication plugin itself obtains the TGT, the client user may not want the TGT to be reused. As described in Client Configuration Parameters for Kerberos Authentication, the local `/etc/krb5.conf` file can be used to cause the client-side plugin to destroy the TGT when done with it.

Connection Commands for Windows Clients in SSPI Mode

On Windows, using the default client-side plugin option (SSPI), the appropriate client command for connecting to the MySQL server varies depending on whether the command authenticates based on command options for the MySQL user name and the UPN password, or instead uses a TGT from the Windows in-memory cache. For details about GSSAPI mode on Windows, see Commands for Windows Clients in GSSAPI Mode.

A command can explicitly specify options for the MySQL user name and the UPN password, or the command can omit those options:

- This command includes options for the MySQL user name and UPN password:

```
mysql --default-auth=authentication_kerberos_client
  --plugin-dir=path/to/plugin/directory
  --user=karl
  --password
```

  The client-side Kerberos authentication plugin combines the user name (`karl`) and the realm specified in the user account (`MYSQL.LOCAL`) to construct the UPN (`karl@MYSQL.LOCAL`). The client-side plugin uses the UPN and password to obtain a TGT, uses the TGT to obtain a MySQL ST, and uses the ST to authenticate to the MySQL server.

  Any information in the Windows in-memory cache is ignored; the user-name and password option values take precedence.

- This command includes an option for the UPN password but not for the MySQL user name:

```
mysql
  --default-auth=authentication_kerberos_client
  --plugin-dir=path/to/plugin/directory
  --password
```

  The client-side Kerberos authentication plugin uses the logged-in user name as the MySQL user name and combines that user name and the realm in the user's MySQL account to construct the UPN. The client-side plugin uses the UPN and the password to obtain a TGT, uses the TGT to obtain a MySQL ST, and uses the ST to authenticate to the MySQL server.

- This command includes no options for the MySQL user name or UPN password:

```
mysql
  --default-auth=authentication_kerberos_client
  --plugin-dir=path/to/plugin/directory
```

  The client-side plugin obtains the TGT from the Windows in-memory cache, uses the TGT to obtain a MySQL ST, and uses the ST to authenticate to the MySQL server.

  This approach requires the client host to be part of the Windows Server Active Directory (AD) domain. If that is not the case, help the MySQL client discover the IP address for the AD domain by manually entering the AD server and realm as the DNS server and prefix:

1. Start `console.exe` and select **Network and Sharing Center**.
2. From the sidebar of the Network and Sharing Center window, select **Change adapter settings**.
3. In the Network Connections window, right-click the network or VPN connection to configure and select **Properties**.
4. From the **Network** tab, locate and click **Internet Protocol Version 4 (TCP/IPv4)**, and then click **Properties**.
5. Click **Advanced** in the Internet Protocol Version 4 (TCP/IPv4) Properties dialog. The Advanced TCP/IP Settings dialog opens.
6. From the **DNS** tab, add the Active Directory server and realm as a DNS server and prefix.

- This command includes an option for the MySQL user name but not for the UPN password:

```
mysql
  --default-auth=authentication_kerberos_client
  --plugin-dir=path/to/plugin/directory
  --user=karl
```

  The client-side Kerberos authentication plugin compares the name specified by the user-name option against the logged-in user name. If the names are the same, the plugin uses the logged-in user TGT for authentication. If the names differ, authentication fails.

Connection Commands for Windows Clients in GSSAPI Mode

On Windows, the client user must specify `GSSAPI` mode explicitly using the `plugin_authentication_kerberos_client_mode` plugin option to enable support through the MIT Kerberos library. The default mode is `SSPI` (see Commands for Windows Clients in SSPI Mode).

It is possible to specify `GSSAPI` mode:

- Prior to invoking the MySQL client program in an option file. The plugin variable name is valid using either underscores or dashes:

```
[mysql]
plugin_authentication_kerberos_client_mode=GSSAPI
```

  Or:

```
[mysql]
plugin-authentication-kerberos-client-mode=GSSAPI
```

- At runtime from the command line using the `mysql` or `mysqldump` client programs. For example, the following commands (with underscores or dashes) causes `mysql` to connect to the server through the MIT Kerberos library on Windows.

```
mysql [connection-options] --plugin_authentication_kerberos_client_mode=GSSAPI
```

  Or:

```
mysql [connection-options] --plugin-authentication-kerberos-client-mode=GSSAPI
```

- Client users can select `GSSAPI` mode from MySQL Workbench and some MySQL connectors. On client hosts running Windows, you can override the default location of:
  - The Kerberos configuration file by setting the `KRB5_CONFIG` environment variable.
  - The default credential cache name with the `KRB5CCNAME` environment variable (for example, `KRB5CCNAME=DIR:/mydir/`).

  For specific client-side plugin information, see the documentation at https://dev.mysql.com/doc/.

The appropriate client command for connecting to the MySQL server varies depending on whether the command authenticates using a TGT from the MIT Kerberos cache, or based on command options for the MySQL user name and the UPN password. GSSAPI support through the MIT library on Windows is similar to GSSAPI on Linux (see Commands for Linux Clients), with the following exceptions:

- Tickets are always retrieved from or placed into the MIT Kerberos cache on hosts running Windows.
- `kinit` runs with Functional Accounts on Windows that have narrow permissions and specific roles. The client user does not know the `kinit` password. For an overview, see https://docs.oracle.com/en/java/javase/11/tools/kinit.html.
- If the client user supplies a password, the MIT Kerberos library on Windows decides whether to use it or rely on the existing ticket.
- The `destroy_tickets` parameter, described in Client Configuration Parameters for Kerberos Authentication, is not supported because the MIT Kerberos library on Windows does not support the required API member (`get_profile_boolean`) to read its value from configuration file.

Client Configuration Parameters for Kerberos Authentication

This section applies only for client hosts running Linux, not client hosts running Windows.

Note

A client host running Windows with the `authentication_kerberos_client` client-side Kerberos plugin set to `GSSAPI` mode does support client configuration parameters, in general, but the MIT Kerberos library on Windows does not support the `destroy_tickets` parameter described in this section.

If no valid ticket-granting ticket (TGT) exists at the time of MySQL client application invocation, the application itself may obtain and cache the TGT. If during the Kerberos authentication process the client application causes a TGT to be cached, any such TGT that was added can be destroyed after it is no longer needed, by setting the appropriate configuration parameter.

The `authentication_kerberos_client` client-side Kerberos plugin reads the local `/etc/krb5.conf` file. If this file is missing or inaccessible, an error occurs. Assuming that the file is accessible, it can include an optional `[appdefaults]` section to provide information used by the plugin. Place the information within the `mysql` part of the section. For example:

```
[appdefaults]
  mysql = {
    destroy_tickets = true
  }
```

The client-side plugin recognizes these parameters in the `mysql` section:

- The `destroy_tickets` value indicates whether the client-side plugin destroys the TGT after obtaining and using it. By default, `destroy_tickets` is `false`, but can be set to `true` to avoid TGT reuse. (This setting applies only to TGTs created by the client-side plugin, not TGTs created by other plugins or externally to MySQL.)

On the client host, a client keytab file may be used to obtain a TGT and TS without supplying a password. For information about keytab files, see https://web.mit.edu/kerberos/krb5-latest/doc/basic/keytab_def.html.

Kerberos Authentication Debugging

The `AUTHENTICATION_KERBEROS_CLIENT_LOG` environment variable enables or disables debug output for Kerberos authentication.

Note

Despite `CLIENT` in the name `AUTHENTICATION_KERBEROS_CLIENT_LOG`, the same environment variable applies to the server-side plugin as well as the client-side plugin.

On the server side, the permitted values are 0 (off) and 1 (on). Log messages are written to the server error log, subject to the server error-logging verbosity level. For example, if you are using priority-based log filtering, the `log_error_verbosity` system variable controls verbosity, as described in Section 5.4.2.5, “Priority-Based Error Log Filtering (log_filter_internal)”.

On the client side, the permitted values are from 1 to 5 and are written to the standard error output. The following table shows the meaning of each log-level value.

Log Level	Meaning
1 or not set	No logging
2	Error messages
3	Error and warning messages
4	Error, warning, and information messages
5	Error, warning, information, and debug messages

6.4.1.9 No-Login Pluggable Authentication

The `mysql_no_login` server-side authentication plugin prevents all client connections to any account that uses it. Use cases for this plugin include:

- Accounts that must be able to execute stored programs and views with elevated privileges without exposing those privileges to ordinary users.
- Proxied accounts that should never permit direct login but are intended to be accessed only through proxy accounts.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.25 Plugin and Library Names for No-Login Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`mysql_no_login`
Client-side plugin	None
Library file	`mysql_no_login.so`

The following sections provide installation and usage information specific to no-login pluggable authentication:

- Installing No-Login Pluggable Authentication
- Uninstalling No-Login Pluggable Authentication
- Using No-Login Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”. For proxy user information, see Section 6.2.19, “Proxy Users”.

Installing No-Login Pluggable Authentication

This section describes how to install the no-login authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `mysql_no_login`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=mysql_no_login.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN mysql_no_login SONAME 'mysql_no_login.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, "Obtaining Server Plugin Information"). For example:

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

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the no-login plugin, see Using No-Login Pluggable Authentication.

Uninstalling No-Login Pluggable Authentication

The method used to uninstall the no-login authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

  ```
  UNINSTALL PLUGIN mysql_no_login;
  ```

Using No-Login Pluggable Authentication

This section describes how to use the no-login authentication plugin to prevent accounts from being used for connecting from MySQL client programs to the server. It is assumed that the server is running with the no-login plugin enabled, as described in Installing No-Login Pluggable Authentication.

To refer to the no-login authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` statement, use the name `mysql_no_login`.

An account that authenticates using `mysql_no_login` may be used as the `DEFINER` for stored program and view objects. If such an object definition also includes `SQL SECURITY DEFINER`, it

executes with that account's privileges. DBAs can use this behavior to provide access to confidential or sensitive data that is exposed only through well-controlled interfaces.

The following example illustrates these principles. It defines an account that does not permit client connections, and associates with it a view that exposes only certain columns of the `mysql.user` system table:

```
CREATE DATABASE nologindb;
CREATE USER 'nologin'@'localhost'
  IDENTIFIED WITH mysql_no_login;
GRANT ALL ON nologindb.*
  TO 'nologin'@'localhost';
GRANT SELECT ON mysql.user
  TO 'nologin'@'localhost';
CREATE DEFINER = 'nologin'@'localhost'
  SQL SECURITY DEFINER
  VIEW nologindb.myview
  AS SELECT User, Host FROM mysql.user;
```

To provide protected access to the view to an ordinary user, do this:

```
GRANT SELECT ON nologindb.myview
  TO 'ordinaryuser'@'localhost';
```

Now the ordinary user can use the view to access the limited information it presents:

```
SELECT * FROM nologindb.myview;
```

Attempts by the user to access columns other than those exposed by the view result in an error, as do attempts to select from the view by users not granted access to it.

Note

Because the `nologin` account cannot be used directly, the operations required to set up objects that it uses must be performed by `root` or similar account that has the privileges required to create the objects and set `DEFINER` values.

The `mysql_no_login` plugin is also useful in proxying scenarios. (For a discussion of concepts involved in proxying, see Section 6.2.19, “Proxy Users”.) An account that authenticates using `mysql_no_login` may be used as a proxied user for proxy accounts:

```
-- create proxied account
CREATE USER 'proxied_user'@'localhost'
  IDENTIFIED WITH mysql_no_login;
-- grant privileges to proxied account
GRANT ...
  ON ...
  TO 'proxied_user'@'localhost';
-- permit proxy_user to be a proxy account for proxied account
GRANT PROXY
  ON 'proxied_user'@'localhost'
  TO 'proxy_user'@'localhost';
```

This enables clients to access MySQL through the proxy account (`proxy_user`) but not to bypass the proxy mechanism by connecting directly as the proxied user (`proxied_user`). A client who connects using the `proxy_user` account has the privileges of the `proxied_user` account, but `proxied_user` itself cannot be used to connect.

For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

6.4.1.10 Socket Peer-Credential Pluggable Authentication

The server-side `auth_socket` authentication plugin authenticates clients that connect from the local host through the Unix socket file. The plugin uses the `SO_PEERCRED` socket option to obtain information about the user running the client program. Thus, the plugin can be used only on systems that support the `SO_PEERCRED` option, such as Linux.

The source code for this plugin can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

The following table shows the plugin and library file names. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.26 Plugin and Library Names for Socket Peer-Credential Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`auth_socket`
Client-side plugin	None, see discussion
Library file	`auth_socket.so`

The following sections provide installation and usage information specific to socket pluggable authentication:

- Installing Socket Pluggable Authentication
- Uninstalling Socket Pluggable Authentication
- Using Socket Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Installing Socket Pluggable Authentication

This section describes how to install the socket authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file:

```
[mysqld]
plugin-load-add=auth_socket.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement:

```
INSTALL PLUGIN auth_socket SONAME 'auth_socket.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

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

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the socket plugin, see Using Socket Pluggable Authentication.

Uninstalling Socket Pluggable Authentication

The method used to uninstall the socket authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN auth_socket;
```

Using Socket Pluggable Authentication

The socket plugin checks whether the socket user name (the operating system user name) matches the MySQL user name specified by the client program to the server. If the names do not match, the plugin checks whether the socket user name matches the name specified in the `authentication_string` column of the `mysql.user` system table row. If a match is found, the plugin permits the connection. The `authentication_string` value can be specified using an `IDENTIFIED ...AS` clause with `CREATE USER` or `ALTER USER`.

Suppose that a MySQL account is created for an operating system 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.

To permit both the `valerie` and `stephanie` operating system users to access MySQL through socket file connections that use the account, this can be done two ways:

- Name both users at account-creation time, one following `CREATE USER`, and the other in the authentication string:

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

- If you have already used `CREATE USER` to create the account for a single user, use `ALTER USER` to add the second user:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
ALTER USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket AS 'stephanie';
```

To access the account, both `valerie` and `stephanie` specify `--user=valerie` at connect time.

6.4.1.11 FIDO Pluggable Authentication

Note

FIDO pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition supports an authentication method that enables users to authenticate to MySQL Server using FIDO authentication. This authentication method is available in MySQL 8.0.27 and higher.

FIDO stands for Fast Identity Online, which provides standards for authentication that does not require use of passwords.

FIDO pluggable authentication provides these capabilities:

- FIDO enables authentication to MySQL Server using devices such as smart cards, security keys, and biometric readers.
- Because authentication can occur other than by providing a password, FIDO enables passwordless authentication.
- On the other hand, device authentication is often used in conjunction with password authentication, so FIDO authentication can be used to good effect for MySQL accounts that use multifactor authentication; see Section 6.2.18, “Multifactor Authentication”.

The following table shows the plugin and library file names. The file name suffix might differ on your system. Common suffixes are `.so` for Unix and Unix-like systems, and `.dll` for Windows. The file must be located in the directory named by the `plugin_dir` system variable. For installation information, see Installing FIDO Pluggable Authentication.

Table 6.27 Plugin and Library Names for FIDO Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`authentication_fido`
Client-side plugin	`authentication_fido_client`
Library file	`authentication_fido.so`, `authentication_fido_client.so`

Note

A `libfido2` library must be available on systems where either the server-side or client-side FIDO authentication plugin is used. If a host machine has more than one FIDO device, the `libfido2` library decides which device to use for registration and authentication. The `libfido2` library does not provide a facility for device selection.

The server-side FIDO authentication plugin is included only in MySQL Enterprise Edition. It is not included in MySQL community distributions. The client-side plugin is included in all distributions, including community distributions, which enables clients from any distribution to connect to a server that has the server-side plugin loaded.

The following sections provide installation and usage information specific to FIDO pluggable authentication:

- Installing FIDO Pluggable Authentication
- Using FIDO Authentication
- FIDO Passwordless Authentication
- FIDO Device Unregistration
- How FIDO Authentication of MySQL Users Works

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Installing FIDO Pluggable Authentication

This section describes how to install the server-side FIDO authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The server-side plugin library file base name is `authentication_fido`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts.

To load the plugin, put a line such as this in your `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=authentication_fido.so
```

After modifying `my.cnf`, restart the server to cause the new setting to take effect.

Alternatively, to load the plugin at runtime, use this statement, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN authentication_fido
  SONAME 'authentication_fido.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME = 'authentication_fido';
+--------------------+---------------+
| PLUGIN_NAME        | PLUGIN_STATUS |
+--------------------+---------------+
| authentication_fido | ACTIVE        |
+--------------------+---------------+
```

If a plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the FIDO authentication plugin, see Using FIDO Authentication.

Using FIDO Authentication

FIDO authentication typically is used in the context of multifactor authentication (see Section 6.2.18, “Multifactor Authentication”). This section shows how to incorporate FIDO device-based authentication into a multifactor account, using the `authentication_fido` plugin.

It is assumed in the following discussion that the server is running with the server-side FIDO authentication plugin enabled, as described in Installing FIDO Pluggable Authentication, and that the client-side FIDO plugin is available in the plugin directory on the client host.

Note

On Windows, FIDO authentication only functions if the process runs as a user with administrator privileges.

It is also assumed that FIDO authentication is used in conjunction with non-FIDO authentication (which implies a 2FA or 3FA account). FIDO can also be used by itself to create 1FA accounts that authenticate in a passwordless manner. In this case, the setup process differs somewhat. For instructions, see FIDO Passwordless Authentication.

An account that is configured to use the `authentication_fido` plugin is associated with a FIDO device. Because of this, a one-time device registration step is required before FIDO authentication can occur. The device registration process has these characteristics:

- Any FIDO device associated with an account must be registered before the account can be used.

- Registration requires that a FIDO device be available on the client host, or registration fails.
- The user is expected to perform the appropriate FIDO device action when prompted during registration (for example, touching the device or performing a biometric scan).
- To perform device registration, the client user must invoke the `mysql` client program or MySQL Shell and specify the `--fido-register-factor` option to specify the factor or factors for which a device is being registered. For example, if the account is set to use FIDO as the second authentication factor, the user invokes `mysql` with the `--fido-register-factor=2` option.
- If the user account is configured with the `authentication_fido` plugin set as the second or third factor, authentication for all preceding factors must succeed before the registration step can proceed.
- The server knows from the information in the user account whether the FIDO device requires registration or has already been registered. When the client program connects, the server places the client session in sandbox mode if the device must be registered, so that registration must occur before anything else can be done. Sandbox mode used for FIDO device registration is similar to that used for handling of expired passwords. See Section 6.2.16, “Server Handling of Expired Passwords”.
- In sandbox mode, no statements other than `ALTER USER` are permitted. Registration is performed using forms of this statement. When invoked with the `--fido-register-factor` option, the `mysql` client generates the `ALTER USER` statements required to perform registration. After registration has been accomplished, the server switches the session out of sandbox mode, and the client can proceed normally. For information about the generated `ALTER USER` statements, refer to the `--fido-register-factor` description.
- When device registration has been performed for the account, the server updates the `mysql.user` system table row for that account to update the device registration status and to store the public key and credential ID.
- The registration step can be performed only by the user named by the account. If one user attempts to perform registration for another user, an error occurs.
- The user should use the same FIDO device during registration and authentication. If, after registering a FIDO device on the client host, the device is reset or a different device is inserted, authentication fails. In this case, the device associated with the account must be unregistered and registration must be done again.

Suppose that you want an account to authenticate first using the `caching_sha2_password` plugin, then using the `authentication_fido` plugin. Create a multifactor account using a statement like this:

```
CREATE USER 'u2'@'localhost'
  IDENTIFIED WITH caching_sha2_password
    BY 'sha2_password'
  AND IDENTIFIED WITH authentication_fido;
```

To connect, supply the factor 1 password to satisfy authentication for that factor, and to initiate registration of the FIDO device, set the `--fido-register-factor` to factor 2.

```
$> mysql --user=u2 --password1 --fido-register-factor=2
Enter password: (enter factor 1 password)
```

Once the factor 1 password is accepted, the client session enters sandbox mode so that device registration can be performed for factor 2. During registration, you are prompted to perform the appropriate FIDO device action, such as touching the device or performing a biometric scan.

When the registration process is complete, the connection to the server is permitted.

Note

The connection to the server is permitted following registration regardless of additional authentication factors in the account's authentication chain.

> For example, if the account in preceding example was defined with a third authentication factor (using non-FIDO authentication), the connection would be permitted after a successful registration without authenticating the third factor. However, subsequent connections would require authenticating all three factors.

FIDO Passwordless Authentication

This section describes how FIDO can be used by itself to create 1FA accounts that authenticate in a passwordless manner. In this context, “passwordless” means that authentication occurs but uses a method other than a password, such as security key or biometric scan. It does not refer to an account that uses a password-based authentication plugin for which the password is empty. That kind of “passwordless” is completely insecure and is not recommended.

The following prerequisites apply when using the `authentication_fido` plugin to achieve passwordless authentication:

- The user that creates a passwordless-authentication account requires the `PASSWORDLESS_USER_ADMIN` privilege in addition to the `CREATE USER` privilege.
- The first element of the `authentication_policy` value must be an asterisk (`*`) and not a plugin name. For example, the default `authentication_policy` value supports enabling passwordless authentication because the first element is an asterisk:

  ```
  authentication_policy='*,,'
  ```

  For information about configuring the `authentication_policy` value, see Configuring the Multifactor Authentication Policy.

To use `authentication_fido` as a passwordless authentication method, the account must be created with `authentication_fido` as the first factor authentication method. The `INITIAL AUTHENTICATION IDENTIFIED BY` clause must also be specified for the first factor (it is not supported with 2nd or 3rd factors). This clause specifies whether a randomly generated or user-specified password will be used for FIDO device registration. After device registration, the server deletes the password and modifies the account to make `authentication_fido` the sole authentication method (the 1FA method).

The required `CREATE USER` syntax is as follows:

```
CREATE USER user
  IDENTIFIED WITH authentication_fido
  INITIAL AUTHENTICATION IDENTIFIED BY {RANDOM PASSWORD | 'auth_string'};
```

The following example uses the `RANDOM PASSWORD` syntax:

```
mysql> CREATE USER 'u1'@'localhost'
         IDENTIFIED WITH authentication_fido
         INITIAL AUTHENTICATION IDENTIFIED BY RANDOM PASSWORD;
+------+-----------+----------------------+-------------+
| user | host      | generated password   | auth_factor |
+------+-----------+----------------------+-------------+
| u1   | localhost | 9XHK]M{l2rnD;VXyHzeF |           1 |
+------+-----------+----------------------+-------------+
```

To perform registration, the user must authenticate to the server with the password associated with the `INITIAL AUTHENTICATION IDENTIFIED BY` clause, either the randomly generated password, or the '*auth_string*' value. If the account was created as just shown, the user executes this command and pastes in the preceding randomly generated password (`9XHK]M{l2rnD;VXyHzeF`) at the prompt:

```
$> mysql --user=u1 --password --fido-register-factor=2
Enter password:
```

The option `--fido-register-factor=2` is used because the `INITIAL AUTHENTICATION IDENTIFIED BY` clause is currently acting as the first factor authentication method. The user must therefore provide the temporary password by using the second factor. On a successful registration,

the server removes the temporary password and revises the account entry in the `mysql.user` system table to list `authentication_fido` as the sole (1FA) authentication method.

When creating a passwordless-authentication account, it is important to include the `INITIAL AUTHENTICATION IDENTIFIED BY` clause in the `CREATE USER` statement. The server will accept a statement without the clause, but the resulting account is unusable because there is no way to connect to the server to register the device. Suppose that you execute a statement like this:

```
CREATE USER 'u2'@'localhost'
  IDENTIFIED WITH authentication_fido;
```

Subsequent attempts to use the account to connect fail like this:

```
$> mysql --user=u2 --skip-password
Failed to open FIDO device.
ERROR 1 (HY000): Unknown MySQL error
```

Note

Passwordless authentication is achieved using the Universal 2nd Factor (U2F) protocol, which does not support additional security measures such as setting a PIN on the device to be registered. It is therefore the responsibility of the device holder to ensure the device is handled in a secure manner.

FIDO Device Unregistration

It is possible to unregister FIDO devices associated with a MySQL account. This might be desirable or necessary under multiple circumstances:

- A FIDO device is to be replaced with a different device. The previous device must be unregistered and the new device registered.

  In this case, the account owner or any user who has the `CREATE USER` privilege can unregister the device. The account owner can register the new device.

- A FIDO device is reset or lost. Authentication attempts will fail until the current device is unregistered and a new registration is performed.

  In this case, the account owner, being unable to authenticate, cannot unregister the current device and must contact the DBA (or any user who has the `CREATE USER` privilege) to do so. Then the account owner can reregister the reset device or register a new device.

Unregistering a FIDO device can be done by the account owner or by any user who has the `CREATE USER` privilege. Use this syntax:

```
ALTER USER user {2 | 3} FACTOR UNREGISTER;
```

To re-register a device or perform a new registration, refer to the instructions in Using FIDO Authentication.

How FIDO Authentication of MySQL Users Works

This section provides an overview of how MySQL and FIDO work together to authenticate MySQL users. For examples showing how to set up MySQL accounts to use the FIDO authentication plugins, see Using FIDO Authentication.

An account that uses FIDO authentication must perform an initial device registration step before it can connect to the server. After the device has been registered, authentication can proceed. FIDO device registration process is as follows:

1. The server sends a random challenge, user ID, and relying party ID (which uniquely identifies a server) to the client. The relying party ID is defined by the `authentication_fido_rp_id` system variable. The default value is `MySQL`.

2. The client receives that information and sends it to the client-side FIDO authentication plugin, which in turn provides it to the FIDO device.
3. After the user has performed the appropriate device action (for example, touching the device or performing a biometric scan) the FIDO device generates a public/private key pair, a key handle, an X.509 certificate, and a signature, which is returned to the server.
4. The server-side FIDO authentication plugin verifies the signature. Upon successful verification, the server stores the credential ID and public key in the `mysql.user` system table.

After registration has been performed successfully, FIDO authentication follows this process:

1. The server sends a random challenge, user ID, relying party ID and credentials to the client.
2. The client sends the same information to the FIDO device.
3. The FIDO device prompts the user to perform the appropriate device action, based on the selection made during registration.
4. This action unlocks the private key and the challenge is signed.
5. This signed challenge is returned to the server.
6. The server-side FIDO authentication plugin verifies the signature with the public key and responds to indicate authentication success or failure.

6.4.1.12 Test Pluggable Authentication

MySQL includes a test plugin that checks account credentials and logs success or failure to the server error log. This is a loadable plugin (not built in) and must be installed prior to use.

The test plugin source code is separate from the server source, unlike the built-in native plugin, so it can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

Note

This plugin is intended for testing and development purposes, and is not for use in production environments or on servers that are exposed to public networks.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.28 Plugin and Library Names for Test Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`test_plugin_server`
Client-side plugin	`auth_test_plugin`
Library file	`auth_test_plugin.so`

The following sections provide installation and usage information specific to test pluggable authentication:

- Installing Test Pluggable Authentication
- Uninstalling Test Pluggable Authentication
- Using Test Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Installing Test Pluggable Authentication

This section describes how to install the server-side test authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=auth_test_plugin.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN test_plugin_server SONAME 'auth_test_plugin.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

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

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the test plugin, see Using Test Pluggable Authentication.

Uninstalling Test Pluggable Authentication

The method used to uninstall the test authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

  ```
  UNINSTALL PLUGIN test_plugin_server;
  ```

Using Test Pluggable Authentication

To use the test authentication plugin, create an account and name that plugin in the `IDENTIFIED WITH` clause:

```
CREATE USER 'testuser'@'localhost'
IDENTIFIED WITH test_plugin_server
BY 'testpassword';
```

The test authentication plugin also requires creating a proxy user as follows:

```
CREATE USER testpassword@localhost;
GRANT PROXY ON testpassword@localhost TO testuser@localhost;
```

Then provide the `--user` and `--password` options for that account when you connect to the server. For example:

```
$> mysql --user=testuser --password
Enter password: testpassword
```

The plugin fetches the password as received from the client and compares it with the value stored in the `authentication_string` column of the account row in the `mysql.user` system table. If the two values match, the plugin returns the `authentication_string` value as the new effective user ID.

You can look in the server error log for a message indicating whether authentication succeeded (notice that the password is reported as the “user”):

```
[Note] Plugin test_plugin_server reported:
'successfully authenticated user testpassword'
```

6.4.1.13 Pluggable Authentication System Variables

These variables are unavailable unless the appropriate server-side plugin is installed:

- `authentication_ldap_sasl` for system variables with names of the form `authentication_ldap_sasl_xxx`
- `authentication_ldap_simple` for system variables with names of the form `authentication_ldap_simple_xxx`

Table 6.29 Authentication Plugin System Variable Summary

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
authentication_fido_rp_id	Yes	Yes	Yes		Global	Yes
authentication_kerberos_service_key_tab	Yes	Yes	Yes		Global	No
authentication_kerberos_service_principal	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_auth_method_name	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_bind_base_dn	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_bind_root_dn	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_bind_root_pwd	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_ca_path	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_group_search_attr	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_group_search_filter	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_init_pool_size	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_log_status	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_max_pool_size	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_referrals	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_server_host	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_server_port	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_tls	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_user_search_attr	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_auth_method_name	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_bind_base_dn	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_bind_root_dn	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_bind_root_pwd	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
authentication_ldap_simple_ca_path	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_group_search_attr	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_group_search_filter	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_init_pool_size	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_log_status	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_max_pool_size	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_referral	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_server_host	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_server_port	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_tls	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_user_search_attr	Yes	Yes	Yes		Global	Yes
authentication_oci_policy	Yes	Yes	Yes		Global	Yes
authentication_windows_log_level	Yes	Yes	Yes		Global	No
authentication_windows_use_principal_name	Yes	Yes	Yes		Global	No

- `authentication_fido_rp_id`

Command-Line Format	`--authentication-fido-rp-id=value`
Introduced	8.0.27
System Variable	`authentication_fido_rp_id`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`MySQL`

This variable specifies the relying party ID used for FIDO device registration and FIOD authentication. If FIDO authentication is attempted and this value is not the one expected by the FIDO device, the device assumes that it is not talking to the correct server and an error occurs. The maximum value length is 255 characters.

- `authentication_kerberos_service_key_tab`

Command-Line Format	`--authentication-kerberos-service-key-tab=file_name`
Introduced	8.0.26
System Variable	`authentication_kerberos_service_key_tab`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	File name
Default Value	`datadir/mysql.keytab`

The name of the server-side key-table (“keytab”) file containing Kerberos service keys to authenticate MySQL service tickets received from clients. The file name should be given as an absolute path name. If this variable is not set, the default is `mysql.keytab` in the data directory.

The file must exist and contain a valid key for the service principal name (SPN) or authentication of clients will fail. (The SPN and same key also must be created in the Kerberos server.) The file may contain multiple service principal names and their respective key combinations.

The file must be generated by the Kerberos server administrator and be copied to a location accessible by the MySQL server. The file can be validated to make sure that it is correct and was copied properly using this command:

```
klist -k file_name
```

For information about keytab files, see https://web.mit.edu/kerberos/krb5-latest/doc/basic/keytab_def.html.

- `authentication_kerberos_service_principal`

Command-Line Format	`--authentication-kerberos-service-principal=name`
Introduced	8.0.26
System Variable	`authentication_kerberos_service_principal`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`mysql/host_name@realm_name`

The Kerberos service principal name (SPN) that the MySQL server sends to clients.

The value is composed from the service name (`mysql`), a host name, and a realm name. The default value is `mysql/host_name@realm_name`. The realm in the service principal name enables retrieving the exact service key.

To use a nondefault value, set the value using the same format. For example, to use a host name of `krbauth.example.com` and a realm of `MYSQL.LOCAL`, set `authentication_kerberos_service_principal` to `mysql/krbauth.example.com@MYSQL.LOCAL`.

The service principal name and service key must already be present in the database managed by the KDC server.

There can be service principal names that differ only by realm name.

- `authentication_ldap_sasl_auth_method_name`

Command-Line Format	`--authentication-ldap-sasl-auth-method-name=value`
System Variable	`authentication_ldap_sasl_auth_method_name`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`SCRAM-SHA-1`

Valid Values (≥ 8.0.23)	`SCRAM-SHA-1`
	
	`SCRAM-SHA-256`

	`GSSAPI`

Valid Values (≥ 8.0.20, ≤ 8.0.22)	`SCRAM-SHA-1`
	
	`GSSAPI`

Valid Values (≤ 8.0.19)	`SCRAM-SHA-1`

For SASL LDAP authentication, the authentication method name. Communication between the authentication plugin and the LDAP server occurs according to this authentication method to ensure password security.

These authentication method values are permitted:

- `SCRAM-SHA-1`: Use a SASL challenge-response mechanism.

  The client-side `authentication_ldap_sasl_client` plugin communicates with the SASL server, using the password to create a challenge and obtain a SASL request buffer, then passes this buffer to the server-side `authentication_ldap_sasl` plugin. The client-side and server-side SASL LDAP plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

- `SCRAM-SHA-256`: Use a SASL challenge-response mechanism.

  This method is similar to `SCRAM-SHA-1`, but is more secure. It is available in MySQL 8.0.23 and higher. It requires an OpenLDAP server built using Cyrus SASL 2.1.27 or higher.

- `GSSAPI`: Use Kerberos, a passwordless and ticket-based protocol.

  GSSAPI/Kerberos is supported as an authentication method for MySQL clients and servers only on Linux. It is useful in Linux environments where applications access LDAP using Microsoft Active Directory, which has Kerberos enabled by default.

  The client-side `authentication_ldap_sasl_client` plugin obtains a service ticket using the ticket-granting ticket (TGT) from Kerberos, but does not use LDAP services directly. The server-side `authentication_ldap_sasl` plugin routes Kerberos messages between the client-side plugin and the LDAP server. Using the credentials thus obtained, the server-side plugin then communicates with the LDAP server to interpret LDAP authentication messages and retrieve LDAP groups.

- `authentication_ldap_sasl_bind_base_dn`

Command-Line Format	`--authentication-ldap-sasl-bind-base-dn=value`
System Variable	`authentication_ldap_sasl_bind_base_dn`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

Default Value	NULL

For SASL LDAP authentication, the base distinguished name (DN). This variable can be used to limit the scope of searches by anchoring them at a certain location (the “base”) within the search tree.

Suppose that members of one set of LDAP user entries each have this form:

```
uid=user_name,ou=People,dc=example,dc=com
```

And that members of another set of LDAP user entries each have this form:

```
uid=user_name,ou=Admin,dc=example,dc=com
```

Then searches work like this for different base DN values:

- If the base DN is `ou=People,dc=example,dc=com`: Searches find user entries only in the first set.
- If the base DN is `ou=Admin,dc=example,dc=com`: Searches find user entries only in the second set.
- If the base DN is `ou=dc=example,dc=com`: Searches find user entries in the first or second set.

In general, more specific base DN values result in faster searches because they limit the search scope more.

- `authentication_ldap_sasl_bind_root_dn`

Command-Line Format	`--authentication-ldap-sasl-bind-root-dn=value`
System Variable	`authentication_ldap_sasl_bind_root_dn`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	NULL

For SASL LDAP authentication, the root distinguished name (DN). This variable is used in conjunction with `authentication_ldap_sasl_bind_root_pwd` as the credentials for authenticating to the LDAP server for the purpose of performing searches. Authentication uses either one or two LDAP bind operations, depending on whether the MySQL account names an LDAP user DN:

- If the account does not name a user DN: `authentication_ldap_sasl` performs an initial LDAP binding using `authentication_ldap_sasl_bind_root_dn` and `authentication_ldap_sasl_bind_root_pwd`. (These are both empty by default, so if they are not set, the LDAP server must permit anonymous connections.) The resulting bind LDAP handle is used to search for the user DN, based on the client user name. `authentication_ldap_sasl` performs a second bind using the user DN and client-supplied password.
- If the account does name a user DN: The first bind operation is unnecessary in this case. `authentication_ldap_sasl` performs a single bind using the user DN and client-supplied password. This is faster than if the MySQL account does not specify an LDAP user DN.

- `authentication_ldap_sasl_bind_root_pwd`

Command-Line Format	`--authentication-ldap-sasl-bind-root-pwd=value`
System Variable	`authentication_ldap_sasl_bind_root_pwd`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`NULL`

For SASL LDAP authentication, the password for the root distinguished name. This variable is used in conjunction with `authentication_ldap_sasl_bind_root_dn`. See the description of that variable.

- `authentication_ldap_sasl_ca_path`

Command-Line Format	`--authentication-ldap-sasl-ca-path=value`
System Variable	`authentication_ldap_sasl_ca_path`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`NULL`

For SASL LDAP authentication, the absolute path of the certificate authority file. Specify this file if it is desired that the authentication plugin perform verification of the LDAP server certificate.

Note

In addition to setting the `authentication_ldap_sasl_ca_path` variable to the file name, you must add the appropriate certificate authority certificates to the file and enable the `authentication_ldap_sasl_tls` system variable. These variables can be set to override the default OpenLDAP TLS configuration; see LDAP Pluggable Authentication and ldap.conf

- `authentication_ldap_sasl_group_search_attr`

Command-Line Format	`--authentication-ldap-sasl-group-search-attr=value`
System Variable	`authentication_ldap_sasl_group_search_attr`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`cn`

For SASL LDAP authentication, the name of the attribute that specifies group names in LDAP directory entries. If `authentication_ldap_sasl_group_search_attr` has its default value of `cn`, searches return the `cn` value as the group name. For example, if an LDAP entry with a `uid` value of `user1` has a `cn` attribute of `mygroup`, searches for `user1` return `mygroup` as the group name.

This variable should be the empty string if you want no group or proxy authentication.

If the group search attribute is `isMemberOf`, LDAP authentication directly retrieves the user attribute `isMemberOf` value and assigns it as group information. If the group search attribute is not `isMemberOf`, LDAP authentication searches for all groups where the user is a member. (The latter is the default behavior.) This behavior is based on how LDAP group information can be stored two ways: 1) A group entry can have an attribute named `memberUid` or `member` with a value that is a user name; 2) A user entry can have an attribute named `isMemberOf` with values that are group names.

- `authentication_ldap_sasl_group_search_filter`

Command-Line Format	`--authentication-ldap-sasl-group-search-filter=value`
System Variable	`authentication_ldap_sasl_group_search_filte`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`(|(&(objectClass=posixGroup)(memberUid=%s))(&(objectClass=group)(member=%s)))`

For SASL LDAP authentication, the custom group search filter.

The search filter value can contain `{UA}` and `{UD}` notation to represent the user name and the full user DN. For example, `{UA}` is replaced with a user name such as `"admin"`, whereas `{UD}` is replaced with a use full DN such as `"uid=admin,ou=People,dc=example,dc=com"`. The following value is the default, which supports both OpenLDAP and Active Directory:

```
(|(&(objectClass=posixGroup)(memberUid={UA}))
  (&(objectClass=group)(member={UD})))
```

In some cases for the user scenario, `memberOf` is a simple user attribute that holds no group information. For additional flexibility, an optional `{GA}` prefix can be used with the group search attribute. Any group attribute with a {GA} prefix is treated as a user attribute having group names. For example, with a value of `{GA}MemberOf`, if the group value is the DN, the first attribute value from the group DN is returned as the group name.

- `authentication_ldap_sasl_init_pool_size`

Command-Line Format	`--authentication-ldap-sasl-init-pool-size=#`
System Variable	`authentication_ldap_sasl_init_pool_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`32767`

Unit	connections

For SASL LDAP authentication, the initial size of the pool of connections to the LDAP server. Choose the value for this variable based on the average number of concurrent authentication requests to the LDAP server.

The plugin uses `authentication_ldap_sasl_init_pool_size` and `authentication_ldap_sasl_max_pool_size` together for connection-pool management:

- When the authentication plugin initializes, it creates `authentication_ldap_sasl_init_pool_size` connections, unless `authentication_ldap_sasl_max_pool_size=0` to disable pooling.
- If the plugin receives an authentication request when there are no free connections in the current connection pool, the plugin can create a new connection, up to the maximum connection pool size given by `authentication_ldap_sasl_max_pool_size`.
- If the plugin receives a request when the pool size is already at its maximum and there are no free connections, authentication fails.
- When the plugin unloads, it closes all pooled connections.

Changes to plugin system variable settings may have no effect on connections already in the pool. For example, modifying the LDAP server host, port, or TLS settings does not affect existing connections. However, if the original variable values were invalid and the connection pool could not be initialized, the plugin attempts to reinitialize the pool for the next LDAP request. In this case, the new system variable values are used for the reinitialization attempt.

If `authentication_ldap_sasl_max_pool_size=0` to disable pooling, each LDAP connection opened by the plugin uses the values the system variables have at that time.

- `authentication_ldap_sasl_log_status`

Command-Line Format	`--authentication-ldap-sasl-log-status=#`
System Variable	`authentication_ldap_sasl_log_status`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value (≥ 8.0.18)	`6`
Maximum Value (≤ 8.0.17)	`5`

For SASL LDAP authentication, the logging level for messages written to the error log. The following table shows the permitted level values and their meanings.

Table 6.30 Log Levels for authentication_ldap_sasl_log_status

Option Value	Types of Messages Logged
`1`	No messages
`2`	Error messages
`3`	Error and warning messages

Option Value	Types of Messages Logged
4	Error, warning, and information messages
5	Same as previous level plus debugging messages from MySQL
6	Same as previous level plus debugging messages from LDAP library

Log level 6 is available as of MySQL 8.0.18.

On the client side, messages can be logged to the standard output by setting the `AUTHENTICATION_LDAP_CLIENT_LOG` environment variable. The permitted and default values are the same as for `authentication_ldap_sasl_log_status`.

The `AUTHENTICATION_LDAP_CLIENT_LOG` environment variable applies only to SASL LDAP authentication. It has no effect for simple LDAP authentication because the client plugin in that case is `mysql_clear_password`, which knows nothing about LDAP operations.

- `authentication_ldap_sasl_max_pool_size`

Command-Line Format	`--authentication-ldap-sasl-max-pool-size=#`
System Variable	`authentication_ldap_sasl_max_pool_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1000`
Minimum Value	`0`
Maximum Value	`32767`
Unit	connections

For SASL LDAP authentication, the maximum size of the pool of connections to the LDAP server. To disable connection pooling, set this variable to 0.

This variable is used in conjunction with `authentication_ldap_sasl_init_pool_size`. See the description of that variable.

- `authentication_ldap_sasl_referral`

Command-Line Format	`--authentication-ldap-sasl-referral[={OFF|ON}]`
Introduced	8.0.20
System Variable	`authentication_ldap_sasl_referral`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean

Default Value	OFF

For SASL LDAP authentication, whether to enable LDAP search referral. See LDAP Search Referral.

This variable can be set to override the default OpenLDAP referral configuration; see LDAP Pluggable Authentication and ldap.conf

- authentication_ldap_sasl_server_host

Command-Line Format	--authentication-ldap-sasl-server-host=host_name
System Variable	authentication_ldap_sasl_server_host
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

For SASL LDAP authentication, the LDAP server host. The permitted values for this variable depend on the authentication method:

- For authentication_ldap_sasl_auth_method_name=SCRAM-SHA-1: The LDAP server host can be a host name or IP address.
- For authentication_ldap_sasl_auth_method_name=SCRAM-SHA-256: The LDAP server host can be a host name or IP address.

- authentication_ldap_sasl_server_port

Command-Line Format	--authentication-ldap-sasl-server-port=port_num
System Variable	authentication_ldap_sasl_server_port
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	389
Minimum Value	1
Maximum Value	32376

For SASL LDAP authentication, the LDAP server TCP/IP port number.

As of MySQL 8.0.14, if the LDAP port number is configured as 636 or 3269, the plugin uses LDAPS (LDAP over SSL) instead of LDAP. (LDAPS differs from startTLS.)

- authentication_ldap_sasl_tls

Command-Line Format	--authentication-ldap-sasl-tls[={OFF|ON}]
System Variable	authentication_ldap_sasl_tls
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Type	Boolean
Default Value	`OFF`

For SASL LDAP authentication, whether connections by the plugin to the LDAP server are secure. If this variable is enabled, the plugin uses TLS to connect securely to the LDAP server. This variable can be set to override the default OpenLDAP TLS configuration; see LDAP Pluggable Authentication and ldap.conf If you enable this variable, you may also wish to set the `authentication_ldap_sasl_ca_path` variable.

MySQL LDAP plugins support the StartTLS method, which initializes TLS on top of a plain LDAP connection.

As of MySQL 8.0.14, LDAPS can be used by setting the `authentication_ldap_sasl_server_port` system variable.

- `authentication_ldap_sasl_user_search_attr`

Command-Line Format	`--authentication-ldap-sasl-user-search-attr=value`
System Variable	`authentication_ldap_sasl_user_search_attr`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`uid`

For SASL LDAP authentication, the name of the attribute that specifies user names in LDAP directory entries. If a user distinguished name is not provided, the authentication plugin searches for the name using this attribute. For example, if the `authentication_ldap_sasl_user_search_attr` value is `uid`, a search for the user name `user1` finds entries with a `uid` value of `user1`.

- `authentication_ldap_simple_auth_method_name`

Command-Line Format	`--authentication-ldap-simple-auth-method-name=value`
System Variable	`authentication_ldap_simple_auth_method_name`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`SIMPLE`

Valid Values	`SIMPLE`
	
	`AD-FOREST`

For simple LDAP authentication, the authentication method name. Communication between the authentication plugin and the LDAP server occurs according to this authentication method.

Note

For all simple LDAP authentication methods, it is recommended to also set TLS parameters to require that communication with the LDAP server take place over secure connections.

These authentication method values are permitted:

- `SIMPLE`: Use simple LDAP authentication. This method uses either one or two LDAP bind operations, depending on whether the MySQL account names an LDAP user distinguished name. See the description of `authentication_ldap_simple_bind_root_dn`.
- `AD-FOREST`: A variation on `SIMPLE`, such that authentication searches all domains in the Active Directory forest, performing an LDAP bind to each Active Directory domain until the user is found in some domain.

- `authentication_ldap_simple_bind_base_dn`

Command-Line Format	`--authentication-ldap-simple-bind-base-dn=value`
System Variable	`authentication_ldap_simple_bind_base_dn`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`NULL`

For simple LDAP authentication, the base distinguished name (DN). This variable can be used to limit the scope of searches by anchoring them at a certain location (the “base”) within the search tree.

Suppose that members of one set of LDAP user entries each have this form:

```
uid=user_name,ou=People,dc=example,dc=com
```

And that members of another set of LDAP user entries each have this form:

```
uid=user_name,ou=Admin,dc=example,dc=com
```

Then searches work like this for different base DN values:

- If the base DN is `ou=People,dc=example,dc=com`: Searches find user entries only in the first set.
- If the base DN is `ou=Admin,dc=example,dc=com`: Searches find user entries only in the second set.
- If the base DN is `ou=dc=example,dc=com`: Searches find user entries in the first or second set.

In general, more specific base DN values result in faster searches because they limit the search scope more.

- `authentication_ldap_simple_bind_root_dn`

Command-Line Format	`--authentication-ldap-simple-bind-root-dn=value`
System Variable	`authentication_ldap_simple_bind_root_dn`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

Default Value	NULL

For simple LDAP authentication, the root distinguished name (DN). This variable is used in conjunction with authentication_ldap_simple_bind_root_pwd as the credentials for authenticating to the LDAP server for the purpose of performing searches. Authentication uses either one or two LDAP bind operations, depending on whether the MySQL account names an LDAP user DN:

- If the account does not name a user DN: authentication_ldap_simple performs an initial LDAP binding using authentication_ldap_simple_bind_root_dn and authentication_ldap_simple_bind_root_pwd. (These are both empty by default, so if they are not set, the LDAP server must permit anonymous connections.) The resulting bind LDAP handle is used to search for the user DN, based on the client user name. authentication_ldap_simple performs a second bind using the user DN and client-supplied password.
- If the account does name a user DN: The first bind operation is unnecessary in this case. authentication_ldap_simple performs a single bind using the user DN and client-supplied password. This is faster than if the MySQL account does not specify an LDAP user DN.

- authentication_ldap_simple_bind_root_pwd

Command-Line Format	--authentication-ldap-simple-bind-root-pwd=value
System Variable	authentication_ldap_simple_bind_root_pwd
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	NULL

For simple LDAP authentication, the password for the root distinguished name. This variable is used in conjunction with authentication_ldap_simple_bind_root_dn. See the description of that variable.

- authentication_ldap_simple_ca_path

Command-Line Format	--authentication-ldap-simple-ca-path=value
System Variable	authentication_ldap_simple_ca_path
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	NULL

For simple LDAP authentication, the absolute path of the certificate authority file. Specify this file if it is desired that the authentication plugin perform verification of the LDAP server certificate.

Note

In addition to setting the authentication_ldap_simple_ca_path variable to the file name, you must add the appropriate certificate authority certificates to the file and enable the authentication_ldap_simple_tls

system variable. These variables can be set to override the default OpenLDAP TLS configuration; see LDAP Pluggable Authentication and ldap.conf

- `authentication_ldap_simple_group_search_attr`

Command-Line Format	`--authentication-ldap-simple-group-search-attr=value`
System Variable	`authentication_ldap_simple_group_search_attr`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`cn`

For simple LDAP authentication, the name of the attribute that specifies group names in LDAP directory entries. If `authentication_ldap_simple_group_search_attr` has its default value of `cn`, searches return the `cn` value as the group name. For example, if an LDAP entry with a `uid` value of `user1` has a `cn` attribute of `mygroup`, searches for `user1` return `mygroup` as the group name.

If the group search attribute is `isMemberOf`, LDAP authentication directly retrieves the user attribute `isMemberOf` value and assigns it as group information. If the group search attribute is not `isMemberOf`, LDAP authentication searches for all groups where the user is a member. (The latter is the default behavior.) This behavior is based on how LDAP group information can be stored two ways: 1) A group entry can have an attribute named `memberUid` or `member` with a value that is a user name; 2) A user entry can have an attribute named `isMemberOf` with values that are group names.

- `authentication_ldap_simple_group_search_filter`

Command-Line Format	`--authentication-ldap-simple-group-search-filter=value`
System Variable	`authentication_ldap_simple_group_search_filter`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`(|(&(objectClass=posixGroup)(memberUid=%s))(&(objectClass=group)(member=%s)))`

For simple LDAP authentication, the custom group search filter.

The search filter value can contain `{UA}` and `{UD}` notation to represent the user name and the full user DN. For example, `{UA}` is replaced with a user name such as `"admin"`, whereas `{UD}` is replaced with a use full DN such as `"uid=admin,ou=People,dc=example,dc=com"`. The following value is the default, which supports both OpenLDAP and Active Directory:

```
(|(&(objectClass=posixGroup)(memberUid={UA}))
  (&(objectClass=group)(member={UD})))
```

In some cases for the user scenario, `memberOf` is a simple user attribute that holds no group information. For additional flexibility, an optional `{GA}` prefix can be used with the group search attribute. Any group attribute with a {GA} prefix is treated as a user attribute having group names. For

example, with a value of `{GA}MemberOf`, if the group value is the DN, the first attribute value from the group DN is returned as the group name.

- `authentication_ldap_simple_init_pool_size`

Command-Line Format	`--authentication-ldap-simple-init-pool-size=#`
System Variable	`authentication_ldap_simple_init_pool_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`32767`
Unit	connections

For simple LDAP authentication, the initial size of the pool of connections to the LDAP server. Choose the value for this variable based on the average number of concurrent authentication requests to the LDAP server.

The plugin uses `authentication_ldap_simple_init_pool_size` and `authentication_ldap_simple_max_pool_size` together for connection-pool management:

- When the authentication plugin initializes, it creates `authentication_ldap_simple_init_pool_size` connections, unless `authentication_ldap_simple_max_pool_size=0` to disable pooling.
- If the plugin receives an authentication request when there are no free connections in the current connection pool, the plugin can create a new connection, up to the maximum connection pool size given by `authentication_ldap_simple_max_pool_size`.
- If the plugin receives a request when the pool size is already at its maximum and there are no free connections, authentication fails.
- When the plugin unloads, it closes all pooled connections.

Changes to plugin system variable settings may have no effect on connections already in the pool. For example, modifying the LDAP server host, port, or TLS settings does not affect existing connections. However, if the original variable values were invalid and the connection pool could not be initialized, the plugin attempts to reinitialize the pool for the next LDAP request. In this case, the new system variable values are used for the reinitialization attempt.

If `authentication_ldap_simple_max_pool_size=0` to disable pooling, each LDAP connection opened by the plugin uses the values the system variables have at that time.

- `authentication_ldap_simple_log_status`

Command-Line Format	`--authentication-ldap-simple-log-status=#`
System Variable	`authentication_ldap_simple_log_status`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No

Type	Integer
Default Value	1
Minimum Value	1
Maximum Value (≥ 8.0.18)	6
Maximum Value (≤ 8.0.17)	5

For simple LDAP authentication, the logging level for messages written to the error log. The following table shows the permitted level values and their meanings.

Table 6.31 Log Levels for authentication_ldap_simple_log_status

Option Value	Types of Messages Logged
1	No messages
2	Error messages
3	Error and warning messages
4	Error, warning, and information messages
5	Same as previous level plus debugging messages from MySQL
6	Same as previous level plus debugging messages from LDAP library

Log level 6 is available as of MySQL 8.0.18.

- `authentication_ldap_simple_max_pool_size`

Command-Line Format	`--authentication-ldap-simple-max-pool-size=#`
System Variable	`authentication_ldap_simple_max_pool_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	1000
Minimum Value	0
Maximum Value	32767
Unit	connections

For simple LDAP authentication, the maximum size of the pool of connections to the LDAP server. To disable connection pooling, set this variable to 0.

This variable is used in conjunction with `authentication_ldap_simple_init_pool_size`. See the description of that variable.

- `authentication_ldap_simple_referral`

Command-Line Format	`--authentication-ldap-simple-referral[={OFF|ON}]`
Introduced	8.0.20
System Variable	`authentication_ldap_simple_referral`

Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

For simple LDAP authentication, whether to enable LDAP search referral. See LDAP Search Referral.

- authentication_ldap_simple_server_host

Command-Line Format	--authentication-ldap-simple-server-host=host_name
System Variable	authentication_ldap_simple_server_host
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

For simple LDAP authentication, the LDAP server host. The permitted values for this variable depend on the authentication method:

- For authentication_ldap_simple_auth_method_name=SIMPLE: The LDAP server host can be a host name or IP address.
- For authentication_ldap_simple_auth_method_name=AD-FOREST. The LDAP server host can be an Active Directory domain name. For example, for an LDAP server URL of ldap://example.mem.local:389, the domain name can be mem.local.

  An Active Directory forest setup can have multiple domains (LDAP server IPs), which can be discovered using DNS. On Unix and Unix-like systems, some additional setup may be required to

configure your DNS server with SRV records that specify the LDAP servers for the Active Directory domain. For information about DNS SRV, see RFC 2782.

Suppose that your configuration has these properties:

- The name server that provides information about Active Directory domains has IP address `10.172.166.100`.
- The LDAP servers have names `ldap1.mem.local` through `ldap3.mem.local` and IP addresses `10.172.166.101` through `10.172.166.103`.

You want the LDAP servers to be discoverable using SRV searches. For example, at the command line, a command like this should list the LDAP servers:

```
host -t SRV _ldap._tcp.mem.local
```

Perform the DNS configuration as follows:

1. Add a line to `/etc/resolv.conf` to specify the name server that provides information about Active Directory domains:

   ```
   nameserver 10.172.166.100
   ```

2. Configure the appropriate zone file for the name server with SRV records for the LDAP servers:

   ```
   _ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap1.mem.local.
   _ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap2.mem.local.
   _ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap3.mem.local.
   ```

3. It may also be necessary to specify the IP address for the LDAP servers in `/etc/hosts` if the server host cannot be resolved. For example, add lines like this to the file:

   ```
   10.172.166.101 ldap1.mem.local
   10.172.166.102 ldap2.mem.local
   10.172.166.103 ldap3.mem.local
   ```

With the DNS configured as just described, the server-side LDAP plugin can discover the LDAP servers and tries to authenticate in all domains until authentication succeeds or there are no more servers.

Windows needs no such settings as just described. Given the LDAP server host in the `authentication_ldap_simple_server_host` value, the Windows LDAP library searches all domains and attempts to authenticate.

- `authentication_ldap_simple_server_port`

Command-Line Format	`--authentication-ldap-simple-server-port=port_num`
System Variable	`authentication_ldap_simple_server_port`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`389`
Minimum Value	`1`

Maximum Value	32376

For simple LDAP authentication, the LDAP server TCP/IP port number.

As of MySQL 8.0.14, if the LDAP port number is configured as 636 or 3269, the plugin uses LDAPS (LDAP over SSL) instead of LDAP. (LDAPS differs from startTLS.)

- authentication_ldap_simple_tls

Command-Line Format	--authentication-ldap-simple-tls[={OFF|ON}]
System Variable	authentication_ldap_simple_tls
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

For simple LDAP authentication, whether connections by the plugin to the LDAP server are secure. If this variable is enabled, the plugin uses TLS to connect securely to the LDAP server. This variable can be set to override the default OpenLDAP TLS configuration; see LDAP Pluggable Authentication and ldap.conf If you enable this variable, you may also wish to set the authentication_ldap_simple_ca_path variable.

MySQL LDAP plugins support the StartTLS method, which initializes TLS on top of a plain LDAP connection.

As of MySQL 8.0.14, LDAPS can be used by setting the authentication_ldap_simple_server_port system variable.

- authentication_ldap_simple_user_search_attr

Command-Line Format	--authentication-ldap-simple-user-search-attr=value
System Variable	authentication_ldap_simple_user_search_attr
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	uid

For simple LDAP authentication, the name of the attribute that specifies user names in LDAP directory entries. If a user distinguished name is not provided, the authentication plugin searches for the name using this attribute. For example, if the authentication_ldap_simple_user_search_attr value is uid, a search for the user name user1 finds entries with a uid value of user1.

6.4.2 The Connection-Control Plugins

MySQL Server includes a plugin library that 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.

  The `CONNECTION_CONTROL` plugin uses the audit plugin interface (see Writing Audit Plugins). To collect information, it subscribes to the `MYSQL_AUDIT_CONNECTION_CLASSMASK` event class, and processes `MYSQL_AUDIT_CONNECTION_CONNECT` and `MYSQL_AUDIT_CONNECTION_CHANGE_USER` subevents to check whether the server should introduce a delay before responding to connection attempts.

- `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` implements an `INFORMATION_SCHEMA` table that exposes more detailed monitoring information for failed connection attempts.

The following sections provide information about connection-control plugin installation and configuration. For information about the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table, see Section 26.6.2, “The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table”.

6.4.2.1 Connection-Control Plugin Installation

This section describes how to install the connection-control plugins, `CONNECTION_CONTROL` and `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS`. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `connection_control`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugins at server startup, use the `--plugin-load-add` option to name the library file that contains them. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=connection_control.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugins at runtime, use these statements, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN CONNECTION_CONTROL
  SONAME 'connection_control.so';
INSTALL PLUGIN CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS
  SONAME 'connection_control.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
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        |
+---------------------------------------+---------------+
```

If a plugin fails to initialize, check the server error log for diagnostic messages.

If the plugins have been previously registered with `INSTALL PLUGIN` or are loaded with `--plugin-load-add`, you can use the `--connection-control` and `--connection-control-failed-login-attempts` options at server startup to control plugin activation. For example, to load the plugins at startup and prevent them from being removed at runtime, use these options:

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

If it is desired to prevent the server from running without a given connection-control plugin, use an option value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

Note

It is possible to install one plugin without the other, but both must be installed for full connection-control capability. In particular, installing only the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` plugin is of little use because, without the `CONNECTION_CONTROL` plugin to provide the data that populates the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table, the table is always empty.

- Connection Delay Configuration
- Connection Failure Assessment
- Connection Failure Monitoring

Connection Delay Configuration

To enable configuring its operation, the `CONNECTION_CONTROL` plugin exposes these system variables:

- `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. To disable failed-connection counting, set `connection_control_failed_connections_threshold` to zero.
- `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.

If `connection_control_failed_connections_threshold` is nonzero, failed-connection counting is enabled and has these properties:

- The delay is zero up through `connection_control_failed_connections_threshold` consecutive failed connection attempts.
- Thereafter, the server adds an increasing delay for subsequent consecutive attempts, until a successful connection occurs. The initial unadjusted delays begin at 1000 milliseconds (1 second) and increase by 1000 milliseconds per attempt. That is, once delay has been activated for an account, the unadjusted delays for subsequent failed attempts are 1000 milliseconds, 2000 milliseconds, 3000 milliseconds, and so forth.
- The actual delay experienced by a client is the unadjusted delay, adjusted to lie within the values of the `connection_control_min_connection_delay` and `connection_control_max_connection_delay` system variables, inclusive.

- Once delay has been activated for an account, the first successful connection thereafter by the account also experiences a delay, but failure counting is reset for subsequent connections.

For example, with the default `connection_control_failed_connections_threshold` value of 3, there is no delay for the first three consecutive failed connection attempts by an account. The actual adjusted delays experienced by the account for the fourth and subsequent failed connections depend on the `connection_control_min_connection_delay` and `connection_control_max_connection_delay` values:

- If `connection_control_min_connection_delay` and `connection_control_max_connection_delay` are 1000 and 20000, the adjusted delays are the same as the unadjusted delays, up to a maximum of 20000 milliseconds. The fourth and subsequent failed connections are delayed by 1000 milliseconds, 2000 milliseconds, 3000 milliseconds, and so forth.
- If `connection_control_min_connection_delay` and `connection_control_max_connection_delay` are 1500 and 20000, the adjusted delays for the fourth and subsequent failed connections are 1500 milliseconds, 2000 milliseconds, 3000 milliseconds, and so forth, up to a maximum of 20000 milliseconds.
- If `connection_control_min_connection_delay` and `connection_control_max_connection_delay` are 2000 and 3000, the adjusted delays for the fourth and subsequent failed connections are 2000 milliseconds, 2000 milliseconds, and 3000 milliseconds, with all subsequent failed connections also delayed by 3000 milliseconds.

You can set the `CONNECTION_CONTROL` system variables at server startup or runtime. Suppose that you want to permit four consecutive failed connection attempts before the server starts delaying its responses, with a minimum delay of 2000 milliseconds. To set the relevant variables at server startup, put these lines in the server `my.cnf` file:

```
[mysqld]
plugin-load-add=connection_control.so
connection_control_failed_connections_threshold=4
connection_control_min_connection_delay=2000
```

To set and persist the variables at runtime, use these statements:

```
SET PERSIST connection_control_failed_connections_threshold = 4;
SET PERSIST connection_control_min_connection_delay = 2000;
```

`SET PERSIST` sets a value for the running MySQL instance. It also saves the value, causing it to carry over to subsequent server restarts. To change a value for the running MySQL instance without having it carry over to subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

The `connection_control_min_connection_delay` and `connection_control_max_connection_delay` system variables both have minimum and maximum values of 1000 and 2147483647. In addition, the permitted range of values of each variable also depends on the current value of the other:

- `connection_control_min_connection_delay` cannot be set greater than the current value of `connection_control_max_connection_delay`.
- `connection_control_max_connection_delay` cannot be set less than the current value of `connection_control_min_connection_delay`.

Thus, to make the changes required for some configurations, you might need to set the variables in a specific order. Suppose that the current minimum and maximum delays are 1000 and 2000, and that you want to set them to 3000 and 5000. You cannot first set `connection_control_min_connection_delay` to 3000 because that is greater than the current `connection_control_max_connection_delay` value of 2000. Instead, set `connection_control_max_connection_delay` to 5000, then set `connection_control_min_connection_delay` to 3000.

Connection Failure Assessment

When the `CONNECTION_CONTROL` plugin is installed, it checks connection attempts and tracks whether they fail or succeed. For this purpose, a failed connection attempt is one for which the client user and host match a known MySQL account but the provided credentials are incorrect, or do not match any known account.

Failed-connection counting is based on the user/host combination for each connection attempt. Determination of the applicable user name and host name takes proxying into account and occurs as follows:

- If the client user proxies another user, the account for failed-connection counting is the proxying user, not the proxied user. For example, if `external_user@example.com` proxies `proxy_user@example.com`, connection counting uses the proxying user, `external_user@example.com`, rather than the proxied user, `proxy_user@example.com`. Both `external_user@example.com` and `proxy_user@example.com` must have valid entries in the `mysql.user` system table and a proxy relationship between them must be defined in the `mysql.proxies_priv` system table (see Section 6.2.19, “Proxy Users”).
- If the client user does not proxy another user, but does match a `mysql.user` entry, counting uses the `CURRENT_USER()` value corresponding to that entry. For example, if a user `user1` connecting from a host `host1.example.com` matches a `user1@host1.example.com` entry, counting uses `user1@host1.example.com`. If the user matches a `user1@%.example.com`, `user1@%.com`, or `user1@%` entry instead, counting uses `user1@%.example.com`, `user1@%.com`, or `user1@%`, respectively.

For the cases just described, the connection attempt matches some `mysql.user` entry, and whether the request succeeds or fails depends on whether the client provides the correct authentication credentials. For example, if the client presents an incorrect password, the connection attempt fails.

If the connection attempt matches no `mysql.user` entry, the attempt fails. In this case, no `CURRENT_USER()` value is available and connection-failure counting uses the user name provided by the client and the client host as determined by the server. For example, if a client attempts to connect as user `user2` from host `host2.example.com`, the user name part is available in the client request and the server determines the host information. The user/host combination used for counting is `user2@host2.example.com`.

Note

The server maintains information about which client hosts can possibly connect to the server (essentially the union of host values for `mysql.user` entries). If a client attempts to connect from any other host, the server rejects the attempt at an early stage of connection setup:

```
ERROR 1130 (HY000): Host 'host_name' is not
allowed to connect to this MySQL server
```

Because this type of rejection occurs so early, `CONNECTION_CONTROL` does not see it, and does not count it.

Connection Failure Monitoring

To monitor failed connections, use 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 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 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.

Assigning a value to `connection_control_failed_connections_threshold` at runtime has these effects:

- All accumulated failed-connection counters are reset to zero.
- The `Connection_control_delay_generated` status variable is reset to zero.
- The `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table becomes empty.

6.4.2.2 Connection-Control System and Status Variables

This section describes the system and status variables that the `CONNECTION_CONTROL` plugin provides to enable its operation to be configured and monitored.

- Connection-Control System Variables
- Connection-Control Status Variables

Connection-Control System Variables

If the `CONNECTION_CONTROL` plugin is installed, it exposes these system variables:

- `connection_control_failed_connections_threshold`

Command-Line Format	`--connection-control-failed-connections-threshold=#`
System Variable	`connection_control_failed_connections_threshol`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`3`
Minimum Value	`0`
Maximum Value	`2147483647`

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

- If the variable has a nonzero value *`N`*, the server adds a delay beginning with consecutive failed attempt *`N`*+1. If an account has reached the point where connection responses are delayed, a delay also occurs for the next subsequent successful connection.
- Setting this variable to zero disables failed-connection counting. In this case, the server never adds delays.

For information about how `connection_control_failed_connections_threshold` interacts with other connection-control system and status variables, see Section 6.4.2.1, “Connection-Control Plugin Installation”.

- `connection_control_max_connection_delay`

Command-Line Format	`--connection-control-max-connection-delay=#`
System Variable	`connection_control_max_connection_delay`
Scope	Global
Dynamic	Yes

SET_VAR Hint Applies	No
Type	Integer
Default Value	2147483647
Minimum Value	1000
Maximum Value	2147483647
Unit	milliseconds

The maximum delay in milliseconds for server response to failed connection attempts, if connection_control_failed_connections_threshold is greater than zero.

For information about how connection_control_max_connection_delay interacts with other connection-control system and status variables, see Section 6.4.2.1, “Connection-Control Plugin Installation”.

- connection_control_min_connection_delay

Command-Line Format	--connection-control-min-connection-delay=#
System Variable	connection_control_min_connection_delay
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1000
Minimum Value	1000
Maximum Value	2147483647
Unit	milliseconds

The minimum delay in milliseconds for server response to failed connection attempts, if connection_control_failed_connections_threshold is greater than zero.

For information about how connection_control_min_connection_delay interacts with other connection-control system and status variables, see Section 6.4.2.1, “Connection-Control Plugin Installation”.

Connection-Control Status Variables

If the CONNECTION_CONTROL plugin is installed, it exposes this status variable:

- Connection_control_delay_generated

  The number of times the server added a delay to its response to a failed connection attempt. This does not count attempts that occur before reaching the threshold defined by the connection_control_failed_connections_threshold system variable.

  This variable provides a simple counter. For more detailed connection-control monitoring information, examine the INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS table; see Section 26.6.2, “The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table”.

  Assigning a value to connection_control_failed_connections_threshold at runtime resets Connection_control_delay_generated to zero.

6.4.3 The Password Validation Component

The `validate_password` component serves to improve security by requiring account passwords and enabling strength testing of potential passwords. This component exposes system variables that enable you to configure password policy, and status variables for component monitoring.

Note

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. (For general information about components, see Section 5.5, “MySQL Components”.) The following instructions describe how to use the component, not the plugin. For instructions on using the plugin form of `validate_password`, see The Password Validation Plugin, in MySQL 5.7 Reference Manual.

The plugin form of `validate_password` is still available but is deprecated; expect it to be removed in a future version of MySQL. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, “Transitioning to the Password Validation Component”.

The `validate_password` component implements these capabilities:

- For SQL statements that assign a password supplied as a cleartext value, `validate_password` checks the password against the current password policy and rejects the password if it is weak (the statement returns an `ER_NOT_VALID_PASSWORD` error). This applies to the `ALTER USER`, `CREATE USER`, and `SET PASSWORD` statements.
- For `CREATE USER` statements, `validate_password` requires that a password be given, and that it satisfies the password policy. This is true even if an account is locked initially because otherwise unlocking the account later would cause it to become accessible without a password that satisfies the policy.
- `validate_password` implements a `VALIDATE_PASSWORD_STRENGTH()` SQL function that assesses the strength of potential passwords. This function takes a password argument and returns an integer from 0 (weak) to 100 (strong).

Note

For statements that assign or modify account passwords (`ALTER USER`, `CREATE USER`, and `SET PASSWORD`), the `validate_password` capabilities described here apply only to accounts that use an authentication plugin that stores credentials internally to MySQL. For accounts that use plugins that perform authentication against a credentials system external to MySQL, password management must be handled externally against that system as well. For more information about internal credentials storage, see Section 6.2.15, “Password Management”.

The preceding restriction does not apply to use of the `VALIDATE_PASSWORD_STRENGTH()` function because it does not affect accounts directly.

Examples:

- `validate_password` checks the cleartext password in the following statement. Under the default password policy, which requires passwords to be at least 8 characters long, the password is weak and the statement produces an error:

```
mysql> ALTER USER USER() IDENTIFIED BY 'abc';
ERROR 1819 (HY000): Your password does not satisfy the current
policy requirements
```

- Passwords specified as hashed values are not checked because the original password value is not available for checking:

```
mysql> ALTER USER 'jeffrey'@'localhost'
```

```
      IDENTIFIED WITH mysql_native_password
      AS '*0D3CED9BEC10A777AEC23CCC353A8C08A633045E';
Query OK, 0 rows affected (0.01 sec)
```

- This account-creation statement fails, even though the account is locked initially, because it does not include a password that satisfies the current password policy:

```
mysql> CREATE USER 'juanita'@'localhost' ACCOUNT LOCK;
ERROR 1819 (HY000): Your password does not satisfy the current
policy requirements
```

- To check a password, use the `VALIDATE_PASSWORD_STRENGTH()` function:

```
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('weak');
+----------------------------------+
| VALIDATE_PASSWORD_STRENGTH('weak') |
+----------------------------------+
|                               25 |
+----------------------------------+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('lessweak$_@123');
+-------------------------------------------+
| VALIDATE_PASSWORD_STRENGTH('lessweak$_@123') |
+-------------------------------------------+
|                                        50 |
+-------------------------------------------+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!');
+--------------------------------------------+
| VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!') |
+--------------------------------------------+
|                                       100 |
+--------------------------------------------+
```

To configure password checking, modify the system variables having names of the form `validate_password.xxx`; these are the parameters that control password policy. See Section 6.4.3.2, “Password Validation Options and Variables”.

If `validate_password` is not installed, the `validate_password.xxx` system variables are not available, passwords in statements are not checked, and the `VALIDATE_PASSWORD_STRENGTH()` function always returns 0. For example, without the plugin installed, accounts can be assigned passwords shorter than 8 characters, or no password at all.

Assuming that `validate_password` is installed, it implements three levels of password checking: `LOW`, `MEDIUM`, and `STRONG`. The default is `MEDIUM`; to change this, modify the value of `validate_password.policy`. The policies implement increasingly strict password tests. The following descriptions refer to default parameter values, which can be modified by changing the appropriate system variables.

- `LOW` policy tests password length only. Passwords must be at least 8 characters long. To change this length, modify `validate_password.length`.
- `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`.
- `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, `validate_password` supports the capability of rejecting 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.

6.4.3.1 Password Validation Component Installation and Uninstallation

This section describes how to install and uninstall the `validate_password` password-validation component. For general information about installing and uninstalling components, see Section 5.5, “MySQL Components”.

Note

If you install MySQL 8.0 using the MySQL Yum repository, MySQL SLES Repository, or RPM packages provided by Oracle, the `validate_password` component is enabled by default after you start your MySQL Server for the first time.

Upgrades to MySQL 8.0 from 5.7 using Yum or RPM packages leave the `validate_password` plugin in place. To make the transition from the `validate_password` plugin to the `validate_password` component, see Section 6.4.3.3, “Transitioning to the Password Validation Component”.

To be usable by the server, the component library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To install the `validate_password` component, use this statement:

```
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.

To uninstall the `validate_password` component, use this statement:

```
UNINSTALL COMPONENT 'file://component_validate_password';
```

`UNINSTALL COMPONENT` unloads the component, and unregisters it from the `mysql.component` system table to cause it not to be loaded during subsequent server startups.

6.4.3.2 Password Validation Options and Variables

This section describes the system and status variables that `validate_password` provides to enable its operation to be configured and monitored.

- Password Validation Component System Variables
- Password Validation Component Status Variables
- Password Validation Plugin Options
- Password Validation Plugin System Variables
- Password Validation Plugin Status Variables

Password Validation Component System Variables

If the `validate_password` component is enabled, it exposes several system variables that enable configuration of password checking:

```
mysql> SHOW VARIABLES LIKE 'validate_password.%';
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| validate_password.check_user_name    | ON     |
| validate_password.dictionary_file    |        |
| validate_password.length             | 8      |
| validate_password.mixed_case_count   | 1      |
| validate_password.number_count       | 1      |
| validate_password.policy             | MEDIUM |
| validate_password.special_char_count | 1      |
```

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

To change how passwords are checked, you can set these system variables at server startup or at runtime. The following list describes the meaning of each variable.

- `validate_password.check_user_name`

Command-Line Format	`--validate-password.check-user-name[={OFF|ON}]`
System Variable	`validate_password.check_user_name`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether `validate_password` compares passwords to the user name part of the effective user account for the current session and rejects them if they match. This variable is unavailable unless `validate_password` is installed.

By default, `validate_password.check_user_name` is enabled. This variable controls user name matching independent of the value of `validate_password.policy`.

When `validate_password.check_user_name` is enabled, it has these effects:

- Checking occurs in all contexts for which `validate_password` is invoked, which includes use of statements such as `ALTER USER` or `SET PASSWORD` to change the current user's password, and invocation of functions such as `VALIDATE_PASSWORD_STRENGTH()`.
- The user names used for comparison are taken from the values of the `USER()` and `CURRENT_USER()` functions for the current session. An implication is that a user who has sufficient privileges to set another user's password can set the password to that user's name, and cannot set that user's password to the name of the user executing the statement. For example, `'root'@'localhost'` can set the password for `'jeffrey'@'localhost'` to `'jeffrey'`, but cannot set the password to `'root'`.
- Only the user name part of the `USER()` and `CURRENT_USER()` function values is used, not the host name part. If a user name is empty, no comparison occurs.
- If a password is the same as the user name or its reverse, a match occurs and the password is rejected.
- User-name matching is case-sensitive. The password and user name values are compared as binary strings on a byte-by-byte basis.
- If a password matches the user name, `VALIDATE_PASSWORD_STRENGTH()` returns 0 regardless of how other `validate_password` system variables are set.

- `validate_password.dictionary_file`

Command-Line Format	`--validate-password.dictionary-file=file_name`
System Variable	`validate_password.dictionary_file`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No

Type	File name

The path name of the dictionary file that `validate_password` uses for checking passwords. This variable is unavailable unless `validate_password` is installed.

By default, this variable has an empty value and dictionary checks are not performed. For dictionary checks to occur, the variable value must be nonempty. If the file is named as a relative path, it is interpreted relative to the server data directory. File contents should be lowercase, one word per line. Contents are treated as having a character set of `utf8mb3`. The maximum permitted file size is 1MB.

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. Assuming that is true, each substring of the password of length 4 up to 100 is compared to the words in the dictionary file. Any match causes the password to be rejected. Comparisons are not case-sensitive.

For `VALIDATE_PASSWORD_STRENGTH()`, the password is checked against all policies, including `STRONG`, so the strength assessment includes the dictionary check regardless of the `validate_password.policy` value.

`validate_password.dictionary_file` can be set at runtime and assigning a value causes the named file to be read without a server restart.

- `validate_password.length`

Command-Line Format	`--validate-password.length=#`
System Variable	`validate_password.length`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`8`
Minimum Value	`0`

The minimum number of characters that `validate_password` requires passwords to have. This variable is unavailable unless `validate_password` is installed.

The `validate_password.length` minimum value is a function of several other related system variables. The value cannot be set less than the value of this expression:

```
validate_password.number_count
+ validate_password.special_char_count
+ (2 * validate_password.mixed_case_count)
```

If `validate_password` adjusts the value of `validate_password.length` due to the preceding constraint, it writes a message to the error log.

- `validate_password.mixed_case_count`

Command-Line Format	`--validate-password.mixed-case-count=#`
System Variable	`validate_password.mixed_case_count`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer

Default Value	1
Minimum Value	0

The minimum number of lowercase and uppercase characters that validate_password requires passwords to have if the password policy is MEDIUM or stronger. This variable is unavailable unless validate_password is installed.

For a given validate_password.mixed_case_count value, the password must have that many lowercase characters, and that many uppercase characters.

- validate_password.number_count

Command-Line Format	--validate-password.number-count=#
System Variable	validate_password.number_count
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Minimum Value	0

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

- validate_password.policy

Command-Line Format	--validate-password.policy=value
System Variable	validate_password.policy
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	1
Valid Values	0 1 2

The password policy enforced by validate_password. This variable is unavailable unless validate_password is installed.

validate_password.policy affects how validate_password uses its other policy-setting system variables, except for checking passwords against user names, which is controlled independently by validate_password.check_user_name.

The validate_password.policy value can be specified using numeric values 0, 1, 2, or the corresponding symbolic values LOW, MEDIUM, STRONG. The following table describes the tests performed for each policy. For the length test, the required length is the value of the

`validate_password.length` system variable. Similarly, the required values for the other tests are given by other `validate_password.xxx` variables.

Policy	Tests Performed
`0` or `LOW`	Length
`1` or `MEDIUM`	Length; numeric, lowercase/uppercase, and special characters
`2` or `STRONG`	Length; numeric, lowercase/uppercase, and special characters; dictionary file

- `validate_password.special_char_count`

Command-Line Format	`--validate-password.special-char-count=#`
System Variable	`validate_password.special_char_count`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`0`

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

Password Validation Component Status Variables

If the `validate_password` component is enabled, it exposes status variables that provide operational information:

```
mysql> SHOW STATUS LIKE 'validate_password.%';
+-----------------------------------------------+---------------------+
| Variable_name                                 | Value               |
+-----------------------------------------------+---------------------+
| validate_password.dictionary_file_last_parsed | 2019-10-03 08:33:49 |
| validate_password.dictionary_file_words_count | 1902                |
+-----------------------------------------------+---------------------+
```

The following list describes the meaning of each status variable.

- `validate_password.dictionary_file_last_parsed`

  When the dictionary file was last parsed. This variable is unavailable unless `validate_password` is installed.

- `validate_password.dictionary_file_words_count`

  The number of words read from the dictionary file. This variable is unavailable unless `validate_password` is installed.

Password Validation Plugin Options

Note

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. The `validate_password`

> plugin is deprecated; expect it to be removed in a future version of MySQL. Consequently, its options are also deprecated, and you should expect them to be removed as well. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, “Transitioning to the Password Validation Component”.

To control activation of the `validate_password` plugin, use this option:

- `--validate-password[=value]`

Command-Line Format	`--validate-password[=value]`
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`
	`FORCE_PLUS_PERMANENT`

This option controls how the server loads the deprecated `validate_password` plugin at startup. The value should be one of those available for plugin-loading options, as described in Section 5.6.1, “Installing and Uninstalling Plugins”. For example, `--validate-password=FORCE_PLUS_PERMANENT` tells the server to load the plugin at startup and prevents it from being removed while the server is running.

This option is available only if the `validate_password` plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load-add`. See Section 6.4.3.1, “Password Validation Component Installation and Uninstallation”.

Password Validation Plugin System Variables

> **Note**
>
> In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. The `validate_password` plugin is deprecated; expect it to be removed in a future version of MySQL. Consequently, its system variables are also deprecated and you should expect them to be removed as well. Use the corresponding system variables of the `validate_password` component instead; see Password Validation Component System Variables. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, “Transitioning to the Password Validation Component”.

- `validate_password_check_user_name`

Command-Line Format	`--validate-password-check-user-name[={OFF|ON}]`
System Variable	`validate_password_check_user_name`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`

This validate_password plugin system variable is deprecated; expect it to be removed in a future version of MySQL. Use the corresponding validate_password.check_user_name system variable of the validate_password component instead.

- validate_password_dictionary_file

Command-Line Format	--validate-password-dictionary-file=file_name
System Variable	validate_password_dictionary_file
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name

This validate_password plugin system variable is deprecated; expect it to be removed in a future version of MySQL. Use the corresponding validate_password.dictionary_file system variable of the validate_password component instead.

- validate_password_length

Command-Line Format	--validate-password-length=#
System Variable	validate_password_length
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	8
Minimum Value	0

This validate_password plugin system variable is deprecated; expect it to be removed in a future version of MySQL. Use the corresponding validate_password.length system variable of the validate_password component instead.

- validate_password_mixed_case_count

Command-Line Format	--validate-password-mixed-case-count=#
System Variable	validate_password_mixed_case_count
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Minimum Value	0

This validate_password plugin system variable is deprecated; expect it to be removed in a future version of MySQL. Use the corresponding validate_password.mixed_case_count system variable of the validate_password component instead.

- `validate_password_number_count`

Command-Line Format	`--validate-password-number-count=#`
System Variable	`validate_password_number_count`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`0`

This `validate_password` plugin system variable is deprecated; expect it to be removed in a future version of MySQL. Use the corresponding `validate_password.number_count` system variable of the `validate_password` component instead.

- `validate_password_policy`

Command-Line Format	`--validate-password-policy=value`
System Variable	`validate_password_policy`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`1`

Valid Values	`0`
	`1`
	`2`

This `validate_password` plugin system variable is deprecated; expect it to be removed in a future version of MySQL. Use the corresponding `validate_password.policy` system variable of the `validate_password` component instead.

- `validate_password_special_char_count`

Command-Line Format	`--validate-password-special-char-count=#`
System Variable	`validate_password_special_char_count`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`0`

This `validate_password` plugin system variable is deprecated; expect it to be removed in a future version of MySQL. Use the corresponding `validate_password.special_char_count` system variable of the `validate_password` component instead.

Password Validation Plugin Status Variables

Note

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. The `validate_password` plugin is deprecated; expect it to be removed in a future version of MySQL. Consequently, its status variables are also deprecated; expect it to be removed. Use the corresponding status variables of the `validate_password` component; see Password Validation Component Status Variables. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, "Transitioning to the Password Validation Component".

- `validate_password_dictionary_file_last_parsed`

  This `validate_password` plugin status variable is deprecated; expect it to be removed in a future version of MySQL. Use the corresponding `validate_password.dictionary_file_last_parsed` status variable of the `validate_password` component instead.

- `validate_password_dictionary_file_words_count`

  This `validate_password` plugin status variable is deprecated; expect it to be removed in a future version of MySQL. Use the corresponding `validate_password.dictionary_file_words_count` status variable of the `validate_password` component instead.

6.4.3.3 Transitioning to the Password Validation Component

Note

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. The `validate_password` plugin is deprecated; expect it to be removed in a future version of MySQL.

MySQL installations that currently use the `validate_password` plugin should make the transition to using the `validate_password` component instead. To do so, use the following procedure. The procedure installs the component before uninstalling the plugin, to avoid having a time window during which no password validation occurs. (The component and plugin can be installed simultaneously. In this case, the server attempts to use the component, falling back to the plugin if the component is unavailable.)

1. Install the `validate_password` component:

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

2. Test the `validate_password` component to ensure that it works as expected. If you need to set any `validate_password.xxx` system variables, you can do so at runtime using `SET GLOBAL`. (Any option file changes that must be made are performed in the next step.)

3. Adjust any references to the plugin system and status variables to refer to the corresponding component system and status variables. Suppose that previously you had configured the plugin at startup using an option file like this:

   ```
   [mysqld]
   validate-password=FORCE_PLUS_PERMANENT
   validate_password_dictionary_file=/usr/share/dict/words
   validate_password_length=10
   validate_password_number_count=2
   ```

   Those settings are appropriate for the plugin, but must be modified to apply to the component. To adjust the option file, omit the `--validate-password` option (it applies only to the plugin, not

the component), and modify the system variable references from no-dot names appropriate for the plugin to dotted names appropriate for the component:

```
[mysqld]
validate_password.dictionary_file=/usr/share/dict/words
validate_password.length=10
validate_password.number_count=2
```

Similar adjustments are needed for applications that refer at runtime to `validate_password` plugin system and status variables. Change the no-dot plugin variable names to the corresponding dotted component variable names.

4. Uninstall the `validate_password` plugin:

```
UNINSTALL PLUGIN validate_password;
```

If the `validate_password` plugin is loaded at server startup using a `--plugin-load` or `--plugin-load-add` option, omit that option from the server startup procedure. For example, if the option is listed in a server option file, remove it from the file.

5. Restart the server.

6.4.4 The MySQL Keyring

MySQL Server supports a keyring that enables internal server components and plugins to securely store sensitive information for later retrieval. The implementation comprises these elements:

- Keyring components and plugins that manage a backing store or communicate with a storage back end. Keyring use involves installing one from among the available components and plugins. Keyring components and plugins both manage keyring data but are configured differently and may have operational differences (see Section 6.4.4.1, “Keyring Components Versus Keyring Plugins”).

  These keyring components are available:

  - `component_keyring_file`: Stores keyring data in a file local to the server host. Available in MySQL Community Edition and MySQL Enterprise Edition distributions as of MySQL 8.0.24. See Section 6.4.4.4, “Using the component_keyring_file File-Based Keyring Component”.
  - `component_keyring_encrypted_file`: Stores keyring data in an encrypted, password-protected file local to the server host. Available in MySQL Enterprise Edition distributions as of MySQL 8.0.24. See Section 6.4.4.5, “Using the component_keyring_encrypted_file Encrypted File-Based Keyring Component”.
  - `component_keyring_oci`: Stores keyring data in the Oracle Cloud Infrastructure Vault. Available in MySQL Enterprise Edition distributions as of MySQL 8.0.31. See Section 6.4.4.11, “Using the Oracle Cloud Infrastructure Vault Keyring Component”.

  These keyring plugins are available:

  - `keyring_file`: Stores keyring data in a file local to the server host. Available in MySQL Community Edition and MySQL Enterprise Edition distributions. See Section 6.4.4.6, “Using the keyring_file File-Based Keyring Plugin”.
  - `keyring_encrypted_file`: Stores keyring data in an encrypted, password-protected file local to the server host. Available in MySQL Enterprise Edition distributions. See Section 6.4.4.7, “Using the keyring_encrypted_file Encrypted File-Based Keyring Plugin”.
  - `keyring_okv`: A KMIP 1.1 plugin for use with KMIP-compatible back end keyring storage products such as Oracle Key Vault and Gemalto SafeNet KeySecure Appliance. Available in MySQL Enterprise Edition distributions. See Section 6.4.4.8, “Using the keyring_okv KMIP Plugin”.

  - `keyring_aws`: Communicates with the Amazon Web Services Key Management Service for key generation and uses a local file for key storage. Available in MySQL Enterprise Edition distributions. See Section 6.4.4.9, “Using the keyring_aws Amazon Web Services Keyring Plugin”.
  - `keyring_hashicorp`: Communicates with HashiCorp Vault for back end storage. Available in MySQL Enterprise Edition distributions as of MySQL 8.0.18. See Section 6.4.4.10, “Using the HashiCorp Vault Keyring Plugin”.
  - `keyring_oci`: Communicates with Oracle Cloud Infrastructure Vault for back end storage. Available in MySQL Enterprise Edition distributions as of MySQL 8.0.22. See Section 6.4.4.12, “Using the Oracle Cloud Infrastructure Vault Keyring Plugin”.
- A keyring service interface for keyring key management. This service is accessible at two levels:
  - SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.
  - C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, “The Keyring Service”.
- Key metadata access:
  - The Performance Schema `keyring_keys` table exposes metadata for keys in the keyring. Key metadata includes key IDs, key owners, and backend key IDs. The `keyring_keys` table does not expose any sensitive keyring data such as key contents. Available as of MySQL 8.0.16. See Section 27.12.18.2, “The keyring_keys table”.
  - The Performance Schema `keyring_component_status` table provides status information about the keyring component in use, if one is installed. Available as of MySQL 8.0.24. See Section 27.12.18.1, “The keyring_component_status Table”.
- A key migration capability. MySQL supports migration of keys between keystores, enabling DBAs to switch a MySQL installation from one keystore to another. See Section 6.4.4.14, “Migrating Keys Between Keyring Keystores”.
- The implementation of keyring plugins is revised as of MySQL 8.0.24 to use the component infrastructure. This is facilitated using the built-in plugin named `daemon_keyring_proxy_plugin` that acts as a bridge between the plugin and component service APIs. See Section 5.6.8, “The Keyring Proxy Bridge Plugin”.

Warning

For encryption key management, the `component_keyring_file` and `component_keyring_encrypted_file` components, and the `keyring_file` and `keyring_encrypted_file` plugins are not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

Within MySQL, keyring service consumers include:

- The `InnoDB` storage engine uses the keyring to store its key for tablespace encryption. See Section 15.13, “InnoDB Data-at-Rest Encryption”.
- MySQL Enterprise Audit uses the keyring to store the audit log file encryption password. See Encrypting Audit Log Files.
- Binary log and relay log management supports keyring-based encryption of log files. With log file encryption activated, the keyring stores the keys used to encrypt passwords for the binary log files and relay log files. See Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.

- The master key to decrypt the file key that decrypts the persisted values of sensitive system variables is stored in the keyring. A keyring component must be enabled on the MySQL Server instance to support secure storage for persisted system variable values, rather than a keyring plugin, which do not support the function. See Persisting Sensitive System Variables.

For general keyring installation instructions, see Section 6.4.4.2, “Keyring Component Installation”, and Section 6.4.4.3, “Keyring Plugin Installation”. For installation and configuration information specific to a given keyring component or plugin, see the section describing it.

For information about using the keyring functions, see Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.

Keyring components, plugins, and functions access a keyring service that provides the interface to the keyring. For information about accessing this service and writing keyring plugins, see Section 5.6.9.2, “The Keyring Service”, and Writing Keyring Plugins.

6.4.4.1 Keyring Components Versus Keyring Plugins

The MySQL Keyring originally implemented keystore capabilities using server plugins, but began transitioning to use the component infrastructure in MySQL 8.0.24. This section briefly compares keyring components and plugins to provide an overview of their differences. It may assist you in making the transition from plugins to components, or, if you are just beginning to use the keyring, assist you in choosing whether to use a component versus using a plugin.

- Keyring plugin loading uses the `--early-plugin-load` option. Keyring component loading uses a manifest.
- Keyring plugin configuration is based on plugin-specific system variables. For keyring components, no system variables are used. Instead, each component has its own configuration file.
- Keyring components have fewer restrictions than keyring plugins with respect to key types and lengths. See Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

Note

`component_keyring_oci` (like the `keyring_oci` plugin) can only generate keys of type AES with a size of 16, 24, or 32 bytes.

- Keyring components support secure storage for persisted system variable values, whereas keyring plugins do not support the function.

  A keyring component must be enabled on the MySQL server instance to support secure storage for persisted system variable values. The sensitive data that can be protected in this way includes items such as private keys and passwords that appear in the values of system variables. In the operating system file where persisted system variables are stored, the names and values of sensitive system variables are stored in an encrypted format, along with a generated file key to decrypt them. The generated file key is in turn encrypted using a master key that is stored in a keyring. See Persisting Sensitive System Variables.

6.4.4.2 Keyring Component Installation

Keyring service consumers require that a keyring component or plugin be installed:

- To use a keyring component, begin with the instructions here.
- To use a keyring plugin instead, begin with Section 6.4.4.3, “Keyring Plugin Installation”.
- If you intend to use keyring functions in conjunction with the chosen keyring component or plugin, install the functions after installing that component or plugin, using the instructions in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.

Note

Only one keyring component or plugin should be enabled at a time. Enabling multiple keyring components or plugins is unsupported and results may not be as anticipated.

MySQL provides these keyring component choices:

- `component_keyring_file`: Stores keyring data in a file local to the server host. Available in MySQL Community Edition and MySQL Enterprise Edition distributions.
- `component_keyring_encrypted_file`: Stores keyring data in an encrypted, password-protected file local to the server host. Available in MySQL Enterprise Edition distributions.
- `component_keyring_oci`: Stores keyring data in the Oracle Cloud Infrastructure Vault. Available in MySQL Enterprise Edition distributions.

To be usable by the server, the component library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

A keyring component or plugin must be loaded early during the server startup sequence so that other components can access it as necessary during their own initialization. For example, the `InnoDB` storage engine uses the keyring for tablespace encryption, so a keyring component or plugin must be loaded and available prior to `InnoDB` initialization.

Note

A keyring component must be enabled on the MySQL server instance if you need to support secure storage for persisted system variable values. The keyring plugin does not support the function. See Persisting Sensitive System Variables.

Unlike keyring plugins, keyring components are not loaded using the `--early-plugin-load` server option or configured using system variables. Instead, the server determines which keyring component to load during startup using a manifest, and the loaded component consults its own configuration file when it initializes. Therefore, to install a keyring component, you must:

1. Write a manifest that tells the server which keyring component to load.
2. Write a configuration file for that keyring component.

The first step in installing a keyring component is writing a manifest that indicates which component to load. During startup, the server reads either a global manifest file, or a global manifest file paired with a local manifest file:

- The server attempts to read its global manifest file from the directory where the server is installed.
- If the global manifest file indicates use of a local manifest file, the server attempts to read its local manifest file from the data directory.
- Although global and local manifest files are located in different directories, the file name is `mysqld.my` in both locations.
- It is not an error for a manifest file not to exist. In this case, the server attempts no component loading associated with the file.

Local manifest files permit setting up component loading for multiple instances of the server, such that loading instructions for each server instance are specific to a given data directory instance. This enables different MySQL instances to use different keyring components.

Server manifest files have these properties:

- A manifest file must be in valid JSON format.
- A manifest file permits these items:
  - `"read_local_manifest"`: This item is permitted only in the global manifest file. If the item is not present, the server uses only the global manifest file. If the item is present, its value is `true` or `false`, indicating whether the server should read component-loading information from the local manifest file.

    If the `"read_local_manifest"` item is present in the global manifest file along with other items, the server checks the `"read_local_manifest"` item value first:

    - If the value is `false`, the server processes the other items in the global manifest file and ignores the local manifest file.
    - If the value is `true`, the server ignores the other items in the global manifest file and attempts to read the local manifest file.
  - `"components"`: This item indicates which component to load. The item value is a string that specifies a valid component URN, such as `"file://component_keyring_file"`. A component URN begins with `file://` and indicates the base name of the library file located in the MySQL plugin directory that implements the component.
- Server access to a manifest file should be read only. For example, a `mysqld.my` server manifest file may be owned by `root` and be read/write to `root`, but should be read only to the account used to run the MySQL server. If the manifest file is found during startup to be read/write to that account, the server writes a warning to the error log suggesting that the file be made read only.
- The database administrator has the responsibility for creating any manifest files to be used, and for ensuring that their access mode and contents are correct. If an error occurs, server startup fails and the administrator must correct any issues indicated by diagnostics in the server error log.

Given the preceding manifest file properties, to configure the server to load `component_keyring_file`, create a global manifest file named `mysqld.my` in the `mysqld` installation directory, and optionally create a local manifest file, also named `mysqld.my`, in the data directory. The following instructions describe how to load `component_keyring_file`. To load a different keyring component, substitute its name for `component_keyring_file`.

- To use a global manifest file only, the file contents look like this:

  ```
  {
    "components": "file://component_keyring_file"
  }
  ```

  Create this file in the directory where `mysqld` is installed.
- Alternatively, to use a global and local manifest file pair, the global file looks like this:

  ```
  {
    "read_local_manifest": true
  }
  ```

  Create this file in the directory where `mysqld` is installed.

  The local file looks like this:

  ```
  {
    "components": "file://component_keyring_file"
  }
  ```

  Create this file in the data directory.

With the manifest in place, proceed to configuring the keyring component. To do this, check the notes for your chosen keyring component for configuration instructions specific to that component:

- `component_keyring_file`: Section 6.4.4.4, “Using the component_keyring_file File-Based Keyring Component”.
- `component_keyring_encrypted_file`: Section 6.4.4.5, “Using the component_keyring_encrypted_file Encrypted File-Based Keyring Component”.
- `component_keyring_oci`: Section 6.4.4.11, “Using the Oracle Cloud Infrastructure Vault Keyring Component”.

After performing any component-specific configuration, start the server. Verify component installation by examining the Performance Schema `keyring_component_status` table:

```
mysql> SELECT * FROM performance_schema.keyring_component_status;
+--------------------+-------------------------------------------------+
| STATUS_KEY         | STATUS_VALUE                                    |
+--------------------+-------------------------------------------------+
| Component_name     | component_keyring_file                          |
| Author             | Oracle Corporation                              |
| License            | GPL                                             |
| Implementation_name | component_keyring_file                         |
| Version            | 1.0                                             |
| Component_status   | Active                                          |
| Data_file          | /usr/local/mysql/keyring/component_keyring_file |
| Read_only          | No                                              |
+--------------------+-------------------------------------------------+
```

A `Component_status` value of `Active` indicates that the component initialized successfully.

If the component cannot be loaded, server startup fails. Check the server error log for diagnostic messages. If the component loads but fails to initialize due to configuration problems, the server starts but the `Component_status` value is `Disabled`. Check the server error log, correct the configuration issues, and use the `ALTER INSTANCE RELOAD KEYRING` statement to reload the configuration.

Keyring components should be loaded only by using a manifest file, not by using the `INSTALL COMPONENT` statement. Keyring components loaded using that statement may be available too late in the server startup sequence for certain components that use the keyring, such as `InnoDB`, because they are registered in the `mysql.component` system table and loaded automatically for subsequent server restarts. But `mysql.component` is an `InnoDB` table, so any components named in it can be loaded during startup only after `InnoDB` initialization.

If no keyring component or plugin is available when a component tries to access the keyring service, the service cannot be used by that component. As a result, the component may fail to initialize or may initialize with limited functionality. For example, if `InnoDB` finds that there are encrypted tablespaces when it initializes, it attempts to access the keyring. If the keyring is unavailable, `InnoDB` can access only unencrypted tablespaces.

6.4.4.3 Keyring Plugin Installation

Keyring service consumers require that a keyring component or plugin be installed:

- To use a keyring plugin, begin with the instructions here. (Also, for general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.)
- To use a keyring component instead, begin with Section 6.4.4.2, “Keyring Component Installation”.
- If you intend to use keyring functions in conjunction with the chosen keyring component or plugin, install the functions after installing that component or plugin, using the instructions in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.

Note

Only one keyring component or plugin should be enabled at a time. Enabling multiple keyring components or plugins is unsupported and results may not be as anticipated.

> A keyring component must be enabled on the MySQL Server instance if you need to support secure storage for persisted system variable values, rather than a keyring plugin, which do not support the function. See Persisting Sensitive System Variables.

MySQL provides these keyring plugin choices:

- `keyring_file`: Stores keyring data in a file local to the server host. Available in MySQL Community Edition and MySQL Enterprise Edition distributions.
- `keyring_encrypted_file`: Stores keyring data in an encrypted, password-protected file local to the server host. Available in MySQL Enterprise Edition distributions.
- `keyring_okv`: A KMIP 1.1 plugin for use with KMIP-compatible back end keyring storage products such as Oracle Key Vault and Gemalto SafeNet KeySecure Appliance. Available in MySQL Enterprise Edition distributions.
- `keyring_aws`: Communicates with the Amazon Web Services Key Management Service as a back end for key generation and uses a local file for key storage. Available in MySQL Enterprise Edition distributions.
- `keyring_hashicorp`: Communicates with HashiCorp Vault for back end storage. Available in MySQL Enterprise Edition distributions.
- `keyring_oci`: Communicates with Oracle Cloud Infrastructure Vault for back end storage. See Section 6.4.4.12, “Using the Oracle Cloud Infrastructure Vault Keyring Plugin”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

A keyring component or plugin must be loaded early during the server startup sequence so that other components can access it as necessary during their own initialization. For example, the `InnoDB` storage engine uses the keyring for tablespace encryption, so a keyring component or plugin must be loaded and available prior to `InnoDB` initialization.

Installation for each keyring plugin is similar. The following instructions describe how to install `keyring_file`. To use a different keyring plugin, substitute its name for `keyring_file`.

The `keyring_file` plugin library file base name is `keyring_file`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin, use the `--early-plugin-load` option to name the plugin library file that contains it. For example, on platforms where the plugin library file suffix is `.so`, use these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
early-plugin-load=keyring_file.so
```

Before starting the server, check the notes for your chosen keyring plugin for configuration instructions specific to that plugin:

- `keyring_file`: Section 6.4.4.6, “Using the keyring_file File-Based Keyring Plugin”.
- `keyring_encrypted_file`: Section 6.4.4.7, “Using the keyring_encrypted_file Encrypted File-Based Keyring Plugin”.
- `keyring_okv`: Section 6.4.4.8, “Using the keyring_okv KMIP Plugin”.
- `keyring_aws`: Section 6.4.4.9, “Using the keyring_aws Amazon Web Services Keyring Plugin”
- `keyring_hashicorp`: Section 6.4.4.10, “Using the HashiCorp Vault Keyring Plugin”

- `keyring_oci`: Section 6.4.4.12, “Using the Oracle Cloud Infrastructure Vault Keyring Plugin”

After performing any plugin-specific configuration, start the server. Verify plugin installation by examining the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

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

If the plugin fails to initialize, check the server error log for diagnostic messages.

Plugins can be loaded by methods other than `--early-plugin-load`, such as the `--plugin-load` or `--plugin-load-add` option or the `INSTALL PLUGIN` statement. However, keyring plugins loaded using those methods may be available too late in the server startup sequence for certain components that use the keyring, such as `InnoDB`:

- Plugin loading using `--plugin-load` or `--plugin-load-add` occurs after `InnoDB` initialization.
- Plugins installed using `INSTALL PLUGIN` are registered in the `mysql.plugin` system table and loaded automatically for subsequent server restarts. However, because `mysql.plugin` is an `InnoDB` table, any plugins named in it can be loaded during startup only after `InnoDB` initialization.

If no keyring component or plugin is available when a component tries to access the keyring service, the service cannot be used by that component. As a result, the component may fail to initialize or may initialize with limited functionality. For example, if `InnoDB` finds that there are encrypted tablespaces when it initializes, it attempts to access the keyring. If the keyring is unavailable, `InnoDB` can access only unencrypted tablespaces. To ensure that `InnoDB` can access encrypted tablespaces as well, use `--early-plugin-load` to load the keyring plugin.

6.4.4.4 Using the component_keyring_file File-Based Keyring Component

The `component_keyring_file` keyring component stores keyring data in a file local to the server host.

Warning

For encryption key management, the `component_keyring_file` and `component_keyring_encrypted_file` components, and the `keyring_file` and `keyring_encrypted_file` plugins are not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

To use `component_keyring_file` for keystore management, you must:

1. Write a manifest that tells the server to load `component_keyring_file`, as described in Section 6.4.4.2, “Keyring Component Installation”.
2. Write a configuration file for `component_keyring_file`, as described here.

When it initializes, `component_keyring_file` reads either a global configuration file, or a global configuration file paired with a local configuration file:

- The component attempts to read its global configuration file from the directory where the component library file is installed (that is, the server plugin directory).
- If the global configuration file indicates use of a local configuration file, the component attempts to read its local configuration file from the data directory.

- Although global and local configuration files are located in different directories, the file name is `component_keyring_file.cnf` in both locations.
- It is an error for no configuration file to exist. `component_keyring_file` cannot initialize without a valid configuration.

Local configuration files permit setting up multiple server instances to use `component_keyring_file`, such that component configuration for each server instance is specific to a given data directory instance. This enables the same keyring component to be used with a distinct data file for each instance.

`component_keyring_file` configuration files have these properties:

- A configuration file must be in valid JSON format.
- A configuration file permits these configuration items:
  - `"read_local_config"`: This item is permitted only in the global configuration file. If the item is not present, the component uses only the global configuration file. If the item is present, its value is `true` or `false`, indicating whether the component should read configuration information from the local configuration file.

    If the `"read_local_config"` item is present in the global configuration file along with other items, the component checks the `"read_local_config"` item value first:
    - If the value is `false`, the component processes the other items in the global configuration file and ignores the local configuration file.
    - If the value is `true`, the component ignores the other items in the global configuration file and attempts to read the local configuration file.
  - `"path"`: The item value is a string that names the file to use for storing keyring data. The file should be named using an absolute path, not a relative path. This item is mandatory in the configuration. If not specified, `component_keyring_file` initialization fails.
  - `"read_only"`: The item value indicates whether the keyring data file is read only. The item value is `true` (read only) or `false` (read/write). This item is mandatory in the configuration. If not specified, `component_keyring_file` initialization fails.
- The database administrator has the responsibility for creating any configuration files to be used, and for ensuring that their contents are correct. If an error occurs, server startup fails and the administrator must correct any issues indicated by diagnostics in the server error log.

Given the preceding configuration file properties, to configure `component_keyring_file`, create a global configuration file named `component_keyring_file.cnf` in the directory where the `component_keyring_file` library file is installed, and optionally create a local configuration file, also named `component_keyring_file.cnf`, in the data directory. The following instructions assume that a keyring data file named `/usr/local/mysql/keyring/component_keyring_file` is to be used in read/write fashion.

- To use a global configuration file only, the file contents look like this:

```
{
  "path": "/usr/local/mysql/keyring/component_keyring_file",
  "read_only": false
}
```

  Create this file in the directory where the `component_keyring_file` library file is installed.
- Alternatively, to use a global and local configuration file pair, the global file looks like this:

```
{
  "read_local_config": true
```

```
}
```

Create this file in the directory where the `component_keyring_file` library file is installed.

The local file looks like this:

```
{
  "path": "/usr/local/mysql/keyring/component_keyring_file",
  "read_only": false
}
```

Create this file in the data directory.

Keyring operations are transactional: `component_keyring_file` uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the data file with a suffix of `.backup`.

`component_keyring_file` supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, “The Keyring Service”.

Example (using the SQL interface):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the characteristics of key values permitted by `component_keyring_file`, see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

6.4.4.5 Using the component_keyring_encrypted_file Encrypted File-Based Keyring Component

Note

`component_keyring_encrypted_file` is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The `component_keyring_encrypted_file` keyring component stores keyring data in an encrypted, password-protected file local to the server host.

Warning

For encryption key management, the `component_keyring_file` and `component_keyring_encrypted_file` components, and the `keyring_file` and `keyring_encrypted_file` plugins are not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

To use `component_keyring_encrypted_file` for keystore management, you must:

1. Write a manifest that tells the server to load `component_keyring_encrypted_file`, as described in Section 6.4.4.2, “Keyring Component Installation”.
2. Write a configuration file for `component_keyring_encrypted_file`, as described here.

When it initializes, `component_keyring_encrypted_file` reads either a global configuration file, or a global configuration file paired with a local configuration file:

- The component attempts to read its global configuration file from the directory where the component library file is installed (that is, the server plugin directory).
- If the global configuration file indicates use of a local configuration file, the component attempts to read its local configuration file from the data directory.
- Although global and local configuration files are located in different directories, the file name is `component_keyring_encrypted_file.cnf` in both locations.
- It is an error for no Preconfiguration file to exist. `component_keyring_encrypted_file` cannot initialize without a valid configuration.

Local configuration files permit setting up multiple server instances to use `component_keyring_encrypted_file`, such that component configuration for each server instance is specific to a given data directory instance. This enables the same keyring component to be used with a distinct data file for each instance.

`component_keyring_encrypted_file` configuration files have these properties:

- A configuration file must be in valid JSON format.
- A configuration file permits these configuration items:
  - `"read_local_config"`: This item is permitted only in the global configuration file. If the item is not present, the component uses only the global configuration file. If the item is present, its value is `true` or `false`, indicating whether the component should read configuration information from the local configuration file.

    If the `"read_local_config"` item is present in the global configuration file along with other items, the component checks the `"read_local_config"` item value first:

    - If the value is `false`, the component processes the other items in the global configuration file and ignores the local configuration file.
    - If the value is `true`, the component ignores the other items in the global configuration file and attempts to read the local configuration file.
  - `"path"`: The item value is a string that names the file to use for storing keyring data. The file should be named using an absolute path, not a relative path. This item is mandatory in the configuration. If not specified, `component_keyring_encrypted_file` initialization fails.
  - `"password"`: The item value is a string that specifies the password for accessing the data file. This item is mandatory in the configuration. If not specified, `component_keyring_encrypted_file` initialization fails.
  - `"read_only"`: The item value indicates whether the keyring data file is read only. The item value is `true` (read only) or `false` (read/write). This item is mandatory in the configuration. If not specified, `component_keyring_encrypted_file` initialization fails.
- The database administrator has the responsibility for creating any configuration files to be used, and for ensuring that their contents are correct. If an error occurs, server startup fails and the administrator must correct any issues indicated by diagnostics in the server error log.
- Any configuration file that stores a password should have a restrictive mode and be accessible only to the account used to run the MySQL server.

Given the preceding configuration file properties, to configure `component_keyring_encrypted_file`, create a global configuration file named `component_keyring_encrypted_file.cnf` in the directory where the `component_keyring_encrypted_file` library file is installed, and optionally create a local configuration file, also named `component_keyring_encrypted_file.cnf`, in the data directory. The following instructions assume that a keyring data file named `/usr/local/mysql/keyring/`

component_keyring_encrypted_file is to be used in read/write fashion. You must also choose a password.

- To use a global configuration file only, the file contents look like this:

```
{
  "path": "/usr/local/mysql/keyring/component_keyring_encrypted_file",
  "password": "password",
  "read_only": false
}
```

  Create this file in the directory where the component_keyring_encrypted_file library file is installed.

- Alternatively, to use a global and local configuration file pair, the global file looks like this:

```
{
  "read_local_config": true
}
```

  Create this file in the directory where the component_keyring_encrypted_file library file is installed.

  The local file looks like this:

```
{
  "path": "/usr/local/mysql/keyring/component_keyring_encrypted_file",
  "password": "password",
  "read_only": false
}
```

  Create this file in the data directory.

Keyring operations are transactional: component_keyring_encrypted_file uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the data file with a suffix of .backup.

component_keyring_encrypted_file supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, “The Keyring Service”.

Example (using the SQL interface):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the characteristics of key values permitted by component_keyring_encrypted_file, see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

6.4.4.6 Using the keyring_file File-Based Keyring Plugin

The keyring_file keyring plugin stores keyring data in a file local to the server host.

Warning

For encryption key management, the keyring_file plugin is not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

To install `keyring_file`, use the general instructions found in Section 6.4.4.3, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_file` found here.

To be usable during the server startup process, `keyring_file` must be loaded using the `--early-plugin-load` option. The `keyring_file_data` system variable optionally configures the location of the file used by the `keyring_file` plugin for data storage. The default value is platform specific. To configure the file location explicitly, set the variable value at startup. For example, use these lines in the server `my.cnf` file, adjusting the `.so` suffix and file location for your platform as necessary:

```
[mysqld]
early-plugin-load=keyring_file.so
keyring_file_data=/usr/local/mysql/mysql-keyring/keyring
```

Keyring operations are transactional: The `keyring_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_file_data` system variable with a suffix of `.backup`.

For additional information about `keyring_file_data`, see Section 6.4.4.19, “Keyring System Variables”.

To ensure that keys are flushed only when the correct keyring storage file exists, `keyring_file` stores a SHA-256 checksum of the keyring in the file. Before updating the file, the plugin verifies that it contains the expected checksum.

The `keyring_file` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, “The Keyring Service”.

Example (using the SQL interface):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the characteristics of key values permitted by `keyring_file`, see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

6.4.4.7 Using the keyring_encrypted_file Encrypted File-Based Keyring Plugin

Note

The `keyring_encrypted_file` plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The `keyring_encrypted_file` keyring plugin stores keyring data in an encrypted, password-protected file local to the server host.

Warning

For encryption key management, the `keyring_encrypted_file` plugin is not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

To install `keyring_encrypted_file`, use the general instructions found in Section 6.4.4.3, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_encrypted_file` found here.

To be usable during the server startup process, `keyring_encrypted_file` must be loaded using the `--early-plugin-load` option. To specify the password for encrypting the keyring data file, set the `keyring_encrypted_file_password` system variable. (The password is mandatory; if not specified at server startup, `keyring_encrypted_file` initialization fails.) The `keyring_encrypted_file_data` system variable optionally configures the location of the file used by the `keyring_encrypted_file` plugin for data storage. The default value is platform specific. To configure the file location explicitly, set the variable value at startup. For example, use these lines in the server `my.cnf` file, adjusting the `.so` suffix and file location for your platform as necessary and substituting your chosen password:

```
[mysqld]
early-plugin-load=keyring_encrypted_file.so
keyring_encrypted_file_data=/usr/local/mysql/mysql-keyring/keyring-encrypted
keyring_encrypted_file_password=password
```

Because the `my.cnf` file stores a password when written as shown, it should have a restrictive mode and be accessible only to the account used to run the MySQL server.

Keyring operations are transactional: The `keyring_encrypted_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_encrypted_file_data` system variable with a suffix of `.backup`.

For additional information about the system variables used to configure the `keyring_encrypted_file` plugin, see Section 6.4.4.19, “Keyring System Variables”.

To ensure that keys are flushed only when the correct keyring storage file exists, `keyring_encrypted_file` stores a SHA-256 checksum of the keyring in the file. Before updating the file, the plugin verifies that it contains the expected checksum. In addition, `keyring_encrypted_file` encrypts file contents using AES before writing the file, and decrypts file contents after reading the file.

The `keyring_encrypted_file` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, “The Keyring Service”.

Example (using the SQL interface):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the characteristics of key values permitted by `keyring_encrypted_file`, see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

6.4.4.8 Using the keyring_okv KMIP Plugin

Note

The `keyring_okv` plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The Key Management Interoperability Protocol (KMIP) enables communication of cryptographic keys between a key management server and its clients. The `keyring_okv` keyring plugin uses the KMIP 1.1 protocol to communicate securely as a client of a KMIP back end. Keyring material is generated exclusively by the back end, not by `keyring_okv`. The plugin works with these KMIP-compatible products:

- Oracle Key Vault
- Gemalto SafeNet KeySecure Appliance
- Townsend Alliance Key Manager
- Entrust KeyControl

Each MySQL Server instance must be registered separately as a client for KMIP. If two or more MySQL Server instances use the same set of credentials, they can interfere with each other's functioning.

The `keyring_okv` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, "General-Purpose Keyring Key-Management Functions".
- C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, "The Keyring Service".

Example (using the SQL interface):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the characteristics of key values permitted by `keyring_okv`, Section 6.4.4.13, "Supported Keyring Key Types and Lengths".

To install `keyring_okv`, use the general instructions found in Section 6.4.4.3, "Keyring Plugin Installation", together with the configuration information specific to `keyring_okv` found here.

- General keyring_okv Configuration
- Configuring keyring_okv for Oracle Key Vault
- Configuring keyring_okv for Gemalto SafeNet KeySecure Appliance
- Configuring keyring_okv for Townsend Alliance Key Manager
- Configuring keyring_okv for Entrust KeyControl
- Password-Protecting the keyring_okv Key File

General keyring_okv Configuration

Regardless of which KMIP back end the `keyring_okv` plugin uses for keyring storage, the `keyring_okv_conf_dir` system variable configures the location of the directory used by `keyring_okv` for its support files. The default value is empty, so you must set the variable to name a properly configured directory before the plugin can communicate with the KMIP back end. Unless you do so, `keyring_okv` writes a message to the error log during server startup that it cannot communicate:

```
[Warning] Plugin keyring_okv reported: 'For keyring_okv to be
initialized, please point the keyring_okv_conf_dir variable to a directory
containing Oracle Key Vault configuration file and ssl materials'
```

The `keyring_okv_conf_dir` variable must name a directory that contains the following items:

- `okvclient.ora`: A file that contains details of the KMIP back end with which `keyring_okv` communicates.
- `ssl`: A directory that contains the certificate and key files required to establish a secure connection with the KMIP back end: `CA.pem`, `cert.pem`, and `key.pem`. If the key file is password-protected,

the `ssl` directory can contain a single-line text file named `password.txt` containing the password needed to decrypt the key file.

Both the `okvclient.ora` file and `ssl` directory with the certificate and key files are required for `keyring_okv` to work properly. The procedure used to populate the configuration directory with these files depends on the KMIP back end used with `keyring_okv`, as described elsewhere.

The configuration directory used by `keyring_okv` as the location for its support files should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the `/usr/local/mysql/mysql-keyring-okv` directory, the following commands (executed as `root`) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring-okv
chmod 750 mysql-keyring-okv
chown mysql mysql-keyring-okv
chgrp mysql mysql-keyring-okv
```

To be usable during the server startup process, `keyring_okv` must be loaded using the `--early-plugin-load` option. Also, set the `keyring_okv_conf_dir` system variable to tell `keyring_okv` where to find its configuration directory. For example, use these lines in the server `my.cnf` file, adjusting the `.so` suffix and directory location for your platform as necessary:

```
[mysqld]
early-plugin-load=keyring_okv.so
keyring_okv_conf_dir=/usr/local/mysql/mysql-keyring-okv
```

For additional information about `keyring_okv_conf_dir`, see Section 6.4.4.19, “Keyring System Variables”.

Configuring keyring_okv for Oracle Key Vault

The discussion here assumes that you are familiar with Oracle Key Vault. Some pertinent information sources:

- Oracle Key Vault site
- Oracle Key Vault documentation

In Oracle Key Vault terminology, clients that use Oracle Key Vault to store and retrieve security objects are called endpoints. To communicate with Oracle Key Vault, it is necessary to register as an endpoint and enroll by downloading and installing endpoint support files. Note that you must register a separate endpoint for each MySQL Server instance. If two or more MySQL Server instances use the same endpoint, they can interfere with each other’s functioning.

The following procedure briefly summarizes the process of setting up `keyring_okv` for use with Oracle Key Vault:

1. Create the configuration directory for the `keyring_okv` plugin to use.
2. Register an endpoint with Oracle Key Vault to obtain an enrollment token.
3. Use the enrollment token to obtain the `okvclient.jar` client software download.
4. Install the client software to populate the `keyring_okv` configuration directory that contains the Oracle Key Vault support files.

Use the following procedure to configure `keyring_okv` and Oracle Key Vault to work together. This description only summarizes how to interact with Oracle Key Vault. For details, visit the Oracle Key Vault site and consult the Oracle Key Vault Administrator's Guide.

1. Create the configuration directory that contains the Oracle Key Vault support files, and make sure that the `keyring_okv_conf_dir` system variable is set to name that directory (for details, see General keyring_okv Configuration).

2. Log in to the Oracle Key Vault management console as a user who has the System Administrator role.

3. Select the Endpoints tab to arrive at the Endpoints page. On the Endpoints page, click Add.

4. Provide the required endpoint information and click Register. The endpoint type should be Other. Successful registration results in an enrollment token.

5. Log out from the Oracle Key Vault server.

6. Connect again to the Oracle Key Vault server, this time without logging in. Use the endpoint enrollment token to enroll and request the `okvclient.jar` software download. Save this file to your system.

7. Install the `okvclient.jar` file using the following command (you must have JDK 1.4 or higher):

   ```
   java -jar okvclient.jar -d dir_name [-v]
   ```

   The directory name following the `-d` option is the location in which to install extracted files. The `-v` option, if given, causes log information to be produced that may be useful if the command fails.

   When the command asks for an Oracle Key Vault endpoint password, do not provide one. Instead, press Enter. (The result is that no password is required when the endpoint connects to Oracle Key Vault.)

8. The preceding command produces an `okvclient.ora` file, which should be in this location under the directory named by the `-d` option in the preceding `java -jar` command:

   ```
   install_dir/conf/okvclient.ora
   ```

   The file contents include lines that look something like this:

   ```
   SERVER=host_ip:port_num
   STANDBY_SERVER=host_ip:port_num
   ```

   The `SERVER` variable is mandatory, and the `STANDBY_SERVER` variable is optional. The `keyring_okv` plugin attempts to communicate with the server running on the host named by the `SERVER` variable and falls back to `STANDBY_SERVER` if that fails.

   From MySQL 8.0.29, you can specify more than one standby server (up to a maximum of 64). If you do, the `keyring_okv` plugin iterates over them until it can establish a connection, and fails if it cannot. To add extra standby servers, edit the `okvclient.ora` file to specify the IP addresses and port numbers of the servers as a comma-separated list in the value of the `STANDBY_SERVER` variable. For example:

   ```
   STANDBY_SERVER=host_ip:port_num,host_ip:port_num,host_ip:port_num,host_ip:port_num
   ```

   Ensure that the list of standby servers is kept short, accurate, and up to date, and servers that are no longer valid are removed. There is a 20-second wait for each connection attempt, so the presence of a long list of invalid servers can significantly affect the `keyring_okv` plugin's connection time and therefore the server startup time.

9. Go to the Oracle Key Vault installer directory and test the setup by running this command:

   ```
   okvutil/bin/okvutil list
   ```

   The output should look something like this:

   ```
   Unique ID                            Type           Identifier
   255AB8DE-C97F-482C-E053-0100007F28B9 Symmetric Key -
   264BF6E0-A20E-7C42-E053-0100007FB29C Symmetric Key -
   ```

   For a fresh Oracle Key Vault server (a server without any key in it), the output looks like this instead, to indicate that there are no keys in the vault:

```
no objects found
```

10. Use this command to extract the `ssl` directory containing SSL materials from the `okvclient.jar` file:

    ```
    jar xf okvclient.jar ssl
    ```

11. Copy the Oracle Key Vault support files (the `okvclient.ora` file and the `ssl` directory) into the configuration directory.

12. (Optional) If you wish to password-protect the key file, use the instructions in Password-Protecting the keyring_okv Key File.

After completing the preceding procedure, restart the MySQL server. It loads the `keyring_okv` plugin and `keyring_okv` uses the files in its configuration directory to communicate with Oracle Key Vault.

Configuring keyring_okv for Gemalto SafeNet KeySecure Appliance

Gemalto SafeNet KeySecure Appliance uses the KMIP protocol (version 1.1 or 1.2). The `keyring_okv` keyring plugin (which supports KMIP 1.1) can use KeySecure as its KMIP back end for keyring storage.

Use the following procedure to configure `keyring_okv` and KeySecure to work together. The description only summarizes how to interact with KeySecure. For details, consult the section named Add a KMIP Server in the KeySecure User Guide.

1. Create the configuration directory that contains the KeySecure support files, and make sure that the `keyring_okv_conf_dir` system variable is set to name that directory (for details, see General keyring_okv Configuration).

2. In the configuration directory, create a subdirectory named `ssl` to use for storing the required SSL certificate and key files.

3. In the configuration directory, create a file named `okvclient.ora`. It should have following format:

   ```
   SERVER=host_ip:port_num
   STANDBY_SERVER=host_ip:port_num
   ```

   For example, if KeySecure is running on host 198.51.100.20 and listening on port 9002, and also running on alternative host 203.0.113.125 and listening on port 8041, the `okvclient.ora` file looks like this:

   ```
   SERVER=198.51.100.20:9002
   STANDBY_SERVER=203.0.113.125:8041
   ```

   From MySQL 8.0.29, you can specify more than one standby server (up to a maximum of 64). If you do, the `keyring_okv` plugin iterates over them until it can establish a connection, and fails if it cannot. To add extra standby servers, edit the `okvclient.ora` file to specify the IP addresses and port numbers of the servers as a comma-separated list in the value of the `STANDBY_SERVER` variable. For example:

   ```
   STANDBY_SERVER=host_ip:port_num,host_ip:port_num,host_ip:port_num,host_ip:port_num
   ```

   Ensure that the list of standby servers is kept short, accurate, and up to date, and servers that are no longer valid are removed. There is a 20-second wait for each connection attempt, so the presence of a long list of invalid servers can significantly affect the `keyring_okv` plugin's connection time and therefore the server startup time.

4. Connect to the KeySecure Management Console as an administrator with credentials for Certificate Authorities access.

5. Navigate to Security >> Local CAs and create a local certificate authority (CA).

6. Go to Trusted CA Lists. Select Default and click on Properties. Then select Edit for Trusted Certificate Authority List and add the CA just created.

7. Download the CA and save it in the `ssl` directory as a file named `CA.pem`.

8. Navigate to Security >> Certificate Requests and create a certificate. Then you can download a compressed `tar` file containing certificate PEM files.

9. Extract the PEM files from in the downloaded file. For example, if the file name is `csr_w_pk_pkcs8.gz`, decompress and unpack it using this command:

   ```
   tar zxvf csr_w_pk_pkcs8.gz
   ```

   Two files result from the extraction operation: `certificate_request.pem` and `private_key_pkcs8.pem`.

10. Use this `openssl` command to decrypt the private key and create a file named `key.pem`:

    ```
    openssl pkcs8 -in private_key_pkcs8.pem -out key.pem
    ```

11. Copy the `key.pem` file into the `ssl` directory.

12. Copy the certificate request in `certificate_request.pem` into the clipboard.

13. Navigate to Security >> Local CAs. Select the same CA that you created earlier (the one you downloaded to create the `CA.pem` file), and click Sign Request. Paste the Certificate Request from the clipboard, choose a certificate purpose of Client (the keyring is a client of KeySecure), and click Sign Request. The result is a certificate signed with the selected CA in a new page.

14. Copy the signed certificate to the clipboard, then save the clipboard contents as a file named `cert.pem` in the `ssl` directory.

15. (Optional) If you wish to password-protect the key file, use the instructions in Password-Protecting the keyring_okv Key File.

After completing the preceding procedure, restart the MySQL server. It loads the `keyring_okv` plugin and `keyring_okv` uses the files in its configuration directory to communicate with KeySecure.

Configuring keyring_okv for Townsend Alliance Key Manager

Townsend Alliance Key Manager uses the KMIP protocol. The `keyring_okv` keyring plugin can use Alliance Key Manager as its KMIP back end for keyring storage. For additional information, see Alliance Key Manager for MySQL.

Configuring keyring_okv for Entrust KeyControl

Entrust KeyControl uses the KMIP protocol. The `keyring_okv` keyring plugin can use Entrust KeyControl as its KMIP back end for keyring storage. For additional information, see the Oracle MySQL and Entrust KeyControl with nShield HSM Integration Guide.

Password-Protecting the keyring_okv Key File

You can optionally protect the key file with a password and supply a file containing the password to enable the key file to be decrypted. To so do, change location to the `ssl` directory and perform these steps:

1. Encrypt the `key.pem` key file. For example, use a command like this, and enter the encryption password at the prompts:

   ```
   $> openssl rsa -des3 -in key.pem -out key.pem.new
   Enter PEM pass phrase:
   Verifying - Enter PEM pass phrase:
   ```

2. Save the encryption password in a single-line text file named `password.txt` in the `ssl` directory.
3. Verify that the encrypted key file can be decrypted using the following command. The decrypted file should display on the console:

```
$> openssl rsa -in key.pem.new -passin file:password.txt
```

4. Remove the original `key.pem` file and rename `key.pem.new` to `key.pem`.
5. Change the ownership and access mode of new `key.pem` file and `password.txt` file as necessary to ensure that they have the same restrictions as other files in the `ssl` directory.

6.4.4.9 Using the keyring_aws Amazon Web Services Keyring Plugin

Note

The `keyring_aws` plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The `keyring_aws` keyring plugin communicates with the Amazon Web Services Key Management Service (AWS KMS) as a back end for key generation and uses a local file for key storage. All keyring material is generated exclusively by the AWS server, not by `keyring_aws`.

MySQL Enterprise Edition can work with `keyring_aws` on Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Debian, Ubuntu, macOS, and Windows. MySQL Enterprise Edition does not support the use of `keyring_aws` on these platforms:

- EL6
- Generic Linux (glibc2.12)
- SLES 12 (with versions after MySQL Server 5.7)
- Solaris

The discussion here assumes that you are familiar with AWS in general and KMS in particular. Some pertinent information sources:

- AWS site
- KMS documentation

The following sections provide configuration and usage information for the `keyring_aws` keyring plugin:

- keyring_aws Configuration
- keyring_aws Operation
- keyring_aws Credential Changes

keyring_aws Configuration

To install `keyring_aws`, use the general instructions found in Section 6.4.4.3, “Keyring Plugin Installation”, together with the plugin-specific configuration information found here.

The plugin library file contains the `keyring_aws` plugin and two loadable functions, `keyring_aws_rotate_cmk()` and `keyring_aws_rotate_keys()`.

To configure `keyring_aws`, you must obtain a secret access key that provides credentials for communicating with AWS KMS and write it to a configuration file:

1. Create an AWS KMS account.
2. Use AWS KMS to create a secret access key ID and secret access key. The access key serves to verify your identity and that of your applications.
3. Use the AWS KMS account to create a KMS key ID. At MySQL startup, set the `keyring_aws_cmk_id` system variable to the CMK ID value. This variable is mandatory and there is no default. (Its value can be changed at runtime if desired using `SET GLOBAL`.)
4. If necessary, create the directory in which the configuration file should be located. The directory should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use `/usr/local/mysql/mysql-keyring/keyring_aws_conf` as the file name, the following commands (executed as `root`) create its parent directory and set the directory mode and ownership:

   ```
   $> cd /usr/local/mysql
   $> mkdir mysql-keyring
   $> chmod 750 mysql-keyring
   $> chown mysql mysql-keyring
   $> chgrp mysql mysql-keyring
   ```

   At MySQL startup, set the `keyring_aws_conf_file` system variable to `/usr/local/mysql/mysql-keyring/keyring_aws_conf` to indicate the configuration file location to the server.
5. Prepare the `keyring_aws` configuration file, which should contain two lines:

   - Line 1: The secret access key ID
   - Line 2: The secret access key

   For example, if the key ID is `wwwwwwwwwwwwEXAMPLE` and the key is `xxxxxxxxxxxxx/yyyyyyy/zzzzzzzzEXAMPLEKEY`, the configuration file looks like this:

   ```
   wwwwwwwwwwwwEXAMPLE
   xxxxxxxxxxxxx/yyyyyyy/zzzzzzzzEXAMPLEKEY
   ```

To be usable during the server startup process, `keyring_aws` must be loaded using the `--early-plugin-load` option. The `keyring_aws_cmk_id` system variable is mandatory and configures the KMS key ID obtained from the AWS KMS server. The `keyring_aws_conf_file` and `keyring_aws_data_file` system variables optionally configure the locations of the files used by the `keyring_aws` plugin for configuration information and data storage. The file location variable default values are platform specific. To configure the locations explicitly, set the variable values at startup. For example, use these lines in the server `my.cnf` file, adjusting the `.so` suffix and file locations for your platform as necessary:

```
[mysqld]
early-plugin-load=keyring_aws.so
keyring_aws_cmk_id='arn:aws:kms:us-west-2:111122223333:key/abcd1234-ef56-ab12-cd34-ef56abcd1234'
keyring_aws_conf_file=/usr/local/mysql/mysql-keyring/keyring_aws_conf
keyring_aws_data_file=/usr/local/mysql/mysql-keyring/keyring_aws_data
```

For the `keyring_aws` plugin to start successfully, the configuration file must exist and contain valid secret access key information, initialized as described previously. The storage file need not exist. If it does not, `keyring_aws` attempts to create it (as well as its parent directory, if necessary).

For additional information about the system variables used to configure the `keyring_aws` plugin, see Section 6.4.4.19, “Keyring System Variables”.

Start the MySQL server and install the functions associated with the `keyring_aws` plugin. This is a one-time operation, performed by executing the following statements, adjusting the `.so` suffix for your platform as necessary:

```
CREATE FUNCTION keyring_aws_rotate_cmk RETURNS INTEGER
  SONAME 'keyring_aws.so';
```

```
CREATE FUNCTION keyring_aws_rotate_keys RETURNS INTEGER
  SONAME 'keyring_aws.so';
```

For additional information about the `keyring_aws` functions, see Section 6.4.4.16, “Plugin-Specific Keyring Key-Management Functions”.

keyring_aws Operation

At plugin startup, the `keyring_aws` plugin reads the AWS secret access key ID and key from its configuration file. It also reads any encrypted keys contained in its storage file into its in-memory cache.

During operation, `keyring_aws` maintains encrypted keys in the in-memory cache and uses the storage file as local persistent storage. Each keyring operation is transactional: `keyring_aws` either successfully changes both the in-memory key cache and the keyring storage file, or the operation fails and the keyring state remains unchanged.

To ensure that keys are flushed only when the correct keyring storage file exists, `keyring_aws` stores a SHA-256 checksum of the keyring in the file. Before updating the file, the plugin verifies that it contains the expected checksum.

The `keyring_aws` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by these functions are accessible at two levels:

- SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, “The Keyring Service”.

Example (using the SQL interface):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

In addition, the `keyring_aws_rotate_cmk()` and `keyring_aws_rotate_keys()` functions “extend” the keyring plugin interface to provide AWS-related capabilities not covered by the standard keyring service interface. These capabilities are accessible only by calling these functions using SQL. There are no corresponding C-language key service functions.

For information about the characteristics of key values permitted by `keyring_aws`, see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

keyring_aws Credential Changes

Assuming that the `keyring_aws` plugin has initialized properly at server startup, it is possible to change the credentials used for communicating with AWS KMS:

1. Use AWS KMS to create a new secret access key ID and secret access key.
2. Store the new credentials in the configuration file (the file named by the `keyring_aws_conf_file` system variable). The file format is as described previously.
3. Reinitialize the `keyring_aws` plugin so that it re-reads the configuration file. Assuming that the new credentials are valid, the plugin should initialize successfully.

   There are two ways to reinitialize the plugin:

   - Restart the server. This is simpler and has no side effects, but is not suitable for installations that require minimal server downtime with as few restarts as possible.
   - Reinitialize the plugin without restarting the server by executing the following statements, adjusting the `.so` suffix for your platform as necessary:

```
UNINSTALL PLUGIN keyring_aws;
INSTALL PLUGIN keyring_aws SONAME 'keyring_aws.so';
```

Note

In addition to loading a plugin at runtime, `INSTALL PLUGIN` has the side effect of registering the plugin it in the `mysql.plugin` system table. Because of this, if you decide to stop using `keyring_aws`, it is not sufficient to remove the `--early-plugin-load` option from the set of options used to start the server. That stops the plugin from loading early, but the server still attempts to load it when it gets to the point in the startup sequence where it loads the plugins registered in `mysql.plugin`.

Consequently, if you execute the `UNINSTALL PLUGIN` plus `INSTALL PLUGIN` sequence just described to change the AWS KMS credentials, then to stop using `keyring_aws`, it is necessary to execute `UNINSTALL PLUGIN` again to unregister the plugin in addition to removing the `--early-plugin-load` option.

6.4.4.10 Using the HashiCorp Vault Keyring Plugin

Note

The `keyring_hashicorp` plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The `keyring_hashicorp` keyring plugin communicates with HashiCorp Vault for back end storage. The plugin supports HashiCorp Vault AppRole authentication. No key information is permanently stored in MySQL server local storage. (An optional in-memory key cache may be used as intermediate storage.) Random key generation is performed on the MySQL server side, with the keys subsequently stored to Hashicorp Vault.

The `keyring_hashicorp` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, “The Keyring Service”.

Example (using the SQL interface):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the characteristics of key values permitted by `keyring_hashicorp`, see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

To install `keyring_hashicorp`, use the general instructions found in Section 6.4.4.3, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_hashicorp` found here. Plugin-specific configuration includes preparation of the certificate and key files needed for connecting to HashiCorp Vault, as well as configuring HashiCorp Vault itself. The following sections provide the necessary instructions.

- Certificate and Key Preparation
- HashiCorp Vault Setup
- keyring_hashicorp Configuration

Certificate and Key Preparation

The `keyring_hashicorp` plugin requires a secure connection to the HashiCorp Vault server, employing the HTTPS protocol. A typical setup includes a set of certificate and key files:

- `company.crt`: A custom CA certificate belonging to the organization. This file is used both by HashiCorp Vault server and the `keyring_hashicorp` plugin.
- `vault.key`: The private key of the HashiCorp Vault server instance. This file is used by HashiCorp Vault server.
- `vault.crt`: The certificate of the HashiCorp Vault server instance. This file must be signed by the organization CA certificate.

The following instructions describe how to create the certificate and key files using OpenSSL. (If you already have those files, proceed to HashiCorp Vault Setup.) The instructions as shown apply to Linux platforms and may require adjustment for other platforms.

Important

Certificates generated by these instructions are self-signed, which may not be very secure. After you gain experience using such files, consider obtaining certificate/key material from a registered certificate authority.

1. Prepare the company and HashiCorp Vault server keys.

   Use the following commands to generate the key files:

   ```
   openssl genrsa -aes256 -out company.key 4096
   openssl genrsa -aes256 -out vault.key 2048
   ```

   The commands produce files holding the company private key (`company.key`) and the Vault server private key (`vault.key`). The keys are randomly generated RSA keys of 4,096 and 2,048 bits, respectively.

   Each command prompts for a password. For testing purposes, the password is not required. To disable it, omit the `-aes256` argument.

   The key files hold sensitive information and should be stored in a secure location. The password (also sensitive) is required later, so write it down and store it in a secure location.

   (Optional) To check key file content and validity, use the following commands:

   ```
   openssl rsa -in company.key -check
   openssl rsa -in vault.key -check
   ```

2. Create the company CA certificate.

   Use the following command to create a company CA certificate file named `company.crt` that is valid for 365 days (enter the command on a single line):

   ```
   openssl req -x509 -new -nodes -key company.key
     -sha256 -days 365 -out company.crt
   ```

   If you used the `-aes256` argument to perform key encryption during key generation, you are prompted for the company key password during CA certificate creation. You are also prompted for information about the certificate holder (that is, you or your company), as shown here:

   ```
   Country Name (2 letter code) [AU]:
   State or Province Name (full name) [Some-State]:
   Locality Name (eg, city) []:
   Organization Name (eg, company) [Internet Widgits Pty Ltd]:
   Organizational Unit Name (eg, section) []:
   Common Name (e.g. server FQDN or YOUR name) []:
   Email Address []:
   ```

Answer the prompts with appropriate values.

3. Create a certificate signing request.

   To create a HashiCorp Vault server certificate, a Certificate Signing Request (CSR) must be prepared for the newly created server key. Create a configuration file named `request.conf` containing the following lines. If the HashiCorp Vault server does not run on the local host, substitute appropriate CN and IP values, and make any other changes required.

   ```
   [req]
   distinguished_name = vault
   x509_entensions = v3_req
   prompt = no

   [vault]
   C = US
   ST = CA
   L = RWC
   O = Company
   CN = 127.0.0.1

   [v3_req]
   subjectAltName = @alternatives
   authorityKeyIdentifier = keyid,issuer
   basicConstraints = CA:TRUE

   [alternatives]
   IP = 127.0.0.1
   ```

   Use this command to create the signing request:

   ```
   openssl req -new -key vault.key -config request.conf -out request.csr
   ```

   The output file (`request.csr`) is an intermediate file that serves as input for creation of the server certificate.

4. Create the HashiCorp Vault server certificate.

   Sign the combined information from the HashiCorp Vault server key (`vault.key`) and the CSR (`request.csr`) with the company certificate (`company.crt`) to create the HashiCorp Vault server certificate (`vault.crt`). Use the following command to do this (enter the command on a single line):

   ```
   openssl x509 -req -in request.csr
     -CA company.crt -CAkey company.key -CAcreateserial
     -out vault.crt -days 365 -sha256
   ```

   To make the `vault.crt` server certificate useful, append the contents of the `company.crt` company certificate to it. This is required so that the company certificate is delivered along with the server certificate in requests.

   ```
   cat company.crt >> vault.crt
   ```

   If you display the contents of the `vault.crt` file, it should look like this:

   ```
   -----BEGIN CERTIFICATE-----
   ... content of HashiCorp Vault server certificate ...
   -----END CERTIFICATE-----
   -----BEGIN CERTIFICATE-----
   ... content of company certificate ...
   -----END CERTIFICATE-----
   ```

HashiCorp Vault Setup

The following instructions describe how to create a HashiCorp Vault setup that facilitates testing the `keyring_hashicorp` plugin.

Important

A test setup is similar to a production setup, but production use of HashiCorp Vault entails additional security considerations such as use of non-self-signed certificates and storing the company certificate in the system trust store. You must implement whatever additional security steps are needed to satisfy your operational requirements.

These instructions assume availability of the certificate and key files created in Certificate and Key Preparation. See that section if you do not have those files.

1. Fetch the HashiCorp Vault binary.

   Download the HashiCorp Vault binary appropriate for your platform from https://www.vaultproject.io/downloads.html.

   Extract the content of the archive to produce the executable `vault` command, which is used to perform HashiCorp Vault operations. If necessary, add the directory where you install the command to the system path.

   (Optional) HashiCorp Vault supports autocomplete options that make it easier to use. For more information, see https://learn.hashicorp.com/vault/getting-started/install#command-completion.

2. Create the HashiCorp Vault server configuration file.

   Prepare a configuration file named `config.hcl` with the following content. For the `tls_cert_file`, `tls_key_file`, and `path` values, substitute path names appropriate for your system.

   ```
   listener "tcp" {
     address="127.0.0.1:8200"
     tls_cert_file="/home/username/certificates/vault.crt"
     tls_key_file="/home/username/certificates/vault.key"
   }

   storage "file" {
     path = "/home/username/vaultstorage/storage"
   }

   ui = true
   ```

3. Start the HashiCorp Vault server.

   To start the Vault server, use the following command, where the `-config` option specifies the path to the configuration file just created:

   ```
   vault server -config=config.hcl
   ```

   During this step, you may be prompted for a password for the Vault server private key stored in the `vault.key` file.

   The server should start, displaying some information on the console (IP, port, and so forth).

   So that you can enter the remaining commands, put the `vault server` command in the background or open another terminal before continuing.

4. Initialize the HashiCorp Vault server.

Note

The operations described in this step are required only when starting Vault the first time, to obtain the unseal key and root token. Subsequent Vault instance restarts require only unsealing using the unseal key.

Issue the following commands (assuming Bourne shell syntax):

```
export VAULT_SKIP_VERIFY=1
vault operator init -n 1 -t 1
```

The first command enables the `vault` command to temporarily ignore the fact that no company certificate has been added to the system trust store. It compensates for the fact that our self-signed CA is not added to that store. (For production use, such a certificate should be added.)

The second command creates a single unseal key with a requirement for a single unseal key to be present for unsealing. (For production use, an instance would have multiple unseal keys with up to that many keys required to be entered to unseal it. The unseal keys should be delivered to key custodians within the company. Use of a single key might be considered a security issue because that permits the vault to be unsealed by a single key custodian.)

Vault should reply with information about the unseal key and root token, plus some additional text (the actual unseal key and root token values differ from those shown here):

```
...
Unseal Key 1: I2xwcFQc892O0Nt2pBiRNlnkHzTUrWS+JybL39BjcOE=
Initial Root Token: s.vTvXeo3tPEYehfcd9WH7oUKz
...
```

Store the unseal key and root token in a secure location.

5. Unseal the HashiCorp Vault server.

   Use this command to unseal the Vault server:

   ```
   vault operator unseal
   ```

   When prompted to enter the unseal key, use the key obtained previously during Vault initialization.

   Vault should produce output indicating that setup is complete and the vault is unsealed.

6. Log in to the HashiCorp Vault server and verify its status.

   Prepare the environment variables required for logging in as root:

   ```
   vault login s.vTvXeo3tPEYehfcd9WH7oUKz
   ```

   For the token value in that command, substitute the content of the root token obtained previously during Vault initialization.

   Verify the Vault server status:

   ```
   vault status
   ```

   The output should contain these lines (among others):

   ```
   ...
   Initialized     true
   Sealed          false
   ...
   ```

7. Set up HashiCorp Vault authentication and storage.

   

   Note

   The operations described in this step are needed only the first time the Vault instance is run. They need not be repeated afterward.

   Enable the AppRole authentication method and verify that it is in the authentication method list:

   ```
   vault auth enable approle
   vault auth list
   ```

   Enable the Vault KeyValue storage engine:

   ```
   vault secrets enable -version=1 kv
   ```

   Create and set up a role for use with the `keyring_hashicorp` plugin (enter the command on a single line):

   ```
   vault write auth/approle/role/mysql token_num_uses=0
     token_ttl=20m token_max_ttl=30m secret_id_num_uses=0
   ```

8. Add an AppRole security policy.

   

   Note

   The operations described in this step are needed only the first time the Vault instance is run. They need not be repeated afterward.

   Prepare a policy that to permit the previously created role to access appropriate secrets. Create a new file named `mysql.hcl` with the following content:

   ```
   path "kv/mysql/*" {
     capabilities = ["create", "read", "update", "delete", "list"]
   }
   ```

   

   Note

   `kv/mysql/` in this example may need adjustment per your local installation policies and security requirements. If so, make the same adjustment wherever else `kv/mysql/` appears in these instructions.

   Import the policy file to the Vault server to create a policy named `mysql-policy`, then assign the policy to the new role:

   ```
   vault policy write mysql-policy mysql.hcl
   vault write auth/approle/role/mysql policies=mysql-policy
   ```

   Obtain the ID of the newly created role and store it in a secure location:

   ```
   vault read auth/approle/role/mysql/role-id
   ```

   Generate a secret ID for the role and store it in a secure location:

   ```
   vault write -f auth/approle/role/mysql/secret-id
   ```

   After these AppRole role ID and secret ID credentials are generated, they are expected to remain valid indefinitely. They need not be generated again and the `keyring_hashicorp` plugin can be configured with them for use on an ongoing basis. For more information about AuthRole authentication, visit https://www.vaultproject.io/docs/auth/approle.html.

keyring_hashicorp Configuration

The plugin library file contains the `keyring_hashicorp` plugin and a loadable function, `keyring_hashicorp_update_config()`. When the plugin initializes and terminates, it automatically loads and unloads the function. There is no need to load and unload the function manually.

The `keyring_hashicorp` plugin supports the configuration parameters shown in the following table. To specify these parameters, assign values to the corresponding system variables.

Configuration Parameter	System Variable	Mandatory
HashiCorp Server URL	`keyring_hashicorp_server_url`	No
AppRole role ID	`keyring_hashicorp_role_id`	Yes
AppRole secret ID	`keyring_hashicorp_secret_id`	Yes
Store path	`keyring_hashicorp_store_path`	Yes
Authorization Path	`keyring_hashicorp_auth_path`	No
CA certificate file path	`keyring_hashicorp_ca_path`	No
Cache control	`keyring_hashicorp_caching`	No

To be usable during the server startup process, `keyring_hashicorp` must be loaded using the `--early-plugin-load` option. As indicated by the preceding table, several plugin-related system variables are mandatory and must also be set. For example, use these lines in the server `my.cnf` file, adjusting the `.so` suffix and file locations for your platform as necessary:

```
[mysqld]
early-plugin-load=keyring_hashicorp.so
keyring_hashicorp_role_id='ee3b495c-d0c9-11e9-8881-8444c71c32aa'
keyring_hashicorp_secret_id='0512af29-d0ca-11e9-95ee-0010e00dd718'
keyring_hashicorp_store_path='/v1/kv/mysql'
keyring_hashicorp_auth_path='/v1/auth/approle/login'
```

Note

Per the HashiCorp documentation, all API routes are prefixed with a protocol version (which you can see in the preceding example as `/v1/` in the `keyring_hashicorp_store_path` and `keyring_hashicorp_auth_path` values). If HashiCorp develops new protocol versions, it may be necessary to change `/v1/` to something else in your configuration.

MySQL Server authenticates against HashiCorp Vault using AppRole authentication. Successful authentication requires that two secrets be provided to Vault, a role ID and a secret ID, which are similar in concept to user name and password. The role ID and secret ID values to use are those obtained during the HashiCorp Vault setup procedure performed previously. To specify the two IDs, assign their respective values to the `keyring_hashicorp_role_id` and `keyring_hashicorp_secret_id` system variables. The setup procedure also results in a store path of `/v1/kv/mysql`, which is the value to assign to `keyring_hashicorp_commit_store_path`.

At plugin initialization time, `keyring_hashicorp` attempts to connect to the HashiCorp Vault server using the configuration values. If the connection is successful, the plugin stores the values in corresponding system variables that have `_commit_` in their name. For example, upon successful connection, the plugin stores the values of `keyring_hashicorp_role_id` and `keyring_hashicorp_store_path` in `keyring_hashicorp_commit_role_id` and `keyring_hashicorp_commit_store_path`.

Reconfiguration at runtime can be performed with the assistance of the `keyring_hashicorp_update_config()` function:

1. Use `SET` statements to assign the desired new values to the configuration system variables shown in the preceding table. These assignments in themselves have no effect on ongoing plugin operation.

2. Invoke `keyring_hashicorp_update_config()` to cause the plugin to reconfigure and reconnect to the HashiCorp Vault server using the new variable values.

3. If the connection is successful, the plugin stores the updated configuration values in corresponding system variables that have `_commit_` in their name.

For example, if you have reconfigured HashiCorp Vault to listen on port 8201 rather than the default 8200, reconfigure `keyring_hashicorp` like this:

```
mysql> SET GLOBAL keyring_hashicorp_server_url = 'https://127.0.0.1:8201';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT keyring_hashicorp_update_config();
+-------------------------------------+
| keyring_hashicorp_update_config()   |
+-------------------------------------+
| Configuration update was successful. |
+-------------------------------------+
1 row in set (0.03 sec)
```

If the plugin is not able to connect to HashiCorp Vault during initialization or reconfiguration and there was no existing connection, the `_commit_` system variables are set to `'Not committed'` for string-valued variables, and `OFF` for Boolean-valued variables. If the plugin is not able to connect but there was an existing connection, that connection remains active and the `_commit_` variables reflect the values used for it.

Note

If you do not set the mandatory system variables at server startup, or if some other plugin initialization error occurs, initialization fails. In this case, you can use the runtime reconfiguration procedure to initialize the plugin without restarting the server.

For additional information about the `keyring_hashicorp` plugin-specific system variables and function, see Section 6.4.4.19, “Keyring System Variables”, and Section 6.4.4.16, “Plugin-Specific Keyring Key-Management Functions”.

6.4.4.11 Using the Oracle Cloud Infrastructure Vault Keyring Component

Note

The Oracle Cloud Infrastructure Vault keyring component is included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

`component_keyring_oci` is part of the component infrastructure that communicates with Oracle Cloud Infrastructure Vault for back end storage. No key information is permanently stored in MySQL server local storage. All keys are stored in Oracle Cloud Infrastructure Vault, making this component well suited for Oracle Cloud Infrastructure MySQL customers for management of their MySQL Enterprise Edition keys.

In MySQL 8.0.24, MySQL Keyring began transitioning from plugins to use the component infrastructure. The introduction of `component_keyring_oci` in MySQL 8.0.31 is a continuation of that effort. For more information, see Keyring Components Versus Keyring Plugins.

Note

Only one keyring component or plugin should be enabled at a time. Enabling multiple keyring components or plugins is unsupported and results may not be as anticipated.

To use `component_keyring_oci` for keystore management, you must:

1. Write a manifest that tells the server to load `component_keyring_oci`, as described in Section 6.4.4.2, “Keyring Component Installation”.

2. Write a configuration file for `component_keyring_oci`, as described here.

After writing a manifest and configuration file, you should be able to access keys that were created using the `keyring_oci` plugin, provided that you specify the same set of configuration options to initialize the keyring component. The built-in backward compatibility of `component_keyring_oci` simplifies migrating from the keyring plugin to the component.

- Configuration Notes
- Verify the Component Installation
- Vault Keyring Component Usage

Configuration Notes

When it initializes, `component_keyring_oci` reads either a global configuration file, or a global configuration file paired with a local configuration file:

- The component attempts to read its global configuration file from the directory where the component library file is installed (that is, the server plugin directory).
- If the global configuration file indicates use of a local configuration file, the component attempts to read its local configuration file from the data directory.
- Although global and local configuration files are located in different directories, the file name is `component_keyring_oci.cnf` in both locations.
- It is an error for no configuration file to exist. `component_keyring_oci` cannot initialize without a valid configuration.

Local configuration files permit setting up multiple server instances to use `component_keyring_oci`, such that component configuration for each server instance is specific to a given data directory instance. This enables the same keyring component to be used with a distinct Oracle Cloud Infrastructure Vault for each instance.

You are assumed to be familiar with Oracle Cloud Infrastructure concepts, but the following documentation may be helpful when setting up resources to be used by `component_keyring_oci`:

- Overview of Vault
- Required Keys and OCIDs
- Managing Keys
- Managing Compartments
- Managing Vaults
- Managing Secrets

`component_keyring_oci` configuration files have these properties:

- A configuration file must be in valid JSON format.
- A configuration file permits these configuration items:
  - `"read_local_config"`: This item is permitted only in the global configuration file. If the item is not present, the component uses only the global configuration file. If the item is present, its value is `true` or `false`, indicating whether the component should read configuration information from the local configuration file.

If the `"read_local_config"` item is present in the global configuration file along with other items, the component checks the `"read_local_config"` item value first:

- If the value is `false`, the component processes the other items in the global configuration file and ignores the local configuration file.
- If the value is `true`, the component ignores the other items in the global configuration file and attempts to read the local configuration file.

- `“user”`: The OCID of the Oracle Cloud Infrastructure user that `component_keyring_oci` uses for connections. Prior to using `component_keyring_oci`, the user account must exist and be granted access to use the configured Oracle Cloud Infrastructure tenancy, compartment, and vault resources. To obtain the user OCID from the Console, use the instructions at Required Keys and OCIDs.

  This value is mandatory.

- `“tenancy”`: The OCID of the Oracle Cloud Infrastructure tenancy that `component_keyring_oci` uses as the location of the MySQL compartment. Prior to using `component_keyring_oci`, you must create a tenancy if it does not exist. To obtain the tenancy OCID from the Console, use the instructions at Required Keys and OCIDs.

  This value is mandatory.

- `“compartment”`: The OCID of the tenancy compartment that `component_keyring_oci` uses as the location of the MySQL keys. Prior to using `component_keyring_oci`, you must create a MySQL compartment or subcompartment if it does not exist. This compartment should contain no vault keys or vault secrets. It should not be used by systems other than MySQL Keyring. For information about managing compartments and obtaining the OCID, see Managing Compartments.

  This value is mandatory.

- `“virtual_vault”`: The OCID of the Oracle Cloud Infrastructure Vault that `component_keyring_oci` uses for encryption operations. Prior to using `component_keyring_oci`, you must create a new vault in the MySQL compartment if it does not exist. (Alternatively, you can reuse an existing vault that is in a parent compartment of the MySQL compartment.) Compartment users can see and use only the keys in their respective compartments. For information about creating a vault and obtaining the vault OCID, see Managing Vaults.

  This value is mandatory.

- `“encryption_endpoint”`: The endpoint of the Oracle Cloud Infrastructure encryption server that `component_keyring_oci` uses for generating encrypted or encoded information (ciphertext) for new keys. The encryption endpoint is vault specific and Oracle Cloud Infrastructure assigns it at vault-creation time. To obtain the endpoint OCID, view the configuration details for your keyring_oci vault, using the instructions at Managing Vaults.

  This value is mandatory.

- `"management_endpoint"`: The endpoint of the Oracle Cloud Infrastructure key management server that `component_keyring_oci` uses for listing existing keys. The key management endpoint is vault specific and Oracle Cloud Infrastructure assigns it at vault-creation time. To

obtain the endpoint OCID, view the configuration details for your keyring_oci vault, using the instructions at Managing Vaults.

This value is mandatory.

- "vaults_endpoint": The endpoint of the Oracle Cloud Infrastructure vaults server that component_keyring_oci uses for obtaining the value of secrets. The vaults endpoint is vault specific and Oracle Cloud Infrastructure assigns it at vault-creation time. To obtain the endpoint OCID, view the configuration details for your keyring_oci vault, using the instructions at Managing Vaults.

  This value is mandatory.

- "secrets_endpoint": The endpoint of the Oracle Cloud Infrastructure secrets server that component_keyring_oci uses for listing, creating, and retiring secrets. The secrets endpoint is vault specific and Oracle Cloud Infrastructure assigns it at vault-creation time. To obtain the endpoint OCID, view the configuration details for your keyring_oci vault, using the instructions at Managing Vaults.

  This value is mandatory.

- "master_key": The OCID of the Oracle Cloud Infrastructure master encryption key that component_keyring_oci uses for encryption of secrets. Prior to using component_keyring_oci, you must create a cryptographic key for the Oracle Cloud Infrastructure compartment if it does not exist. Provide a MySQL-specific name for the generated key and do not use it for other purposes. For information about key creation, see Managing Keys.

  This value is mandatory.

- "key_file": The path name of the file containing the RSA private key that component_keyring_oci uses for Oracle Cloud Infrastructure authentication. You must also upload the corresponding RSA public key using the Console. The Console displays the key fingerprint value, which you can use to set the "key_fingerprint" value. For information about generating and uploading API keys, see Required Keys and OCIDs.

  This value is mandatory.

- "key_fingerprint": The fingerprint of the RSA private key that component_keyring_oci uses for Oracle Cloud Infrastructure authentication. To obtain the key fingerprint while creating the API keys, execute this command:

  ```
  openssl rsa -pubout -outform DER -in ~/.oci/oci_api_key.pem | openssl md5 -c
  ```

  Alternatively, obtain the fingerprint from the Console, which automatically displays the fingerprint when you upload the RSA public key. For information about obtaining key fingerprints, see Required Keys and OCIDs.

  This value is mandatory.

- "ca_certificate": The path name of the CA certificate bundle file that component_keyring_oci component uses for Oracle Cloud Infrastructure certificate verification. The file contains one or more certificates for peer verification. If no file is specified, the default CA bundle installed on the system is used. If the value is set to disabled (case-sensitive), component_keyring_oci performs no certificate verification.

Given the preceding configuration file properties, to configure component_keyring_oci, create a global configuration file named component_keyring_oci.cnf in the directory where the component_keyring_oci library file is installed, and optionally create a local configuration file, also named component_keyring_oci.cnf, in the data directory.

Verify the Component Installation

After performing any component-specific configuration, start the server. Verify component installation by examining the Performance Schema `keyring_component_status` table:

```
mysql> SELECT * FROM performance_schema.keyring_component_status;
+--------------------+-------------------------------------------------------------------------+
| STATUS_KEY         | STATUS_VALUE                                                            |
+--------------------+-------------------------------------------------------------------------+
| Component_name     | component_keyring_oci                                                   |
| Author             | Oracle Corporation                                                      |
| License            | PROPRIETARY                                                             |
| Implementation_name | component_keyring_oci                                                  |
| Version            | 1.0                                                                     |
| Component_status   | Active                                                                  |
| user               | ocid1.user.oc1..aaaaaaaasqly<...>                                       |
| tenancy            | ocid1.tenancy.oc1..aaaaaaaai<...>                                       |
| compartment        | ocid1.compartment.oc1..aaaaaaaah2swh<...>                               |
| virtual_vault      | ocid1.vault.oc1.iad.bbo5xyzkaaeuk.abuwcljtmvxp4r<...>                   |
| master_key         | ocid1.key.oc1.iad.bbo5xyzkaaeuk.abuwcljrbsrewgap<...>                   |
| encryption_endpoint | bbo5xyzkaaeuk-crypto.kms.us-<...>                                      |
| management_endpoint | bbo5xyzkaaeuk-management.kms.us-<...>                                  |
| vaults_endpoint    | vaults.us-<...>                                                         |
| secrets_endpoint   | secrets.vaults.us-<...>                                                 |
| key_file           | ~/.oci/oci_api_key.pem                                                  |
| key_fingerprint    | ca:7c:e1:fa:86:b6:40:af:39:d6<...>                                      |
| ca_certificate     | disabled                                                                |
+--------------------+-------------------------------------------------------------------------+
```

A `Component_status` value of `Active` indicates that the component initialized successfully.

If the component cannot be loaded, server startup fails. Check the server error log for diagnostic messages. If the component loads but fails to initialize due to configuration problems, the server starts but the `Component_status` value is `Disabled`. Check the server error log, correct the configuration issues, and use the `ALTER INSTANCE RELOAD KEYRING` statement to reload the configuration.

It is possible to query MySQL server for the list of existing keys. To see which keys exist, examine the Performance Schema `keyring_keys` table.

```
mysql> SELECT * FROM performance_schema.keyring_keys;
+------------------------------+-------------+----------------+
| KEY_ID                       | KEY_OWNER   | BACKEND_KEY_ID |
+------------------------------+-------------+----------------+
| audit_log-20210322T130749-1 |             |                |
| MyKey                        | me@localhost |                |
| YourKey                      | me@localhost |                |
+------------------------------+-------------+----------------+
```

Vault Keyring Component Usage

`component_keyring_oci` supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, “The Keyring Service”.

Example (using the SQL interface):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the characteristics of key values permitted by `component_keyring_oci`, see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

6.4.4.12 Using the Oracle Cloud Infrastructure Vault Keyring Plugin

Note

The `keyring_oci` plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The `keyring_oci` plugin is a keyring plugin that communicates with Oracle Cloud Infrastructure Vault for back end storage. No key information is permanently stored in MySQL server local storage. All keys are stored in Oracle Cloud Infrastructure Vault, making this plugin well suited for Oracle Cloud Infrastructure MySQL customers for management of their MySQL Enterprise Edition keys.

The `keyring_oci` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 5.6.9.2, “The Keyring Service”.

Example (using the SQL interface):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the characteristics of key values permitted by `keyring_oci`, see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

To install `keyring_oci`, use the general instructions found in Section 6.4.4.3, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_oci` found here. Plugin-specific configuration involves setting a number of system variables to indicate the names or values of Oracle Cloud Infrastructure resources.

You are assumed to be familiar with Oracle Cloud Infrastructure concepts, but the following documentation may be helpful when setting up resources to be used by the `keyring_oci` plugin:

- Overview of Vault
- Resource Identifiers
- Required Keys and OCIDs
- Managing Keys
- Managing Compartments
- Managing Vaults
- Managing Secrets

The `keyring_oci` plugin supports the configuration parameters shown in the following table. To specify these parameters, assign values to the corresponding system variables.

Configuration Parameter	System Variable	Mandatory
User OCID	`keyring_oci_user`	Yes
Tenancy OCID	`keyring_oci_tenancy`	Yes
Compartment OCID	`keyring_oci_compartment`	Yes
Vault OCID	`keyring_oci_virtual_vault`	Yes
Master key OCID	`keyring_oci_master_key`	Yes
Encryption server endpoint	`keyring_oci_encryption_endpoint`	Yes

Configuration Parameter	System Variable	Mandatory
Key management server endpoint	`keyring_oci_management_endpoint`	Yes
Vaults server endpoint	`keyring_oci_vaults_endpoint`	Yes
Secrets server endpoint	`keyring_oci_secrets_endpoint`	Yes
RSA private key file	`keyring_oci_key_file`	Yes
RSA private key fingerprint	`keyring_oci_key_fingerprint`	Yes
CA certificate bundle file	`keyring_oci_ca_certificate`	No

To be usable during the server startup process, `keyring_oci` must be loaded using the `--early-plugin-load` option. As indicated by the preceding table, several plugin-related system variables are mandatory and must also be set:

- Oracle Cloud Infrastructure uses Oracle Cloud IDs (OCIDs) extensively to designate resources, and several `keyring_oci` parameters specify OCID values of the resources to use. Consequently, prior to using the `keyring_oci` plugin, these prerequisites must be satisfied:
  - A user for connecting to Oracle Cloud Infrastructure must exist. Create the user if necessary and assign the user OCID to the `keyring_oci_user` system variable.
  - The Oracle Cloud Infrastructure tenancy to be used must exist, as well as the MySQL compartment within the tenancy, and the vault within the compartment. Create these resources if necessary and make sure the user is enabled to use them. Assign the OCIDs for the tenancy, compartment and vault to the `keyring_oci_tenancy`, `keyring_oci_compartment`, and `keyring_oci_virtual_vault` system variables.
  - A master key for encryption must exist. Create it if necessary and assign its OCID to the `keyring_oci_master_key` system variable.
- Several server endpoints must be specified. These endpoints are vault specific and Oracle Cloud Infrastructure assigns them at vault-creation time. Obtain their values from the vault details page and assign them to the `keyring_oci_encryption_endpoint`, `keyring_oci_management_endpoint`, `keyring_oci_vaults_endpoint`, and `keyring_oci_secrets_endpoint` system variables.
- The Oracle Cloud Infrastructure API uses an RSA private/public key pair for authentication. To create this key pair and obtain the key fingerprint, use the instructions at Required Keys and OCIDs. Assign the private key file name and key fingerprint to the `keyring_oci_key_file` and `keyring_oci_key_fingerprint` system variables.

In addition to the mandatory system variables, `keyring_oci_ca_certificate` optionally may be set to specify a certificate authority (CA) certificate bundle file for peer authentication.

Important

If you copy a parameter from the Oracle Cloud Infrastructure Console, the copied value may include an initial `https://` part. Omit that part when setting the corresponding `keyring_oci` system variable.

For example, to load and configure `keyring_oci`, use these lines in the server `my.cnf` file (adjust the `.so` suffix and file location for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_oci.so
keyring_oci_user=ocid1.user.oc1..longAlphaNumericString
keyring_oci_tenancy=ocid1.tenancy.oc1..longAlphaNumericString
keyring_oci_compartment=ocid1.compartment.oc1..longAlphaNumericString
keyring_oci_virtual_vault=ocid1.vault.oc1.iad.shortAlphaNumericString.longAlphaNumericString
keyring_oci_master_key=ocid1.key.oc1.iad.shortAlphaNumericString.longAlphaNumericString
keyring_oci_encryption_endpoint=shortAlphaNumericString-crypto.kms.us-ashburn-1.oraclecloud.com
```

```
keyring_oci_management_endpoint=shortAlphaNumericString-management.kms.us-ashburn-1.oraclecloud.com
keyring_oci_vaults_endpoint=vaults.us-ashburn-1.oci.oraclecloud.com
keyring_oci_secrets_endpoint=secrets.vaults.us-ashburn-1.oci.oraclecloud.com
keyring_oci_key_file=file_name
keyring_oci_key_fingerprint=12:34:56:78:90:ab:cd:ef:12:34:56:78:90:ab:cd:ef
```

For additional information about the `keyring_oci` plugin-specific system variables, see Section 6.4.4.19, “Keyring System Variables”.

The `keyring_oci` plugin does not support runtime reconfiguration and none of its system variables can be modified at runtime. To change configuration parameters, do this:

- Modify parameter settings in the `my.cnf` file, or use `SET PERSIST_ONLY` for parameters that are persisted to `mysqld-auto.conf`.
- Restart the server.

6.4.4.13 Supported Keyring Key Types and Lengths

MySQL Keyring supports keys of different types (encryption algorithms) and lengths:

- The available key types depend on which keyring plugin is installed.
- The permitted key lengths are subject to multiple factors:
  - General keyring loadable-function interface limits (for keys managed using one of the keyring functions described in Section 6.4.4.15, “General-Purpose Keyring Key-Management Functions”), or limits from back end implementations. These length limits can vary by key operation type.
  - In addition to the general limits, individual keyring plugins may impose restrictions on key lengths per key type.

Table 6.32, “General Keyring Key Length Limits” shows the general key-length limits. (The lower limits for `keyring_aws` are imposed by the AWS KMS interface, not the keyring functions.) For keyring plugins, Table 6.33, “Keyring Plugin Key Types and Lengths” shows the key types each keyring plugin permits, as well as any plugin-specific key-length restrictions. For most keyring components, the general key-length limits apply and there are no key-type restrictions.

Note

`component_keyring_oci` (like the `keyring_oci` plugin) can only generate keys of type `AES` with a size of 16, 24, or 32 bytes.

Table 6.32 General Keyring Key Length Limits

Key Operation	Maximum Key Length
Generate key	16,384 bytes (2,048 prior to MySQL 8.0.18); 1,024 for `keyring_aws`
Store key	16,384 bytes (2,048 prior to MySQL 8.0.18); 4,096 for `keyring_aws`
Fetch key	16,384 bytes (2,048 prior to MySQL 8.0.18); 4,096 for `keyring_aws`

Table 6.33 Keyring Plugin Key Types and Lengths

Plugin Name	Permitted Key Type	Plugin-Specific Length Restrictions

`keyring_aws`	`AES`
	
	`SECRET`
		16, 24, or 32 bytes
		
		None

`keyring_encrypted_file`	`AES`	None

Plugin Name	Permitted Key Type	Plugin-Specific Length Restrictions
	DSA	None
	RSA	None
	SECRET	None
keyring_file	AES	None
	DSA	None
	RSA	None
	SECRET	None
keyring_hashicorp	AES	None
	DSA	None
	RSA	None
	SECRET	None
keyring_oci	AES	16, 24, or 32 bytes
keyring_okv	AES	16, 24, or 32 bytes
	SECRET	None

The SECRET key type, available as of MySQL 8.0.19, is intended for general-purpose storage of sensitive data using the MySQL keyring, and is supported by most keyring components and keyring plugins. The keyring encrypts and decrypts SECRET data as a byte stream upon storage and retrieval.

Example keyring operations involving the SECRET key type:

```
SELECT keyring_key_generate('MySecret1', 'SECRET', 20);
SELECT keyring_key_remove('MySecret1');

SELECT keyring_key_store('MySecret2', 'SECRET', 'MySecretData');
SELECT keyring_key_fetch('MySecret2');
SELECT keyring_key_length_fetch('MySecret2');
SELECT keyring_key_type_fetch('MySecret2');
SELECT keyring_key_remove('MySecret2');
```

6.4.4.14 Migrating Keys Between Keyring Keystores

A keyring migration copies keys from one keystore to another, enabling a DBA to switch a MySQL installation to a different keystore. A successful migration operation has this result:

- The destination keystore contains the keys it had prior to the migration, plus the keys from the source keystore.
- The source keystore remains the same before and after the migration (because keys are copied, not moved).

If a key to be copied already exists in the destination keystore, an error occurs and the destination keystore is restored to its premigration state.

The keyring manages keystores using keyring components and keyring plugins. This pertains to migration strategy because the way in which the source and destination keystores are managed determines whether a particular type of key migration is possible and the procedure for performing it:

- Migration from one keyring plugin to another: The MySQL server has an operational mode that provides this capability.

- Migration from a keyring plugin to a keyring component: The MySQL server has an operational mode that provides this capability as of MySQL 8.0.24.
- Migration from one keyring component to another: The `mysql_migrate_keyring` utility provides this capability. `mysql_migrate_keyring` is available as of MySQL 8.0.24.
- Migration from a keyring component to a keyring plugin: There is no provision for this capability.

The following sections discuss the characteristics of offline and online migrations and describe how to perform migrations.

- Offline and Online Key Migrations
- Key Migration Using a Migration Server
- Key Migration Using the mysql_migrate_keyring Utility
- Key Migration Involving Multiple Running Servers

Offline and Online Key Migrations

A key migration is either offline or online:

- Offline migration: For use when you are sure that no running server on the local host is using the source or destination keystore. In this case, the migration operation can copy keys from the source keystore to the destination without the possibility of a running server modifying keystore content during the operation.
- Online migration: For use when a running server on the local host is using the source keystore. In this case, care must be taken to prevent that server from updating keystores during the migration. This involves connecting to the running server and instructing it to pause keyring operations so that keys can be copied safely from the source keystore to the destination. When key copying is complete, the running server is permitted to resume keyring operations.

When you plan a key migration, use these points to decide whether it should be offline or online:

- Do not perform offline migration involving a keystore that is in use by a running server.
- Pausing keyring operations during an online migration is accomplished by connecting to the running server and setting its global `keyring_operations` system variable to `OFF` before key copying and `ON` after key copying. This has several implications:
  - `keyring_operations` was introduced in MySQL 5.7.21, so online migration is possible only if the running server is from MySQL 5.7.21 or higher. If the running server is older, you must stop it, perform an offline migration, and restart it. All migration instructions elsewhere that refer to `keyring_operations` are subject to this condition.
  - The account used to connect to the running server must have the privileges required to modify `keyring_operations`. These privileges are `ENCRYPTION_KEY_ADMIN` in addition to either `SYSTEM_VARIABLES_ADMIN` or the deprecated `SUPER` privilege.
  - If an online migration operation exits abnormally (for example, if it is forcibly terminated), it is possible for `keyring_operations` to remain disabled on the running server, leaving it unable to perform keyring operations. In this case, it may be necessary to connect to the running server and enable `keyring_operations` manually using this statement:

    ```
    SET GLOBAL keyring_operations = ON;
    ```
- Online key migration provides for pausing keyring operations on a single running server. To perform a migration if multiple running servers are using the keystores involved, use the procedure described at Key Migration Involving Multiple Running Servers.

Key Migration Using a Migration Server

Note

Online key migration using a migration server is only supported if the running server allows socket connections or TCP/IP connections using TLS; it is not supported when, for example, the server is running on a Windows platform and only allows shared memory connections.

A MySQL server becomes a migration server if invoked in a special operational mode that supports key migration. A migration server does not accept client connections. Instead, it runs only long enough to migrate keys, then exits. A migration server reports errors to the console (the standard error output).

A migration server supports these migration types:

- Migration from one keyring plugin to another.
- Migration from a keyring plugin to a keyring component. This capability is available as of MySQL 8.0.24. Older servers support only migration from one keyring plugin to another, in which case the parts of these instructions that refer to keyring components do not apply.

A migration server does not support migration from one keyring component to another. For that type of migration, see Key Migration Using the mysql_migrate_keyring Utility.

To perform a key migration operation using a migration server, determine the key migration options required to specify which keyring plugins or components are involved, and whether the migration is offline or online:

- To indicate the source keyring plugin and the destination keyring plugin or component, specify these options:
  - `--keyring-migration-source`: The source keyring plugin that manages the keys to be migrated.
  - `--keyring-migration-destination`: The destination keyring plugin or component to which the migrated keys are to be copied.
  - `--keyring-migration-to-component`: This option is required if the destination is a keyring component rather than a keyring plugin.

  The `--keyring-migration-source` and `--keyring-migration-destination` options signify to the server that it should run in key migration mode. For key migration operations, both options are mandatory. Each plugin or component is specified using the name of its library file, including any platform-specific extension such as `.so` or `.dll`. The source and destination must differ, and the migration server must support them both.
- For an offline migration, no additional key migration options are needed.
- For an online migration, some running server currently is using the source or destination keystore. To invoke the migration server, specify additional key migration options that indicate how to connect to the running server. This is necessary so that the migration server can connect to the running server and tell it to pause keyring use during the migration operation.

  Use of any of the following options signifies an online migration:
  - `--keyring-migration-host`: The host where the running server is located. This is always the local host because the migration server can migrate keys only between keystores managed by local plugins and components.
  - `--keyring-migration-user`, `--keyring-migration-password`: The account credentials to use to connect to the running server.

- `--keyring-migration-port`: For TCP/IP connections, the port number to connect to on the running server.
- `--keyring-migration-socket`: For Unix socket file or Windows named pipe connections, the socket file or named pipe to connect to on the running server.

For additional details about the key migration options, see Section 6.4.4.18, “Keyring Command Options”.

Start the migration server with key migration options indicating the source and destination keystores and whether the migration is offline or online, possibly with other options. Keep the following considerations in mind:

- Other server options might be required, such as configuration parameters for the two keyring plugins. For example, if `keyring_file` is the source or destination, you must set the `keyring_file_data` system variable if the keyring data file location is not the default location. Other non-keyring options may be required as well. One way to specify these options is by using `--defaults-file` to name an option file that contains the required options.
- The migration server expects path name option values to be full paths. Relative path names may not be resolved as you expect.
- The user who invokes a server in key-migration mode must not be the `root` operating system user, unless the `--user` option is specified with a non-`root` user name to run the server as that user.
- The user a server in key-migration mode runs as must have permission to read and write any local keyring files, such as the data file for a file-based plugin.

  If you invoke the migration server from a system account different from that normally used to run MySQL, it might create keyring directories or files that are inaccessible to the server during normal operation. Suppose that `mysqld` normally runs as the `mysql` operating system user, but you invoke the migration server while logged in as `isabel`. Any new directories or files created by the migration server are owned by `isabel`. Subsequent startup fails when a server run as the `mysql` operating system user attempts to access file system objects owned by `isabel`.

  To avoid this issue, start the migration server as the `root` operating system user and provide a `--user=`*`user_name`* option, where *`user_name`* is the system account normally used to run MySQL. Alternatively, after the migration, examine the keyring-related file system objects and change their ownership and permissions if necessary using `chown`, `chmod`, or similar commands, so that the objects are accessible to the running server.

Example command line for offline migration between two keyring plugins (enter the command on a single line):

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
  --keyring-migration-source=keyring_file.so
  --keyring-migration-destination=keyring_encrypted_file.so
  --keyring_encrypted_file_password=password
```

Example command line for online migration between two keyring plugins:

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
  --keyring-migration-source=keyring_file.so
  --keyring-migration-destination=keyring_encrypted_file.so
  --keyring_encrypted_file_password=password
  --keyring-migration-host=127.0.0.1
  --keyring-migration-user=root
  --keyring-migration-password=root_password
```

To perform a migration when the destination is a keyring component rather than a keyring plugin, specify the `--keyring-migration-to-component` option, and name the component as the value of the `--keyring-migration-destination` option.

Example command line for offline migration from a keyring plugin to a keyring component:

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
  --keyring-migration-to-component
  --keyring-migration-source=keyring_file.so
  --keyring-migration-destination=component_keyring_encrypted_file.so
```

Notice that in this case, no `keyring_encrypted_file_password` value is specified. The password for the component data file is listed in the component configuration file.

Example command line for online migration from a keyring plugin to a keyring component:

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
  --keyring-migration-to-component
  --keyring-migration-source=keyring_file.so
  --keyring-migration-destination=component_keyring_encrypted_file.so
  --keyring-migration-host=127.0.0.1
  --keyring-migration-user=root
  --keyring-migration-password=root_password
```

The key migration server performs a migration operation as follows:

1. (Online migration only) Connect to the running server using the connection options.
2. (Online migration only) Disable `keyring_operations` on the running server.
3. Load the keyring plugin/component libraries for the source and destination keystores.
4. Copy keys from the source keystore to the destination.
5. Unload the keyring plugin/component libraries for the source and destination keystores.
6. (Online migration only) Enable `keyring_operations` on the running server.
7. (Online migration only) Disconnect from the running server.

If an error occurs during key migration, the destination keystore is restored to its premigration state.

After a successful online key migration operation, the running server might need to be restarted:

- If the running server was using the source keystore before the migration and should continue to use it after the migration, it need not be restarted after the migration.
- If the running server was using the destination keystore before the migration and should continue to use it after the migration, it should be restarted after the migration to load all keys migrated into the destination keystore.
- If the running server was using the source keystore before the migration but should use the destination keystore after the migration, it must be reconfigured to use the destination keystore and restarted. In this case, be aware that although the running server is paused from modifying the source keystore during the migration itself, it is not paused during the interval between the migration and the subsequent restart. Care should be taken that the server does not modify the source keystore during this interval because any such changes will not be reflected in the destination keystore.

Key Migration Using the mysql_migrate_keyring Utility

The `mysql_migrate_keyring` utility migrates keys from one keyring component to another. It does not support migrations involving keyring plugins. For that type of migration, use a MySQL server operating in key migration mode; see Key Migration Using a Migration Server.

To perform a key migration operation using `mysql_migrate_keyring`, determine the key migration options required to specify which keyring components are involved, and whether the migration is offline or online:

- To indicate the source and destination keyring components and their location, specify these options:
  - `--source-keyring`: The source keyring component that manages the keys to be migrated.
  - `--destination-keyring`: The destination keyring component to which the migrated keys are to be copied.
  - `--component-dir`: The directory containing keyring component library files. This is typically the value of the `plugin_dir` system variable for the local MySQL server.

  All three options are mandatory. Each keyring component name is a component library file name specified without any platform-specific extension such as `.so` or `.dll`. For example, to use the component for which the library file is `component_keyring_file.so`, specify the option as `--source-keyring=component_keyring_file`. The source and destination must differ, and `mysql_migrate_keyring` must support them both.
- For an offline migration, no additional options are needed.
- For an online migration, some running server currently is using the source or destination keystore. In this case, specify the `--online-migration` option to signify an online migration. In addition, specify connection options indicating how to connect to the running server, so that `mysql_migrate_keyring` can connect to it and tell it to pause keyring use during the migration operation.

  The `--online-migration` option is commonly used in conjunction with connection options such as these:
  - `--host`: The host where the running server is located. This is always the local host because `mysql_migrate_keyring` can migrate keys only between keystores managed by local components.
  - `--user`, `--password`: The account credentials to use to connect to the running server.
  - `--port`: For TCP/IP connections, the port number to connect to on the running server.
  - `--socket`: For Unix socket file or Windows named pipe connections, the socket file or named pipe to connect to on the running server.

For descriptions of all available options, see Section 4.6.8, “mysql_migrate_keyring — Keyring Key Migration Utility”.

Start `mysql_migrate_keyring` with options indicating the source and destination keystores and whether the migration is offline or online, possibly with other options. Keep the following considerations in mind:

- The user who invokes `mysql_migrate_keyring` must not be the `root` operating system user.
- The user who invokes `mysql_migrate_keyring` must have permission to read and write any local keyring files, such as the data file for a file-based plugin.

  If you invoke `mysql_migrate_keyring` from a system account different from that normally used to run MySQL, it might create keyring directories or files that are inaccessible to the server during normal operation. Suppose that `mysqld` normally runs as the `mysql` operating system user, but you invoke `mysql_migrate_keyring` while logged in as `isabel`. Any new directories or files created by `mysql_migrate_keyring` are owned by `isabel`. Subsequent startup fails when a server run as the `mysql` operating system user attempts to access file system objects owned by `isabel`.

  To avoid this issue, invoke `mysql_migrate_keyring` as the `mysql` operating system user. Alternatively, after the migration, examine the keyring-related file system objects and change their ownership and permissions if necessary using `chown`, `chmod`, or similar commands, so that the objects are accessible to the running server.

Suppose that you want to migrate keys from `component_keyring_file` to `component_keyring_encrypted_file`, and that the local server stores its keyring component library files in `/usr/local/mysql/lib/plugin`.

If no running server is using the keyring, an offline migration is permitted. Invoke `mysql_migrate_keyring` like this (enter the command on a single line):

```
mysql_migrate_keyring
  --component-dir=/usr/local/mysql/lib/plugin
  --source-keyring=component_keyring_file
  --destination-keyring=component_keyring_encrypted_file
```

If a running server is using the keyring, you must perform an online migration instead. In this case, the `--online-migration` option must be given, along with any connection options required to specify which server to connect to and the MySQL account to use.

The following command performs an online migration. It connects to the local server using a TCP/IP connection and the `admin` account. The command prompts for a password, which you should enter when prompted:

```
mysql_migrate_keyring
  --component-dir=/usr/local/mysql/lib/plugin
  --source-keyring=component_keyring_file
  --destination-keyring=component_keyring_encrypted_file
  --online-migration --host=127.0.0.1 --user=admin --password
```

`mysql_migrate_keyring` performs a migration operation as follows:

1. (Online migration only) Connect to the running server using the connection options.
2. (Online migration only) Disable `keyring_operations` on the running server.
3. Load the keyring component libraries for the source and destination keystores.
4. Copy keys from the source keystore to the destination.
5. Unload the keyring component libraries for the source and destination keystores.
6. (Online migration only) Enable `keyring_operations` on the running server.
7. (Online migration only) Disconnect from the running server.

If an error occurs during key migration, the destination keystore is restored to its premigration state.

After a successful online key migration operation, the running server might need to be restarted:

- If the running server was using the source keystore before the migration and should continue to use it after the migration, it need not be restarted after the migration.
- If the running server was using the destination keystore before the migration and should continue to use it after the migration, it should be restarted after the migration to load all keys migrated into the destination keystore.
- If the running server was using the source keystore before the migration but should use the destination keystore after the migration, it must be reconfigured to use the destination keystore and restarted. In this case, be aware that although the running server is paused from modifying the source keystore during the migration itself, it is not paused during the interval between the migration and the subsequent restart. Care should be taken that the server does not modify the source keystore during this interval because any such changes will not be reflected in the destination keystore.

Key Migration Involving Multiple Running Servers

Online key migration provides for pausing keyring operations on a single running server. To perform a migration if multiple running servers are using the keystores involved, use this procedure:

1. Connect to each running server manually and set `keyring_operations=OFF`. This ensures that no running server is using the source or destination keystore and satisfies the required condition for offline migration.
2. Use a migration server or `mysql_migrate_keyring` to perform an offline key migration for each paused server.
3. Connect to each running server manually and set `keyring_operations=ON`.

All running servers must support the `keyring_operations` system variable. Any server that does not must be stopped before the migration and restarted after.

6.4.4.15 General-Purpose Keyring Key-Management Functions

MySQL Server supports a keyring service that enables internal components and plugins to securely store sensitive information for later retrieval.

MySQL Server also includes an SQL interface for keyring key management, implemented as a set of general-purpose functions that access the capabilities provided by the internal keyring service. The keyring functions are contained in a plugin library file, which also contains a `keyring_udf` plugin that must be enabled prior to function invocation. For these functions to be used, a keyring plugin such as `keyring_file` or `keyring_okv` must be enabled.

The functions described here are general purpose and intended for use with any keyring plugin. A given keyring plugin might have functions of its own that are intended for use only with that plugin; see Section 6.4.4.16, “Plugin-Specific Keyring Key-Management Functions”.

The following sections provide installation instructions for the keyring functions and demonstrate how to use them. For information about the keyring service functions invoked by these functions, see Section 5.6.9.2, “The Keyring Service”. For general keyring information, see Section 6.4.4, “The MySQL Keyring”.

- Installing or Uninstalling General-Purpose Keyring Functions
- Using General-Purpose Keyring Functions
- General-Purpose Keyring Function Reference

Installing or Uninstalling General-Purpose Keyring Functions

This section describes how to install or uninstall the keyring functions, which are implemented in a plugin library file that also contains a `keyring_udf` plugin. For general information about installing or uninstalling plugins and loadable functions, see Section 5.6.1, “Installing and Uninstalling Plugins”, and Section 5.7.1, “Installing and Uninstalling Loadable Functions”.

The keyring functions enable keyring key management operations, but the `keyring_udf` plugin must also be installed because the functions do not work correctly without it. Attempts to use the functions without the `keyring_udf` plugin result in an error.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `keyring_udf`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the `keyring_udf` plugin and the keyring functions, use the `INSTALL PLUGIN` and `CREATE FUNCTION` statements, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN keyring_udf SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_generate RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_length_fetch RETURNS INTEGER
```

```
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_type_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_store RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_remove RETURNS INTEGER
  SONAME 'keyring_udf.so';
```

If the plugin and functions are used on a source replication server, install them on all replicas as well to avoid replication issues.

Once installed as just described, the plugin and functions remain installed until uninstalled. To remove them, use the `UNINSTALL PLUGIN` and `DROP FUNCTION` statements:

```
UNINSTALL PLUGIN keyring_udf;
DROP FUNCTION keyring_key_generate;
DROP FUNCTION keyring_key_fetch;
DROP FUNCTION keyring_key_length_fetch;
DROP FUNCTION keyring_key_type_fetch;
DROP FUNCTION keyring_key_store;
DROP FUNCTION keyring_key_remove;
```

Using General-Purpose Keyring Functions

Before using the keyring general-purpose functions, install them according to the instructions provided in Installing or Uninstalling General-Purpose Keyring Functions.

The keyring functions are subject to these constraints:

- To use any keyring function, the `keyring_udf` plugin must be enabled. Otherwise, an error occurs:

  ```
  ERROR 1123 (HY000): Can't initialize function 'keyring_key_generate';
  This function requires keyring_udf plugin which is not installed.
  Please install
  ```

  To install the `keyring_udf` plugin, see Installing or Uninstalling General-Purpose Keyring Functions.

- The keyring functions invoke keyring service functions (see Section 5.6.9.2, “The Keyring Service”). The service functions in turn use whatever keyring plugin is installed (for example, `keyring_file` or `keyring_okv`). Therefore, to use any keyring function, some underlying keyring plugin must be enabled. Otherwise, an error occurs:

  ```
  ERROR 3188 (HY000): Function 'keyring_key_generate' failed because
  underlying keyring service returned an error. Please check if a
  keyring plugin is installed and that provided arguments are valid
  for the keyring you are using.
  ```

  To install a keyring plugin, see Section 6.4.4.3, “Keyring Plugin Installation”.

- A user must possess the global `EXECUTE` privilege to use any keyring function. Otherwise, an error occurs:

  ```
  ERROR 1123 (HY000): Can't initialize function 'keyring_key_generate';
  The user is not privileged to execute this function. User needs to
  have EXECUTE
  ```

  To grant the global `EXECUTE` privilege to a user, use this statement:

  ```
  GRANT EXECUTE ON *.* TO user;
  ```

  Alternatively, should you prefer to avoid granting the global `EXECUTE` privilege while still permitting users to access specific key-management operations, “wrapper” stored programs can be defined (a technique described later in this section).

- A key stored in the keyring by a given user can be manipulated later only by the same user. That is, the value of the `CURRENT_USER()` function at the time of key manipulation must have the same

value as when the key was stored in the keyring. (This constraint rules out the use of the keyring functions for manipulation of instance-wide keys, such as those created by `InnoDB` to support tablespace encryption.)

To enable multiple users to perform operations on the same key, “wrapper” stored programs can be defined (a technique described later in this section).

- Keyring functions support the key types and lengths supported by the underlying keyring plugin. For information about keys specific to a particular keyring plugin, see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

To create a new random key and store it in the keyring, call `keyring_key_generate()`, passing to it an ID for the key, along with the key type (encryption method) and its length in bytes. The following call creates a 2,048-bit DSA-encrypted key named `MyKey`:

```
mysql> SELECT keyring_key_generate('MyKey', 'DSA', 256);
+-----------------------------------------+
| keyring_key_generate('MyKey', 'DSA', 256) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

A return value of 1 indicates success. If the key cannot be created, the return value is `NULL` and an error occurs. One reason this might be is that the underlying keyring plugin does not support the specified combination of key type and key length; see Section 6.4.4.13, “Supported Keyring Key Types and Lengths”.

To be able to check the return type regardless of whether an error occurs, use `SELECT ... INTO` *`@var_name`* and test the variable value:

```
mysql> SELECT keyring_key_generate('', '', -1) INTO @x;
ERROR 3188 (HY000): Function 'keyring_key_generate' failed because
underlying keyring service returned an error. Please check if a
keyring plugin is installed and that provided arguments are valid
for the keyring you are using.
mysql> SELECT @x;
+------+
| @x   |
+------+
| NULL |
+------+
mysql> SELECT keyring_key_generate('x', 'AES', 16) INTO @x;
mysql> SELECT @x;
+------+
| @x   |
+------+
|    1 |
+------+
```

This technique also applies to other keyring functions that for failure return a value and an error.

The ID passed to `keyring_key_generate()` provides a means by which to refer to the key in subsequent functions calls. For example, use the key ID to retrieve its type as a string or its length in bytes as an integer:

```
mysql> SELECT keyring_key_type_fetch('MyKey');
+---------------------------------+
| keyring_key_type_fetch('MyKey') |
+---------------------------------+
| DSA                             |
+---------------------------------+
mysql> SELECT keyring_key_length_fetch('MyKey');
+-----------------------------------+
| keyring_key_length_fetch('MyKey') |
+-----------------------------------+
|                               256 |
+-----------------------------------+
```

To retrieve a key value, pass the key ID to `keyring_key_fetch()`. The following example uses `HEX()` to display the key value because it may contain nonprintable characters. The example also uses a short key for brevity, but be aware that longer keys provide better security:

```
mysql> SELECT keyring_key_generate('MyShortKey', 'DSA', 8);
+-------------------------------------------+
| keyring_key_generate('MyShortKey', 'DSA', 8) |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+
mysql> SELECT HEX(keyring_key_fetch('MyShortKey'));
+-------------------------------------+
| HEX(keyring_key_fetch('MyShortKey')) |
+-------------------------------------+
| 1DB3B0FC3328A24C                    |
+-------------------------------------+
```

Keyring functions treat key IDs, types, and values as binary strings, so comparisons are case-sensitive. For example, IDs of `MyKey` and `mykey` refer to different keys.

To remove a key, pass the key ID to `keyring_key_remove()`:

```
mysql> SELECT keyring_key_remove('MyKey');
+----------------------------+
| keyring_key_remove('MyKey') |
+----------------------------+
|                          1 |
+----------------------------+
```

To obfuscate and store a key that you provide, pass the key ID, type, and value to `keyring_key_store()`:

```
mysql> SELECT keyring_key_store('AES_key', 'AES', 'Secret string');
+-----------------------------------------------------+
| keyring_key_store('AES_key', 'AES', 'Secret string') |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
```

As indicated previously, a user must have the global `EXECUTE` privilege to call keyring functions, and the user who stores a key in the keyring initially must be the same user who performs subsequent operations on the key later, as determined from the `CURRENT_USER()` value in effect for each function call. To permit key operations to users who do not have the global `EXECUTE` privilege or who may not be the key “owner,” use this technique:

1. Define “wrapper” stored programs that encapsulate the required key operations and have a `DEFINER` value equal to the key owner.
2. Grant the `EXECUTE` privilege for specific stored programs to the individual users who should be able to invoke them.
3. If the operations implemented by the wrapper stored programs do not include key creation, create any necessary keys in advance, using the account named as the `DEFINER` in the stored program definitions.

This technique enables keys to be shared among users and provides to DBAs more fine-grained control over who can do what with keys, without having to grant global privileges.

The following example shows how to set up a shared key named `SharedKey` that is owned by the DBA, and a `get_shared_key()` stored function that provides access to the current key value. The value can be retrieved by any user with the `EXECUTE` privilege for that function, which is created in the `key_schema` schema.

From a MySQL administrative account (`'root'@'localhost'` in this example), create the administrative schema and the stored function to access the key:

```
mysql> CREATE SCHEMA key_schema;

mysql> CREATE DEFINER = 'root'@'localhost'
       FUNCTION key_schema.get_shared_key()
       RETURNS BLOB READS SQL DATA
       RETURN keyring_key_fetch('SharedKey');
```

From the administrative account, ensure that the shared key exists:

```
mysql> SELECT keyring_key_generate('SharedKey', 'DSA', 8);
+-------------------------------------------+
| keyring_key_generate('SharedKey', 'DSA', 8) |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+
```

From the administrative account, create an ordinary user account to which key access is to be granted:

```
mysql> CREATE USER 'key_user'@'localhost'
       IDENTIFIED BY 'key_user_pwd';
```

From the `key_user` account, verify that, without the proper `EXECUTE` privilege, the new account cannot access the shared key:

```
mysql> SELECT HEX(key_schema.get_shared_key());
ERROR 1370 (42000): execute command denied to user 'key_user'@'localhost'
for routine 'key_schema.get_shared_key'
```

From the administrative account, grant `EXECUTE` to `key_user` for the stored function:

```
mysql> GRANT EXECUTE ON FUNCTION key_schema.get_shared_key
       TO 'key_user'@'localhost';
```

From the `key_user` account, verify that the key is now accessible:

```
mysql> SELECT HEX(key_schema.get_shared_key());
+----------------------------------+
| HEX(key_schema.get_shared_key()) |
+----------------------------------+
| 9BAFB9E75CEEB013                 |
+----------------------------------+
```

General-Purpose Keyring Function Reference

For each general-purpose keyring function, this section describes its purpose, calling sequence, and return value. For information about the conditions under which these functions can be invoked, see Using General-Purpose Keyring Functions.

- `keyring_key_fetch(`*`key_id`*`)`

  Given a key ID, deobfuscates and returns the key value.

  Arguments:

  - *`key_id`*: A string that specifies the key ID.

  Return value:

  Returns the key value as a string for success, `NULL` if the key does not exist, or `NULL` and an error for failure.

Note

Key values retrieved using `keyring_key_fetch()` are subject to the general keyring function limits described in Section 6.4.4.13, “Supported Keyring Key Types and Lengths”. A key value longer than that length can be stored using a keyring service function (see Section 5.6.9.2, “The Keyring

> Service"), but if retrieved using `keyring_key_fetch()` is truncated to the general keyring function limit.

Example:

```
mysql> SELECT keyring_key_generate('RSA_key', 'RSA', 16);
+-----------------------------------------+
| keyring_key_generate('RSA_key', 'RSA', 16) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
mysql> SELECT HEX(keyring_key_fetch('RSA_key'));
+----------------------------------+
| HEX(keyring_key_fetch('RSA_key')) |
+----------------------------------+
| 91C2253B696064D3556984B6630F891A  |
+----------------------------------+
mysql> SELECT keyring_key_type_fetch('RSA_key');
+----------------------------------+
| keyring_key_type_fetch('RSA_key') |
+----------------------------------+
| RSA                              |
+----------------------------------+
mysql> SELECT keyring_key_length_fetch('RSA_key');
+-----------------------------------+
| keyring_key_length_fetch('RSA_key') |
+-----------------------------------+
|                                16 |
+-----------------------------------+
```

The example uses `HEX()` to display the key value because it may contain nonprintable characters. The example also uses a short key for brevity, but be aware that longer keys provide better security.

- `keyring_key_generate(`*`key_id`*`, `*`key_type`*`, `*`key_length`*`)`

  Generates a new random key with a given ID, type, and length, and stores it in the keyring. The type and length values must be consistent with the values supported by the underlying keyring plugin. See Section 6.4.4.13, "Supported Keyring Key Types and Lengths".

  Arguments:

  - *`key_id`*: A string that specifies the key ID.
  - *`key_type`*: A string that specifies the key type.
  - *`key_length`*: An integer that specifies the key length in bytes.

  Return value:

  Returns 1 for success, or `NULL` and an error for failure.

  Example:

```
mysql> SELECT keyring_key_generate('RSA_key', 'RSA', 384);
+-------------------------------------------+
| keyring_key_generate('RSA_key', 'RSA', 384) |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+
```

- `keyring_key_length_fetch(`*`key_id`*`)`

  Given a key ID, returns the key length.

  Arguments:

  - *`key_id`*: A string that specifies the key ID.

Return value:

Returns the key length in bytes as an integer for success, `NULL` if the key does not exist, or `NULL` and an error for failure.

Example:

See the description of `keyring_key_fetch()`.

- `keyring_key_remove(key_id)`

  Removes the key with a given ID from the keyring.

  Arguments:

  - *`key_id`*: A string that specifies the key ID.

  Return value:

  Returns 1 for success, or `NULL` for failure.

  Example:

```
mysql> SELECT keyring_key_remove('AES_key');
+------------------------------+
| keyring_key_remove('AES_key') |
+------------------------------+
|                            1 |
+------------------------------+
```

- `keyring_key_store(key_id, key_type, key)`

  Obfuscates and stores a key in the keyring.

  Arguments:

  - *`key_id`*: A string that specifies the key ID.
  - *`key_type`*: A string that specifies the key type.
  - *`key`*: A string that specifies the key value.

  Return value:

  Returns 1 for success, or `NULL` and an error for failure.

  Example:

```
mysql> SELECT keyring_key_store('new key', 'DSA', 'My key value');
+-----------------------------------------------------+
| keyring_key_store('new key', 'DSA', 'My key value') |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
```

- `keyring_key_type_fetch(key_id)`

  Given a key ID, returns the key type.

  Arguments:

  - *`key_id`*: A string that specifies the key ID.

  Return value:

  Returns the key type as a string for success, `NULL` if the key does not exist, or `NULL` and an error for failure.

  Example:

  See the description of `keyring_key_fetch()`.

6.4.4.16 Plugin-Specific Keyring Key-Management Functions

For each keyring plugin-specific function, this section describes its purpose, calling sequence, and return value. For information about general-purpose keyring functions, see Section 6.4.4.15, "General-Purpose Keyring Key-Management Functions".

- `keyring_aws_rotate_cmk()`

  Associated keyring plugin: `keyring_aws`

  `keyring_aws_rotate_cmk()` rotates the AWS KMS key. Rotation changes only the key that AWS KMS uses for subsequent data key-encryption operations. AWS KMS maintains previous CMK versions, so keys generated using previous CMKs remain decryptable after rotation.

  Rotation changes the CMK value used inside AWS KMS but does not change the ID used to refer to it, so there is no need to change the `keyring_aws_cmk_id` system variable after calling `keyring_aws_rotate_cmk()`.

  This function requires the `SUPER` privilege.

  Arguments:

  None.

  Return value:

  Returns 1 for success, or `NULL` and an error for failure.

- `keyring_aws_rotate_keys()`

  Associated keyring plugin: `keyring_aws`

  `keyring_aws_rotate_keys()` rotates keys stored in the `keyring_aws` storage file named by the `keyring_aws_data_file` system variable. Rotation sends each key stored in the file to AWS KMS for re-encryption using the value of the `keyring_aws_cmk_id` system variable as the CMK value, and stores the new encrypted keys in the file.

  `keyring_aws_rotate_keys()` is useful for key re-encryption under these circumstances:

  - After rotating the CMK; that is, after invoking the `keyring_aws_rotate_cmk()` function.
  - After changing the `keyring_aws_cmk_id` system variable to a different key value.

  This function requires the `SUPER` privilege.

Arguments:

None.

Return value:

Returns 1 for success, or `NULL` and an error for failure.

- `keyring_hashicorp_update_config()`

  Associated keyring plugin: `keyring_hashicorp`

  When invoked, the `keyring_hashicorp_update_config()` function causes `keyring_hashicorp` to perform a runtime reconfiguration, as described in keyring_hashicorp Configuration.

  This function requires the `SYSTEM_VARIABLES_ADMIN` privilege because it modifies global system variables.

  Arguments:

  None.

  Return value:

  Returns the string `'Configuration update was successful.'` for success, or `'Configuration update failed.'` for failure.

6.4.4.17 Keyring Metadata

This section describes sources of information about keyring use.

To see whether a keyring plugin is loaded, check the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

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

To see which keys exist, check the Performance Schema `keyring_keys` table:

```
mysql> SELECT * FROM performance_schema.keyring_keys;
+-----------------------------+--------------+----------------+
| KEY_ID                      | KEY_OWNER    | BACKEND_KEY_ID |
+-----------------------------+--------------+----------------+
| audit_log-20210322T130749-1 |              |                |
| MyKey                       | me@localhost |                |
| YourKey                     | me@localhost |                |
+-----------------------------+--------------+----------------+
```

To see whether a keyring component is loaded, check the Performance Schema `keyring_component_status` table. For example:

```
mysql> SELECT * FROM performance_schema.keyring_component_status;
+--------------------+-------------------------------------------------+
| STATUS_KEY         | STATUS_VALUE                                    |
+--------------------+-------------------------------------------------+
| Component_name     | component_keyring_file                          |
| Author             | Oracle Corporation                              |
```

```
| License             | GPL                                                |
| Implementation_name | component_keyring_file                             |
| Version             | 1.0                                                |
| Component_status    | Active                                             |
| Data_file           | /usr/local/mysql/keyring/component_keyring_file |
| Read_only           | No                                                 |
+---------------------+----------------------------------------------------+
```

A `Component_status` value of `Active` indicates that the component initialized successfully. If the component loaded but failed to initialize, the value is `Disabled`.

6.4.4.18 Keyring Command Options

MySQL supports the following keyring-related command-line options:

- `--keyring-migration-destination=`*`plugin`*

Command-Line Format	`--keyring-migration-destination=plugin_name`
Type	String

The destination keyring plugin for key migration. See Section 6.4.4.14, “Migrating Keys Between Keyring Keystores”. The option value interpretation depends on whether `--keyring-migration-to-component` is specified:

- If no, the option value is a keyring plugin, interpreted the same way as for `--keyring-migration-source`.
- If yes, the option value is a keyring component, specified as the component library name in the plugin directory, including any platform-specific extension such as `.so` or `.dll`.

Note

`--keyring-migration-source` and `--keyring-migration-destination` are mandatory for all keyring migration operations. The source and destination plugins must differ, and the migration server must support both plugins.

- `--keyring-migration-host=`*`host_name`*

Command-Line Format	`--keyring-migration-host=host_name`
Type	String
Default Value	`localhost`

The host location of the running server that is currently using one of the key migration keystores. See Section 6.4.4.14, “Migrating Keys Between Keyring Keystores”. Migration always occurs on the local host, so the option always specifies a value for connecting to a local server, such as `localhost`, `127.0.0.1`, `::1`, or the local host IP address or host name.

- `--keyring-migration-password[=`*`password`*`]`

Command-Line Format	`--keyring-migration-password[=password]`
Type	String

The password of the MySQL account used for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.4.14, “Migrating Keys Between Keyring Keystores”.

The password value is optional. If not given, the server prompts for one. If given, there must be *no space* between `--keyring-migration-password=` and the password following it. If no password option is specified, the default is to send no password.

Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, "End-User Guidelines for Password Security". You can use an option file to avoid giving the password on the command line. In this case, the file should have a restrictive mode and be accessible only to the account used to run the migration server.

- `--keyring-migration-port=port_num`

Command-Line Format	`--keyring-migration-port=port_num`
Type	Numeric
Default Value	`3306`

For TCP/IP connections, the port number for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.4.14, "Migrating Keys Between Keyring Keystores".

- `--keyring-migration-socket=path`

Command-Line Format	`--keyring-migration-socket={file_name|pipe_name}`
Type	String

For Unix socket file or Windows named pipe connections, the socket file or named pipe for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.4.14, "Migrating Keys Between Keyring Keystores".

- `--keyring-migration-source=plugin`

Command-Line Format	`--keyring-migration-source=plugin_name`
Type	String

The source keyring plugin for key migration. See Section 6.4.4.14, "Migrating Keys Between Keyring Keystores".

The option value is similar to that for `--plugin-load`, except that only one plugin library can be specified. The value is given as *`plugin_library`* or *`name=plugin_library`*, where *`plugin_library`* is the name of a library file that contains plugin code, and *`name`* is the name of a plugin to load. If a plugin library is named without any preceding plugin name, the server loads all plugins in the library. With a preceding plugin name, the server loads only the named plugin from the library. The server looks for plugin library files in the directory named by the `plugin_dir` system variable.

Note

`--keyring-migration-source` and `--keyring-migration-destination` are mandatory for all keyring migration operations. The source and destination plugins must differ, and the migration server must support both plugins.

- `--keyring-migration-to-component`

Command-Line Format	`--keyring-migration-to-component[={OFF|ON}]`

Introduced	8.0.24
Type	Boolean
Default Value	OFF

Indicates that a key migration is from a keyring plugin to a keyring component. This option makes it possible to migrate keys from any keyring plugin to any keyring component, which facilitates transitioning a MySQL installation from keyring plugins to keyring components.

For key migration from one keyring component to another, use the `mysql_migrate_keyring` utility. Migration from a keyring component to a keyring plugin is not supported. See Section 6.4.4.14, "Migrating Keys Between Keyring Keystores".

- `--keyring-migration-user=`*`user_name`*

Command-Line Format	`--keyring-migration-user=user_name`
Type	String

The user name of the MySQL account used for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.4.14, "Migrating Keys Between Keyring Keystores".

6.4.4.19 Keyring System Variables

MySQL Keyring plugins support the following system variables. Use them to configure keyring plugin operation. These variables are unavailable unless the appropriate keyring plugin is installed (see Section 6.4.4.3, "Keyring Plugin Installation").

- `keyring_aws_cmk_id`

Command-Line Format	`--keyring-aws-cmk-id=value`
System Variable	`keyring_aws_cmk_id`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

The KMS key ID obtained from the AWS KMS server and used by the `keyring_aws` plugin. This variable is unavailable unless that plugin is installed.

This variable is mandatory. If not specified, `keyring_aws` initialization fails.

- `keyring_aws_conf_file`

Command-Line Format	`--keyring-aws-conf-file=file_name`
System Variable	`keyring_aws_conf_file`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	File name
Default Value	`platform specific`

The location of the configuration file for the `keyring_aws` plugin. This variable is unavailable unless that plugin is installed.

At plugin startup, `keyring_aws` reads the AWS secret access key ID and key from the configuration file. For the `keyring_aws` plugin to start successfully, the configuration file must exist and contain valid secret access key information, initialized as described in Section 6.4.4.9, “Using the keyring_aws Amazon Web Services Keyring Plugin”.

The default file name is `keyring_aws_conf`, located in the default keyring file directory. The location of this default directory is the same as for the `keyring_file_data` system variable. See the description of that variable for details, as well as for considerations to take into account if you create the directory manually.

- `keyring_aws_data_file`

Command-Line Format	`--keyring-aws-data-file`
System Variable	`keyring_aws_data_file`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	File name
Default Value	`platform specific`

The location of the storage file for the `keyring_aws` plugin. This variable is unavailable unless that plugin is installed.

At plugin startup, if the value assigned to `keyring_aws_data_file` specifies a file that does not exist, the `keyring_aws` plugin attempts to create it (as well as its parent directory, if necessary). If the file does exist, `keyring_aws` reads any encrypted keys contained in the file into its in-memory cache. `keyring_aws` does not cache unencrypted keys in memory.

The default file name is `keyring_aws_data`, located in the default keyring file directory. The location of this default directory is the same as for the `keyring_file_data` system variable. See the description of that variable for details, as well as for considerations to take into account if you create the directory manually.

- `keyring_aws_region`

Command-Line Format	`--keyring-aws-region=value`
System Variable	`keyring_aws_region`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`us-east-1`

Valid Values (≥ 8.0.30)	`af-south-1`
	`ap-east-1`
	`ap-northeast-1`
	`ap-northeast-2`
	`ap-northeast-3`
	`ap-south-1`

	ap-southeast-1 ap-southeast-2 ca-central-1 cn-north-1 cn-northwest-1 eu-central-1 eu-north-1 eu-south-1 eu-west-1 eu-west-2 eu-west-3 me-south-1 sa-east-1 us-east-1 us-east-2 us-gov-east-1 us-iso-east-1 us-iso-west-1 us-isob-east-1 us-west-1 us-west-2
Valid Values (≥ 8.0.17, ≤ 8.0.29)	ap-northeast-1 ap-northeast-2 ap-south-1 ap-southeast-1 ap-southeast-2 ca-central-1 cn-north-1 cn-northwest-1 eu-central-1 eu-west-1

	eu-west-2 eu-west-3 sa-east-1 us-east-1 us-east-2 us-west-1 us-west-2
Valid Values (≤ 8.0.16)	ap-northeast-1 ap-northeast-2 ap-south-1 ap-southeast-1 ap-southeast-2 eu-central-1 eu-west-1 sa-east-1 us-east-1 us-west-1 us-west-2

The AWS region for the keyring_aws plugin. This variable is unavailable unless that plugin is installed.

- keyring_encrypted_file_data

Command-Line Format	--keyring-encrypted-file-data=file_name
System Variable	keyring_encrypted_file_data
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	platform specific

The path name of the data file used for secure data storage by the keyring_encrypted_file plugin. This variable is unavailable unless that plugin is installed. The file location should be in a directory considered for use only by keyring plugins. For example, do not locate the file under the data directory.

Keyring operations are transactional: The keyring_encrypted_file plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The

backup file has the same name as the value of the keyring_encrypted_file_data system variable with a suffix of .backup.

Do not use the same keyring_encrypted_file data file for multiple MySQL instances. Each instance should have its own unique data file.

The default file name is keyring_encrypted, located in a directory that is platform specific and depends on the value of the INSTALL_LAYOUT CMake option, as shown in the following table. To specify the default directory for the file explicitly if you are building from source, use the INSTALL_MYSQLKEYRINGDIR CMake option.

INSTALL_LAYOUT Value	Default keyring_encrypted_file_data Value
DEB, RPM, SVR4	/var/lib/mysql-keyring/keyring_encrypted
Otherwise	keyring/keyring_encrypted under the CMAKE_INSTALL_PREFIX value

At plugin startup, if the value assigned to keyring_encrypted_file_data specifies a file that does not exist, the keyring_encrypted_file plugin attempts to create it (as well as its parent directory, if necessary).

If you create the directory manually, it should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the /usr/local/mysql/mysql-keyring directory, the following commands (executed as root) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring
chmod 750 mysql-keyring
chown mysql mysql-keyring
chgrp mysql mysql-keyring
```

If the keyring_encrypted_file plugin cannot create or access its data file, it writes an error message to the error log. If an attempted runtime assignment to keyring_encrypted_file_data results in an error, the variable value remains unchanged.

Important

Once the keyring_encrypted_file plugin has created its data file and started to use it, it is important not to remove the file. Loss of the file causes data encrypted using its keys to become inaccessible. (It is permissible to rename or move the file, as long as you change the value of keyring_encrypted_file_data to match.)

- keyring_encrypted_file_password

Command-Line Format	--keyring-encrypted-file-password=password
System Variable	keyring_encrypted_file_password
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Type	String

The password used by the `keyring_encrypted_file` plugin. This variable is unavailable unless that plugin is installed.

This variable is mandatory. If not specified, `keyring_encrypted_file` initialization fails.

If this variable is specified in an option file, the file should have a restrictive mode and be accessible only to the account used to run the MySQL server.

Important

Once the `keyring_encrypted_file_password` value has been set, changing it does not rotate the keyring password and could make the server inaccessible. If an incorrect password is provided, the `keyring_encrypted_file` plugin cannot load keys from the encrypted keyring file.

The password value cannot be displayed at runtime with `SHOW VARIABLES` or the Performance Schema `global_variables` table because the display value is obfuscated.

- `keyring_file_data`

Command-Line Format	`--keyring-file-data=file_name`
System Variable	`keyring_file_data`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	File name
Default Value	`platform specific`

The path name of the data file used for secure data storage by the `keyring_file` plugin. This variable is unavailable unless that plugin is installed. The file location should be in a directory considered for use only by keyring plugins. For example, do not locate the file under the data directory.

Keyring operations are transactional: The `keyring_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_file_data` system variable with a suffix of `.backup`.

Do not use the same `keyring_file` data file for multiple MySQL instances. Each instance should have its own unique data file.

The default file name is `keyring`, located in a directory that is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option, as shown in the following table. To specify the default directory for the file explicitly if you are building from source, use the `INSTALL_MYSQLKEYRINGDIR` `CMake` option.

`INSTALL_LAYOUT` Value	**Default `keyring_file_data` Value**
`DEB`, `RPM`, `SVR4`	`/var/lib/mysql-keyring/keyring`

INSTALL_LAYOUT Value	Default keyring_file_data Value
Otherwise	keyring/keyring under the CMAKE_INSTALL_PREFIX value

At plugin startup, if the value assigned to keyring_file_data specifies a file that does not exist, the keyring_file plugin attempts to create it (as well as its parent directory, if necessary).

If you create the directory manually, it should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the /usr/local/mysql/mysql-keyring directory, the following commands (executed as root) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring
chmod 750 mysql-keyring
chown mysql mysql-keyring
chgrp mysql mysql-keyring
```

If the keyring_file plugin cannot create or access its data file, it writes an error message to the error log. If an attempted runtime assignment to keyring_file_data results in an error, the variable value remains unchanged.

Important

Once the keyring_file plugin has created its data file and started to use it, it is important not to remove the file. For example, InnoDB uses the file to store the master key used to decrypt the data in tables that use InnoDB tablespace encryption; see Section 15.13, “InnoDB Data-at-Rest Encryption”. Loss of the file causes data in such tables to become inaccessible. (It is permissible to rename or move the file, as long as you change the value of keyring_file_data to match.) It is recommended that you create a separate backup of the keyring data file immediately after you create the first encrypted table and before and after master key rotation.

- keyring_hashicorp_auth_path

Command-Line Format	--keyring-hashicorp-auth-path=value
Introduced	8.0.18
System Variable	keyring_hashicorp_auth_path
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	/v1/auth/approle/login

The authentication path where AppRole authentication is enabled within the HashiCorp Vault server, for use by the keyring_hashicorp plugin. This variable is unavailable unless that plugin is installed.

- keyring_hashicorp_ca_path

Command-Line Format	--keyring-hashicorp-ca-path=file_name
Introduced	8.0.18
System Variable	keyring_hashicorp_ca_path
Scope	Global

Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	empty string

The absolute path name of a local file accessible to the MySQL server that contains a properly formatted TLS certificate authority for use by the keyring_hashicorp plugin. This variable is unavailable unless that plugin is installed.

If this variable is not set, the keyring_hashicorp plugin opens an HTTPS connection without using server certificate verification, and trusts any certificate delivered by the HashiCorp Vault server. For this to be safe, it must be assumed that the Vault server is not malicious and that no man-in-the-middle attack is possible. If those assumptions are invalid, set keyring_hashicorp_ca_path to the path of a trusted CA certificate. (For example, for the instructions in Certificate and Key Preparation, this is the company.crt file.)

- keyring_hashicorp_caching

Command-Line Format	--keyring-hashicorp-caching[={OFF|ON}]
Introduced	8.0.18
System Variable	keyring_hashicorp_caching
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Whether to enable the optional in-memory key cache used by the keyring_hashicorp plugin to cache keys from the HashiCorp Vault server. This variable is unavailable unless that plugin is installed. If the cache is enabled, the plugin populates it during initialization. Otherwise, the plugin populates only the key list during initialization.

Enabling the cache is a compromise: It improves performance, but maintains a copy of sensitive key information in memory, which may be undesirable for security purposes.

- keyring_hashicorp_commit_auth_path

Introduced	8.0.18
System Variable	keyring_hashicorp_commit_auth_path
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

This variable is associated with keyring_hashicorp_auth_path, from which it takes its value during keyring_hashicorp plugin initialization. This variable is unavailable unless that plugin is installed. It reflects the “committed” value actually used for plugin operation if initialization succeeds. For additional information, see keyring_hashicorp Configuration.

- keyring_hashicorp_commit_ca_path

Introduced	8.0.18

System Variable	keyring_hashicorp_commit_ca_path
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

This variable is associated with keyring_hashicorp_ca_path, from which it takes its value during keyring_hashicorp plugin initialization. This variable is unavailable unless that plugin is installed. It reflects the “committed” value actually used for plugin operation if initialization succeeds. For additional information, see keyring_hashicorp Configuration.

- keyring_hashicorp_commit_caching

Introduced	8.0.18
System Variable	keyring_hashicorp_commit_caching
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

This variable is associated with keyring_hashicorp_caching, from which it takes its value during keyring_hashicorp plugin initialization. This variable is unavailable unless that plugin is installed. It reflects the “committed” value actually used for plugin operation if initialization succeeds. For additional information, see keyring_hashicorp Configuration.

- keyring_hashicorp_commit_role_id

Introduced	8.0.18
System Variable	keyring_hashicorp_commit_role_id
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

This variable is associated with keyring_hashicorp_role_id, from which it takes its value during keyring_hashicorp plugin initialization. This variable is unavailable unless that plugin is installed. It reflects the “committed” value actually used for plugin operation if initialization succeeds. For additional information, see keyring_hashicorp Configuration.

- keyring_hashicorp_commit_server_url

Introduced	8.0.18
System Variable	keyring_hashicorp_commit_server_url
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

This variable is associated with keyring_hashicorp_server_url, from which it takes its value during keyring_hashicorp plugin initialization. This variable is unavailable unless that plugin is

installed. It reflects the “committed” value actually used for plugin operation if initialization succeeds. For additional information, see keyring_hashicorp Configuration.

- `keyring_hashicorp_commit_store_path`

Introduced	8.0.18
System Variable	`keyring_hashicorp_commit_store_path`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String

This variable is associated with `keyring_hashicorp_store_path`, from which it takes its value during `keyring_hashicorp` plugin initialization. This variable is unavailable unless that plugin is installed. It reflects the “committed” value actually used for plugin operation if initialization succeeds. For additional information, see keyring_hashicorp Configuration.

- `keyring_hashicorp_role_id`

Command-Line Format	`--keyring-hashicorp-role-id=value`
Introduced	8.0.18
System Variable	`keyring_hashicorp_role_id`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`empty string`

The HashiCorp Vault AppRole authentication role ID, for use by the `keyring_hashicorp` plugin. This variable is unavailable unless that plugin is installed. The value must be in UUID format.

This variable is mandatory. If not specified, `keyring_hashicorp` initialization fails.

- `keyring_hashicorp_secret_id`

Command-Line Format	`--keyring-hashicorp-secret-id=value`
Introduced	8.0.18
System Variable	`keyring_hashicorp_secret_id`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`empty string`

The HashiCorp Vault AppRole authentication secret ID, for use by the `keyring_hashicorp` plugin. This variable is unavailable unless that plugin is installed. The value must be in UUID format.

This variable is mandatory. If not specified, `keyring_hashicorp` initialization fails.

The value of this variable is sensitive, so its value is masked by `*` characters when displayed.

- `keyring_hashicorp_server_url`

Command-Line Format	`--keyring-hashicorp-server-url=value`
Introduced	8.0.18
System Variable	`keyring_hashicorp_server_url`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`https://127.0.0.1:8200`

The HashiCorp Vault server URL, for use by the `keyring_hashicorp` plugin. This variable is unavailable unless that plugin is installed. The value must begin with `https://`.

- `keyring_hashicorp_store_path`

Command-Line Format	`--keyring-hashicorp-store-path=value`
Introduced	8.0.18
System Variable	`keyring_hashicorp_store_path`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String
Default Value	`empty string`

A store path within the HashiCorp Vault server that is writeable when appropriate AppRole credentials are provided by the `keyring_hashicorp` plugin. This variable is unavailable unless that plugin is installed. To specify the credentials, set the `keyring_hashicorp_role_id` and `keyring_hashicorp_secret_id` system variables (for example, as shown in keyring_hashicorp Configuration).

This variable is mandatory. If not specified, `keyring_hashicorp` initialization fails.

- `keyring_oci_ca_certificate`

Command-Line Format	`--keyring-oci-ca-certificate=file_name`
Introduced	8.0.22
System Variable	`keyring_oci_ca_certificate`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String
Default Value	`empty string`

The path name of the CA certificate bundle file that the `keyring_oci` plugin uses for Oracle Cloud Infrastructure certificate verification. This variable is unavailable unless that plugin is installed.

The file contains one or more certificates for peer verification. If no file is specified, the default CA bundle installed on the system is used. If the value is `disabled` (case-sensitive), `keyring_oci` performs no certificate verification.

- `keyring_oci_compartment`

Command-Line Format	`--keyring-oci-compartment=ocid`
Introduced	8.0.22
System Variable	`keyring_oci_compartment`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String

The OCID of the tenancy compartment that the `keyring_oci` plugin uses as the location of the MySQL keys. This variable is unavailable unless that plugin is installed.

Prior to using `keyring_oci`, you must create a MySQL compartment or subcompartment if it does not exist. This compartment should contain no vault keys or vault secrets. It should not be used by systems other than MySQL Keyring.

For information about managing compartments and obtaining the OCID, see Managing Compartments.

This variable is mandatory. If not specified, `keyring_oci` initialization fails.

- `keyring_oci_encryption_endpoint`

Command-Line Format	`--keyring-oci-encryption-endpoint=value`
Introduced	8.0.22
System Variable	`keyring_oci_encryption_endpoint`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String

The endpoint of the Oracle Cloud Infrastructure encryption server that the `keyring_oci` plugin uses for generating ciphertext for new keys. This variable is unavailable unless that plugin is installed.

The encryption endpoint is vault specific and Oracle Cloud Infrastructure assigns it at vault-creation time. To obtain the endpoint OCID, view the configuration details for your `keyring_oci` vault, using the instructions at Managing Vaults.

This variable is mandatory. If not specified, `keyring_oci` initialization fails.

- `keyring_oci_key_file`

Command-Line Format	`--keyring-oci-key-file=file_name`
Introduced	8.0.22
System Variable	`keyring_oci_key_file`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No

Type	String

The path name of the file containing the RSA private key that the `keyring_oci` plugin uses for Oracle Cloud Infrastructure authentication. This variable is unavailable unless that plugin is installed.

You must also upload the corresponding RSA public key using the Console. The Console displays the key fingerprint value, which you can use to set the `keyring_oci_key_fingerprint` system variable.

For information about generating and uploading API keys, see Required Keys and OCIDs.

This variable is mandatory. If not specified, `keyring_oci` initialization fails.

- `keyring_oci_key_fingerprint`

Command-Line Format	`--keyring-oci-key-fingerprint=value`
Introduced	8.0.22
System Variable	`keyring_oci_key_fingerprint`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The fingerprint of the RSA private key that the `keyring_oci` plugin uses for Oracle Cloud Infrastructure authentication. This variable is unavailable unless that plugin is installed.

To obtain the key fingerprint while creating the API keys, execute this command:

```
openssl rsa -pubout -outform DER -in ~/.oci/oci_api_key.pem | openssl md5 -c
```

Alternatively, obtain the fingerprint from the Console, which automatically displays the fingerprint when you upload the RSA public key.

For information about obtaining key fingerprints, see Required Keys and OCIDs.

This variable is mandatory. If not specified, `keyring_oci` initialization fails.

- `keyring_oci_management_endpoint`

Command-Line Format	`--keyring-oci-management-endpoint=value`
Introduced	8.0.22
System Variable	`keyring_oci_management_endpoint`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Type	String

The endpoint of the Oracle Cloud Infrastructure key management server that the `keyring_oci` plugin uses for listing existing keys. This variable is unavailable unless that plugin is installed.

The key management endpoint is vault specific and Oracle Cloud Infrastructure assigns it at vault-creation time. To obtain the endpoint OCID, view the configuration details for your `keyring_oci` vault, using the instructions at Managing Vaults.

This variable is mandatory. If not specified, `keyring_oci` initialization fails.

- `keyring_oci_master_key`

Command-Line Format	`--keyring-oci-master-key=ocid`
Introduced	8.0.22
System Variable	`keyring_oci_master_key`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String

The OCID of the Oracle Cloud Infrastructure master encryption key that the `keyring_oci` plugin uses for encryption of secrets. This variable is unavailable unless that plugin is installed.

Prior to using `keyring_oci`, you must create a cryptographic key for the Oracle Cloud Infrastructure compartment if it does not exist. Provide a MySQL-specific name for the generated key, and do not use it for other purposes.

For information about key creation, see Managing Keys.

This variable is mandatory. If not specified, `keyring_oci` initialization fails.

- `keyring_oci_secrets_endpoint`

Command-Line Format	`--keyring-oci-secrets-endpoint=value`
Introduced	8.0.22
System Variable	`keyring_oci_secrets_endpoint`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String

The endpoint of the Oracle Cloud Infrastructure secrets server that the `keyring_oci` plugin uses for listing, creating, and retiring secrets. This variable is unavailable unless that plugin is installed.

The secrets endpoint is vault specific and Oracle Cloud Infrastructure assigns it at vault-creation time. To obtain the endpoint OCID, view the configuration details for your `keyring_oci` vault, using the instructions at Managing Vaults.

This variable is mandatory. If not specified, `keyring_oci` initialization fails.

- `keyring_oci_tenancy`

Command-Line Format	`--keyring-oci-tenancy=ocid`

Introduced	8.0.22
System Variable	keyring_oci_tenancy
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The OCID of the Oracle Cloud Infrastructure tenancy that the keyring_oci plugin uses as the location of the MySQL compartment. This variable is unavailable unless that plugin is installed.

Prior to using keyring_oci, you must create a tenancy if it does not exist. To obtain the tenancy OCID from the Console, use the instructions at Required Keys and OCIDs.

This variable is mandatory. If not specified, keyring_oci initialization fails.

- keyring_oci_user

Command-Line Format	--keyring-oci-user=ocid
Introduced	8.0.22
System Variable	keyring_oci_user
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The OCID of the Oracle Cloud Infrastructure user that the keyring_oci plugin uses for cloud connections. This variable is unavailable unless that plugin is installed.

Prior to using keyring_oci, this user must exist and be granted access to use the configured Oracle Cloud Infrastructure tenancy, compartment, and vault resources.

To obtain the user OCID from the Console, use the instructions at Required Keys and OCIDs.

This variable is mandatory. If not specified, keyring_oci initialization fails.

- keyring_oci_vaults_endpoint

Command-Line Format	--keyring-oci-vaults-endpoint=value
Introduced	8.0.22
System Variable	keyring_oci_vaults_endpoint
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The endpoint of the Oracle Cloud Infrastructure vaults server that the keyring_oci plugin uses for obtaining the value of secrets. This variable is unavailable unless that plugin is installed.

The vaults endpoint is vault specific and Oracle Cloud Infrastructure assigns it at vault-creation time. To obtain the endpoint OCID, view the configuration details for your keyring_oci vault, using the instructions at Managing Vaults.

This variable is mandatory. If not specified, keyring_oci initialization fails.

- `keyring_oci_virtual_vault`

Command-Line Format	`--keyring-oci-virtual-vault=ocid`
Introduced	8.0.22
System Variable	`keyring_oci_virtual_vault`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String

The OCID of the Oracle Cloud Infrastructure Vault that the `keyring_oci` plugin uses for encryption operations. This variable is unavailable unless that plugin is installed.

Prior to using `keyring_oci`, you must create a new vault in the MySQL compartment if it does not exist. (Alternatively, you can reuse an existing vault that is in a parent compartment of the MySQL compartment.) Compartment users can see and use only the keys in their respective compartments.

For information about creating a vault and obtaining the vault OCID, see Managing Vaults.

This variable is mandatory. If not specified, `keyring_oci` initialization fails.

- `keyring_okv_conf_dir`

Command-Line Format	`--keyring-okv-conf-dir=dir_name`
System Variable	`keyring_okv_conf_dir`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Directory name
Default Value	`empty string`

The path name of the directory that stores configuration information used by the `keyring_okv` plugin. This variable is unavailable unless that plugin is installed. The location should be a directory considered for use only by the `keyring_okv` plugin. For example, do not locate the directory under the data directory.

The default `keyring_okv_conf_dir` value is empty. For the `keyring_okv` plugin to be able to access Oracle Key Vault, the value must be set to a directory that contains Oracle Key Vault configuration and SSL materials. For instructions on setting up this directory, see Section 6.4.4.8, “Using the keyring_okv KMIP Plugin”.

The directory should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the `/usr/local/mysql/mysql-keyring-okv` directory, the following commands (executed as `root`) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring-okv
chmod 750 mysql-keyring-okv
chown mysql mysql-keyring-okv
chgrp mysql mysql-keyring-okv
```

If the value assigned to `keyring_okv_conf_dir` specifies a directory that does not exist, or that does not contain configuration information that enables a connection to Oracle Key Vault to be established, `keyring_okv` writes an error message to the error log. If an attempted runtime

assignment to `keyring_okv_conf_dir` results in an error, the variable value and keyring operation remain unchanged.

- `keyring_operations`

System Variable	`keyring_operations`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether keyring operations are enabled. This variable is used during key migration operations. See Section 6.4.4.14, “Migrating Keys Between Keyring Keystores”. The privileges required to modify this variable are `ENCRYPTION_KEY_ADMIN` in addition to either `SYSTEM_VARIABLES_ADMIN` or the deprecated `SUPER` privilege.

6.4.5 MySQL Enterprise Audit

Note

MySQL Enterprise Audit is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin named `audit_log`. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring, logging, and blocking of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables 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. From MySQL 8.0.30, you can add statistics for the time and size of each query to detect outliers.

After you install the audit plugin (see Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”), it writes an audit log file. By default, the file is named `audit.log` in the server data directory. To change the name of the file, set the `audit_log_file` system variable at server startup.

By default, audit log file contents are written in new-style XML format, without compression or encryption. To select the file format, set the `audit_log_format` system variable at server startup. For details on file format and contents, see Section 6.4.5.4, “Audit Log File Formats”.

For more information about controlling how logging occurs, including audit log file naming and format selection, see Section 6.4.5.5, “Configuring Audit Logging Characteristics”. To perform filtering of audited events, see Section 6.4.5.7, “Audit Log Filtering”. For descriptions of the parameters used to configure the audit log plugin, see Audit Log Options and Variables.

If the audit log plugin is enabled, the Performance Schema (see Chapter 27, *MySQL Performance Schema*) has instrumentation for it. To identify the relevant instruments, use this query:

```
SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE '%/alog/%';
```

6.4.5.1 Elements of MySQL Enterprise Audit

MySQL Enterprise Audit is based on the audit log plugin and related elements:

- A server-side plugin named `audit_log` examines auditable events and determines whether to write them to the audit log.
- A set of functions enables manipulation of filtering definitions that control logging behavior, the encryption password, and log file reading.
- Tables in the `mysql` system database provide persistent storage of filter and user account data.
- System variables enable audit log configuration and status variables provide runtime operational information.
- The `AUDIT_ADMIN` privilege enable users to administer the audit log, and (from MySQL 8.0.28) the `AUDIT_ABORT_EXEMPT` privilege enables system users to execute queries that would otherwise be blocked by an “abort” item in the audit log filter.

6.4.5.2 Installing or Uninstalling MySQL Enterprise Audit

This section describes how to install or uninstall MySQL Enterprise Audit, which is implemented using the audit log plugin and related elements described in Section 6.4.5.1, “Elements of MySQL Enterprise Audit”. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

Plugin upgrades are not automatic when you upgrade a MySQL installation and some plugin loadable functions must be loaded manually (see Installing Loadable Functions). Alternatively, you can reinstall the plugin after upgrading MySQL to load new functions.

Important

Read this entire section before following its instructions. Parts of the procedure differ depending on your environment.

Note

If installed, the `audit_log` plugin involves some minimal overhead even when disabled. To avoid this overhead, do not install MySQL Enterprise Audit unless you plan to use it.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To install MySQL Enterprise Audit, look in the `share` directory of your MySQL installation and choose the script that is appropriate for your platform. The available scripts differ in the suffix used to refer to the plugin library file:

- `audit_log_filter_win_install.sql`: Choose this script for Windows systems that use `.dll` as the file name suffix.
- `audit_log_filter_linux_install.sql`: Choose this script for Linux and similar systems that use `.so` as the file name suffix.

Run the script as follows. The example here uses the Linux installation script. Make the appropriate substitution for your system.

```
$> mysql -u root -p < audit_log_filter_linux_install.sql
Enter password: (enter root password here)
```

Note

Some MySQL versions have introduced changes to the structure of the MySQL Enterprise Audit tables. To ensure that your tables are up to date for upgrades from earlier versions of MySQL 8.0, perform the MySQL upgrade procedure, making sure to use the option that forces an update (see Section 2.10,

"Upgrading MySQL"). If you prefer to run the update statements only for the MySQL Enterprise Audit tables, see the following discussion.

As of MySQL 8.0.12, for new MySQL installations, the `USER` and `HOST` columns in the `audit_log_user` table used by MySQL Enterprise Audit have definitions that better correspond to the definitions of the `User` and `Host` columns in the `mysql.user` system table. For upgrades to an installation for which MySQL Enterprise Audit is already installed, it is recommended that you alter the table definitions as follows:

```
ALTER TABLE mysql.audit_log_user
  DROP FOREIGN KEY audit_log_user_ibfk_1;
ALTER TABLE mysql.audit_log_filter
  CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_as_ci;
ALTER TABLE mysql.audit_log_user
  CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_as_ci;
ALTER TABLE mysql.audit_log_user
  MODIFY COLUMN USER VARCHAR(32);
ALTER TABLE mysql.audit_log_user
  ADD FOREIGN KEY (FILTERNAME) REFERENCES mysql.audit_log_filter(NAME);
```

Note

To use MySQL Enterprise Audit in the context of source/replica replication, Group Replication, or InnoDB Cluster, you must prepare the replica nodes prior to running the installation script on the source node. This is necessary because the `INSTALL PLUGIN` statement in the script is not replicated.

1. On each replica node, extract the `INSTALL PLUGIN` statement from the installation script and execute it manually.
2. On the source node, run the installation script as described previously.

To verify plugin installation, examine the Information Schema `PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, "Obtaining Server Plugin Information"). For example:

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

If the plugin fails to initialize, check the server error log for diagnostic messages.

After MySQL Enterprise Audit is installed, you can use the `--audit-log` option for subsequent server startups to control `audit_log` plugin activation. For example, to prevent the plugin from being removed at runtime, use this option:

```
[mysqld]
audit-log=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without the audit plugin, use `--audit-log` with a value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

Important

By default, rule-based audit log filtering logs no auditable events for any users. This differs from legacy audit log behavior, which logs all auditable events for all users (see Section 6.4.5.10, "Legacy Mode Audit Log Filtering"). Should you wish to produce log-everything behavior with rule-based filtering, create a simple filter to enable logging and assign it to the default account:

```
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
SELECT audit_log_filter_set_user('%', 'log_all');
```

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

Once installed as just described, MySQL Enterprise Audit remains installed until uninstalled. To remove it, execute the following statements:

```
DROP TABLE IF EXISTS mysql.audit_log_user;
DROP TABLE IF EXISTS mysql.audit_log_filter;
UNINSTALL PLUGIN audit_log;
DROP FUNCTION audit_log_filter_set_filter;
DROP FUNCTION audit_log_filter_remove_filter;
DROP FUNCTION audit_log_filter_set_user;
DROP FUNCTION audit_log_filter_remove_user;
DROP FUNCTION audit_log_filter_flush;
DROP FUNCTION audit_log_encryption_password_get;
DROP FUNCTION audit_log_encryption_password_set;
DROP FUNCTION audit_log_read;
DROP FUNCTION audit_log_read_bookmark;
DROP FUNCTION audit_log_rotate;
```

6.4.5.3 MySQL Enterprise Audit Security Considerations

By default, contents of audit log files produced by the audit log plugin are not encrypted and may contain sensitive information, such as the text of SQL statements. For security reasons, audit log files should be written to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log. The default file name is `audit.log` in the data directory. This can be changed by setting the `audit_log_file` system variable at server startup. Other audit log files may exist due to log rotation.

For additional security, enable audit log file encryption. See Encrypting Audit Log Files.

6.4.5.4 Audit Log File Formats

The MySQL server calls the audit log plugin to write an audit record to its log file whenever an auditable event occurs. Typically the first audit record written after plugin startup contains the server description and startup options. Elements following that one represent events such as client connect and disconnect events, executed SQL statements, and so forth. Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures. Contents of files referenced by statements such as `LOAD DATA` are not logged.

To select the log format that the audit log plugin uses to write its log file, set the `audit_log_format` system variable at server startup. These formats are available:

- New-style XML format (`audit_log_format=NEW`): An XML format that has better compatibility with Oracle Audit Vault than old-style XML format. MySQL 8.0 uses new-style XML format by default.
- Old-style XML format (`audit_log_format=OLD`): The original audit log format used by default in older MySQL series.
- JSON format (`audit_log_format=JSON`): Writes the audit log as a JSON array. Only this format supports the optional query time and size statistics, which are available from MySQL 8.0.30.

By default, audit log file contents are written in new-style XML format, without compression or encryption.

If you change `audit_log_format`, it is recommended that you also change `audit_log_file`. For example, if you set `audit_log_format` to `JSON`, set `audit_log_file` to `audit.json`. Otherwise, newer log files will have a different format than older files, but they will all have the same base name with nothing to indicate when the format changed.

- New-Style XML Audit Log File Format
- Old-Style XML Audit Log File Format
- JSON Audit Log File Format

New-Style XML Audit Log File Format

Here is a sample log file in new-style XML format (`audit_log_format=NEW`), reformatted slightly for readability:

```
<?xml version="1.0" encoding="utf-8"?>
<AUDIT>
 <AUDIT_RECORD>
  <TIMESTAMP>2019-10-03T14:06:33 UTC</TIMESTAMP>
  <RECORD_ID>1_2019-10-03T14:06:33</RECORD_ID>
  <NAME>Audit</NAME>
  <SERVER_ID>1</SERVER_ID>
  <VERSION>1</VERSION>
  <STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
    --socket=/usr/local/mysql/mysql.sock
    --port=3306</STARTUP_OPTIONS>
  <OS_VERSION>i686-Linux</OS_VERSION>
  <MYSQL_VERSION>5.7.21-log</MYSQL_VERSION>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
  <TIMESTAMP>2019-10-03T14:09:38 UTC</TIMESTAMP>
  <RECORD_ID>2_2019-10-03T14:06:33</RECORD_ID>
  <NAME>Connect</NAME>
  <CONNECTION_ID>5</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
  <CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
  <CONNECTION_ATTRIBUTES>
   <ATTRIBUTE>
    <NAME>_pid</NAME>
    <VALUE>42794</VALUE>
   </ATTRIBUTE>
   ...
   <ATTRIBUTE>
    <NAME>program_name</NAME>
    <VALUE>mysqladmin</VALUE>
   </ATTRIBUTE>
  </CONNECTION_ATTRIBUTES>
  <PRIV_USER>root</PRIV_USER>
  <PROXY_USER/>
  <DB>test</DB>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2019-10-03T14:09:38 UTC</TIMESTAMP>
  <RECORD_ID>6_2019-10-03T14:06:33</RECORD_ID>
  <NAME>Query</NAME>
  <CONNECTION_ID>5</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>drop_table</COMMAND_CLASS>
  <SQLTEXT>DROP TABLE IF EXISTS t</SQLTEXT>
 </AUDIT_RECORD>
```

```
...

 <AUDIT_RECORD>
  <TIMESTAMP>2019-10-03T14:09:39 UTC</TIMESTAMP>
  <RECORD_ID>8_2019-10-03T14:06:33</RECORD_ID>
  <NAME>Quit</NAME>
  <CONNECTION_ID>5</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
  <CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2019-10-03T14:09:43 UTC</TIMESTAMP>
  <RECORD_ID>11_2019-10-03T14:06:33</RECORD_ID>
  <NAME>Quit</NAME>
  <CONNECTION_ID>6</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
  <CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
  <TIMESTAMP>2019-10-03T14:09:45 UTC</TIMESTAMP>
  <RECORD_ID>12_2019-10-03T14:06:33</RECORD_ID>
  <NAME>NoAudit</NAME>
  <SERVER_ID>1</SERVER_ID>
 </AUDIT_RECORD>
</AUDIT>
```

The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The root element is `<AUDIT>`. The root element contains `<AUDIT_RECORD>` elements, each of which provides information about an audited event. When the audit log plugin begins writing a new log file, it writes the XML declaration and opening `<AUDIT>` root element tag. When the plugin closes a log file, it writes the closing `</AUDIT>` root element tag. The closing tag is not present while the file is open.

Elements within `<AUDIT_RECORD>` elements have these characteristics:

- Some elements appear in every `<AUDIT_RECORD>` element. Others are optional and may appear depending on the audit record type.
- Order of elements within an `<AUDIT_RECORD>` element is not guaranteed.
- Element values are not fixed length. Long values may be truncated as indicated in the element descriptions given later.
- The `<`, `>`, `"`, and `&` characters are encoded as `&lt;`, `&gt;`, `&quot;`, and `&amp;`, respectively. NUL bytes (U+00) are encoded as the `?` character.
- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

  ```
  #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
  ```

The following elements are mandatory in every `<AUDIT_RECORD>` element:

- `<NAME>`

A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

Example:

```
<NAME>Query</NAME>
```

Some common <NAME> values:

```
Audit    When auditing starts, which may be server startup time
Connect  When a client connects, also known as logging in
Query    An SQL statement (executed directly)
Prepare  Preparation of an SQL statement; usually followed by Execute
Execute  Execution of an SQL statement; usually follows Prepare
Shutdown Server shutdown
Quit     When a client disconnects
NoAudit  Auditing has been turned off
```

The possible values are Audit, Binlog Dump, Change user, Close stmt, Connect Out, Connect, Create DB, Daemon, Debug, Delayed insert, Drop DB, Execute, Fetch, Field List, Init DB, Kill, Long Data, NoAudit, Ping, Prepare, Processlist, Query, Quit, Refresh, Register Slave, Reset stmt, Set option, Shutdown, Sleep, Statistics, Table Dump, TableDelete, TableInsert, TableRead, TableUpdate, Time.

Many of these values correspond to the COM_*xxx* command values listed in the my_command.h header file. For example, Create DB and Change user correspond to COM_CREATE_DB and COM_CHANGE_USER, respectively.

Events having <NAME> values of Table*XXX* accompany Query events. For example, the following statement generates one Query event, two TableRead events, and a TableInsert events:

```
INSERT INTO t3 SELECT t1.* FROM t1 JOIN t2;
```

Each Table*XXX* event contains <TABLE> and <DB> elements to identify the table to which the event refers and the database that contains the table.

- <RECORD_ID>

  A unique identifier for the audit record. The value is composed from a sequence number and timestamp, in the format *SEQ_TIMESTAMP*. When the audit log plugin opens the audit log file, it initializes the sequence number to the size of the audit log file, then increments the sequence by 1 for each record logged. The timestamp is a UTC value in *YYYY-MM-DD*T*hh:mm:ss* format indicating the date and time when the audit log plugin opened the file.

  Example:

  ```
  <RECORD_ID>12_2019-10-03T14:06:33</RECORD_ID>
  ```

- <TIMESTAMP>

  A string representing a UTC value in *YYYY-MM-DD*T*hh:mm:ss* UTC format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a <TIMESTAMP> value occurring after the statement finishes, not when it was received.

  Example:

  ```
  <TIMESTAMP>2019-10-03T14:09:45 UTC</TIMESTAMP>
  ```

The following elements are optional in <AUDIT_RECORD> elements. Many of them occur only with specific <NAME> element values.

- <COMMAND_CLASS>

A string that indicates the type of action performed.

Example:

```
<COMMAND_CLASS>drop_table</COMMAND_CLASS>
```

The values correspond to the `statement/sql/xxx` command counters. For example, *`xxx`* is `drop_table` and `select` for `DROP TABLE` and `SELECT` statements, respectively. The following statement displays the possible names:

```
SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%'
ORDER BY name;
```

- `<CONNECTION_ATTRIBUTES>`

  As of MySQL 8.0.19, events with a `<COMMAND_CLASS>` value of `connect` may include a `<CONNECTION_ATTRIBUTES>` element to display the connection attributes passed by the client at connect time. (For information about these attributes, which are also exposed in Performance Schema tables, see Section 27.12.9, “Performance Schema Connection Attribute Tables”.)

  The `<CONNECTION_ATTRIBUTES>` element contains one `<ATTRIBUTE>` element per attribute, each of which contains `<NAME>` and `<VALUE>` elements to indicate the attribute name and value, respectively.

  Example:

```
<CONNECTION_ATTRIBUTES>
 <ATTRIBUTE>
  <NAME>_pid</NAME>
  <VALUE>42794</VALUE>
 </ATTRIBUTE>
 <ATTRIBUTE>
  <NAME>_os</NAME>
  <VALUE>macos0.14</VALUE>
 </ATTRIBUTE>
 <ATTRIBUTE>
  <NAME>_platform</NAME>
  <VALUE>x86_64</VALUE>
 </ATTRIBUTE>
 <ATTRIBUTE>
  <NAME>_client_version</NAME>
  <VALUE>8.0.19</VALUE>
 </ATTRIBUTE>
 <ATTRIBUTE>
  <NAME>_client_name</NAME>
  <VALUE>libmysql</VALUE>
 </ATTRIBUTE>
 <ATTRIBUTE>
  <NAME>program_name</NAME>
  <VALUE>mysqladmin</VALUE>
 </ATTRIBUTE>
</CONNECTION_ATTRIBUTES>
```

  If no connection attributes are present in the event, none are logged and no `<CONNECTION_ATTRIBUTES>` element appears. This can occur if the connection attempt is unsuccessful, the client passes no attributes, or the connection occurs internally such as during server startup or when initiated by a plugin.

- `<CONNECTION_ID>`

  An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

  Example:

```
<CONNECTION_ID>127</CONNECTION_ID>
```

- `<CONNECTION_TYPE>`

  The security state of the connection to the server. Permitted values are `TCP/IP` (TCP/IP connection established without encryption), `SSL/TLS` (TCP/IP connection established with encryption), `Socket` (Unix socket file connection), `Named Pipe` (Windows named pipe connection), and `Shared Memory` (Windows shared memory connection).

  Example:

```
<CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
```

- `<DB>`

  A string representing a database name.

  Example:

```
<DB>test</DB>
```

  For connect events, this element indicates the default database; the element is empty if there is no default database. For table-access events, the element indicates the database to which the accessed table belongs.

- `<HOST>`

  A string representing the client host name.

  Example:

```
<HOST>localhost</HOST>
```

- `<IP>`

  A string representing the client IP address.

  Example:

```
<IP>127.0.0.1</IP>
```

- `<MYSQL_VERSION>`

  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable.

  Example:

```
<MYSQL_VERSION>5.7.21-log</MYSQL_VERSION>
```

- `<OS_LOGIN>`

  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this element is empty. The value is the same as that of the `external_user` system variable (see Section 6.2.19, “Proxy Users”).

  Example:

```
<OS_LOGIN>jeffrey</OS_LOGIN>
```

- `<OS_VERSION>`

  A string representing the operating system on which the server was built or is running.

Example:

```
<OS_VERSION>x86_64-Linux</OS_VERSION>
```

- `<PRIV_USER>`

  A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and may differ from the `<USER>` value.

  Example:

  ```
  <PRIV_USER>jeffrey</PRIV_USER>
  ```

- `<PROXY_USER>`

  A string representing the proxy user (see Section 6.2.19, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example:

  ```
  <PROXY_USER>developer</PROXY_USER>
  ```

- `<SERVER_ID>`

  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example:

  ```
  <SERVER_ID>1</SERVER_ID>
  ```

- `<SQLTEXT>`

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example:

  ```
  <SQLTEXT>DELETE FROM t1</SQLTEXT>
  ```

- `<STARTUP_OPTIONS>`

  A string representing the options that were given on the command line or in option files when the MySQL server was started. The first option is the path to the server executable.

  Example:

  ```
  <STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
    --port=3306 --log_output=FILE</STARTUP_OPTIONS>
  ```

- `<STATUS>`

  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function. See the description for `<STATUS_CODE>` for information about how it differs from `<STATUS>`.

  The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Server Error Message Reference.

  Warnings are not logged.

  Example:

```
<STATUS>1051</STATUS>
```

- `<STATUS_CODE>`

  An unsigned integer representing the command status: 0 for success, 1 if an error occurred.

  The `STATUS_CODE` value differs from the `STATUS` value: `STATUS_CODE` is 0 for success and 1 for error, which is compatible with the EZ_collector consumer for Audit Vault. `STATUS` is the value of the `mysql_errno()` C API function. This is 0 for success and nonzero for error, and thus is not necessarily 1 for error.

  Example:

  ```
  <STATUS_CODE>0</STATUS_CODE>
  ```

- `<TABLE>`

  A string representing a table name.

  Example:

  ```
  <TABLE>t3</TABLE>
  ```

- `<USER>`

  A string representing the user name sent by the client. This may differ from the `<PRIV_USER>` value.

  Example:

  ```
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  ```

- `<VERSION>`

  An unsigned integer representing the version of the audit log file format.

  Example:

  ```
  <VERSION>1</VERSION>
  ```

Old-Style XML Audit Log File Format

Here is a sample log file in old-style XML format (`audit_log_format=OLD`), reformatted slightly for readability:

```
<?xml version="1.0" encoding="utf-8"?>
<AUDIT>
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:00 UTC"
    RECORD_ID="1_2019-10-03T14:25:00"
    NAME="Audit"
    SERVER_ID="1"
    VERSION="1"
    STARTUP_OPTIONS="--port=3306"
    OS_VERSION="i686-Linux"
    MYSQL_VERSION="5.7.21-log"/>
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:24 UTC"
    RECORD_ID="2_2019-10-03T14:25:00"
    NAME="Connect"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
```

```
    COMMAND_CLASS="connect"
    CONNECTION_TYPE="SSL/TLS"
    PRIV_USER="root"
    PROXY_USER=""
    DB="test"/>

...

  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:24 UTC"
    RECORD_ID="6_2019-10-03T14:25:00"
    NAME="Query"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root[root] @ localhost [127.0.0.1]"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="drop_table"
    SQLTEXT="DROP TABLE IF EXISTS t"/>

...

  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:24 UTC"
    RECORD_ID="8_2019-10-03T14:25:00"
    NAME="Quit"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="connect"
    CONNECTION_TYPE="SSL/TLS"/>
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:32 UTC"
    RECORD_ID="12_2019-10-03T14:25:00"
    NAME="NoAudit"
    SERVER_ID="1"/>
</AUDIT>
```

The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The root element is `<AUDIT>`. The root element contains `<AUDIT_RECORD>` elements, each of which provides information about an audited event. When the audit log plugin begins writing a new log file, it writes the XML declaration and opening `<AUDIT>` root element tag. When the plugin closes a log file, it writes the closing `</AUDIT>` root element tag. The closing tag is not present while the file is open.

Attributes of `<AUDIT_RECORD>` elements have these characteristics:

- Some attributes appear in every `<AUDIT_RECORD>` element. Others are optional and may appear depending on the audit record type.
- Order of attributes within an `<AUDIT_RECORD>` element is not guaranteed.
- Attribute values are not fixed length. Long values may be truncated as indicated in the attribute descriptions given later.
- The `<`, `>`, `"`, and `&` characters are encoded as `&lt;`, `&gt;`, `&quot;`, and `&amp;`, respectively. NUL bytes (U+00) are encoded as the `?` character.
- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

  ```
  #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
  ```

The following attributes are mandatory in every `<AUDIT_RECORD>` element:

- `NAME`

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example: `NAME="Query"`

  Some common `NAME` values:

```
Audit    When auditing starts, which may be server startup time
Connect  When a client connects, also known as logging in
Query    An SQL statement (executed directly)
Prepare  Preparation of an SQL statement; usually followed by Execute
Execute  Execution of an SQL statement; usually follows Prepare
Shutdown Server shutdown
Quit     When a client disconnects
NoAudit  Auditing has been turned off
```

  The possible values are `Audit`, `Binlog Dump`, `Change user`, `Close stmt`, `Connect Out`, `Connect`, `Create DB`, `Daemon`, `Debug`, `Delayed insert`, `Drop DB`, `Execute`, `Fetch`, `Field List`, `Init DB`, `Kill`, `Long Data`, `NoAudit`, `Ping`, `Prepare`, `Processlist`, `Query`, `Quit`, `Refresh`, `Register Slave`, `Reset stmt`, `Set option`, `Shutdown`, `Sleep`, `Statistics`, `Table Dump`, `TableDelete`, `TableInsert`, `TableRead`, `TableUpdate`, `Time`.

  Many of these values correspond to the `COM_`*`xxx`* command values listed in the `my_command.h` header file. For example, `"Create DB"` and `"Change user"` correspond to `COM_CREATE_DB` and `COM_CHANGE_USER`, respectively.

  Events having `NAME` values of `Table`*`XXX`* accompany `Query` events. For example, the following statement generates one `Query` event, two `TableRead` events, and a `TableInsert` events:

```
INSERT INTO t3 SELECT t1.* FROM t1 JOIN t2;
```

  Each `Table`*`XXX`* event has `TABLE` and `DB` attributes to identify the table to which the event refers and the database that contains the table.

  `Connect` events for old-style XML audit log format do not include connection attributes.

- `RECORD_ID`

  A unique identifier for the audit record. The value is composed from a sequence number and timestamp, in the format *`SEQ_TIMESTAMP`*. When the audit log plugin opens the audit log file, it initializes the sequence number to the size of the audit log file, then increments the sequence by 1 for each record logged. The timestamp is a UTC value in *`YYYY-MM-DD`*`Thh:mm:ss` format indicating the date and time when the audit log plugin opened the file.

  Example: `RECORD_ID="12_2019-10-03T14:25:00"`

- `TIMESTAMP`

  A string representing a UTC value in *`YYYY-MM-DD`*`Thh:mm:ss UTC` format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a `TIMESTAMP` value occurring after the statement finishes, not when it was received.

  Example: `TIMESTAMP="2019-10-03T14:25:32 UTC"`

The following attributes are optional in `<AUDIT_RECORD>` elements. Many of them occur only for elements with specific values of the `NAME` attribute.

- `COMMAND_CLASS`

  A string that indicates the type of action performed.

Example: `COMMAND_CLASS="drop_table"`

The values correspond to the `statement/sql/xxx` command counters. For example, *`xxx`* is `drop_table` and `select` for `DROP TABLE` and `SELECT` statements, respectively. The following statement displays the possible names:

```
SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%'
ORDER BY name;
```

- `CONNECTION_ID`

  An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

  Example: `CONNECTION_ID="127"`

- `CONNECTION_TYPE`

  The security state of the connection to the server. Permitted values are `TCP/IP` (TCP/IP connection established without encryption), `SSL/TLS` (TCP/IP connection established with encryption), `Socket` (Unix socket file connection), `Named Pipe` (Windows named pipe connection), and `Shared Memory` (Windows shared memory connection).

  Example: `CONNECTION_TYPE="SSL/TLS"`

- `DB`

  A string representing a database name.

  Example: `DB="test"`

  For connect events, this attribute indicates the default database; the attribute is empty if there is no default database. For table-access events, the attribute indicates the database to which the accessed table belongs.

- `HOST`

  A string representing the client host name.

  Example: `HOST="localhost"`

- `IP`

  A string representing the client IP address.

  Example: `IP="127.0.0.1"`

- `MYSQL_VERSION`

  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable.

  Example: `MYSQL_VERSION="5.7.21-log"`

- `OS_LOGIN`

  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this attribute is empty. The value is the same as that of the `external_user` system variable (see Section 6.2.19, “Proxy Users”).

Example: `OS_LOGIN="jeffrey"`

- `OS_VERSION`

  A string representing the operating system on which the server was built or is running.

  Example: `OS_VERSION="x86_64-Linux"`

- `PRIV_USER`

  A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and it may differ from the `USER` value.

  Example: `PRIV_USER="jeffrey"`

- `PROXY_USER`

  A string representing the proxy user (see Section 6.2.19, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example: `PROXY_USER="developer"`

- `SERVER_ID`

  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example: `SERVER_ID="1"`

- `SQLTEXT`

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example: `SQLTEXT="DELETE FROM t1"`

- `STARTUP_OPTIONS`

  A string representing the options that were given on the command line or in option files when the MySQL server was started.

  Example: `STARTUP_OPTIONS="--port=3306 --log_output=FILE"`

- `STATUS`

  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function. See the description for `STATUS_CODE` for information about how it differs from `STATUS`.

  The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Server Error Message Reference.

  Warnings are not logged.

  Example: `STATUS="1051"`

- `STATUS_CODE`

  An unsigned integer representing the command status: 0 for success, 1 if an error occurred.

  The `STATUS_CODE` value differs from the `STATUS` value: `STATUS_CODE` is 0 for success and 1 for error, which is compatible with the EZ_collector consumer for Audit Vault. `STATUS` is the value of the `mysql_errno()` C API function. This is 0 for success and nonzero for error, and thus is not necessarily 1 for error.

  Example: `STATUS_CODE="0"`

- `TABLE`

  A string representing a table name.

  Example: `TABLE="t3"`

- `USER`

  A string representing the user name sent by the client. This may differ from the `PRIV_USER` value.

- `VERSION`

  An unsigned integer representing the version of the audit log file format.

  Example: `VERSION="1"`

JSON Audit Log File Format

For JSON-format audit logging (`audit_log_format=JSON`), the log file contents form a `JSON` array with each array element representing an audited event as a `JSON` hash of key-value pairs. Examples of complete event records appear later in this section. The following is an excerpt of partial events:

```
[
  {
    "timestamp": "2019-10-03 13:50:01",
    "id": 0,
    "class": "audit",
    "event": "startup",
    ...
  },
  {
    "timestamp": "2019-10-03 15:02:32",
    "id": 0,
    "class": "connection",
    "event": "connect",
    ...
  },
  ...
  {
    "timestamp": "2019-10-03 17:37:26",
    "id": 0,
    "class": "table_access",
    "event": "insert",
      ...
  }
  ...
]
```

The audit log file is written using UTF-8 (up to 4 bytes per character). When the audit log plugin begins writing a new log file, it writes the opening `[` array marker. When the plugin closes a log file, it writes the closing `]` array marker. The closing marker is not present while the file is open.

Items within audit records have these characteristics:

- Some items appear in every audit record. Others are optional and may appear depending on the audit record type.

- Order of items within an audit record is not guaranteed.
- Item values are not fixed length. Long values may be truncated as indicated in the item descriptions given later.
- The " and \ characters are encoded as \" and \\, respectively.

JSON format is the only audit log file format that supports the optional query time and size statistics, which are available from MySQL 8.0.30. This data is available in the slow query log for qualifying queries, and in the context of the audit log it similarly helps to detect outliers for activity analysis.

To add the query statistics to the log file, you must set them up as a filter using the `audit_log_filter_set_filter()` audit log function as the service element of the JSON filtering syntax. For instructions to do this, see Adding Query Statistics for Outlier Detection. For the `bytes_sent` and `bytes_received` fields to be populated, the system variable `log_slow_extra` must be set to ON.

The following examples show the JSON object formats for different event types (as indicated by the `class` and `event` items), reformatted slightly for readability:

Auditing startup event:

```
{ "timestamp": "2019-10-03 14:21:56",
  "id": 0,
  "class": "audit",
  "event": "startup",
  "connection_id": 0,
  "startup_data": { "server_id": 1,
                    "os_version": "i686-Linux",
                    "mysql_version": "5.7.21-log",
                    "args": ["/usr/local/mysql/bin/mysqld",
                             "--loose-audit-log-format=JSON",
                             "--log-error=log.err",
                             "--pid-file=mysqld.pid",
                             "--port=3306" ] } }
```

When the audit log plugin starts as a result of server startup (as opposed to being enabled at runtime), `connection_id` is set to 0, and `account` and `login` are not present.

Auditing shutdown event:

```
{ "timestamp": "2019-10-03 14:28:20",
  "id": 3,
  "class": "audit",
  "event": "shutdown",
  "connection_id": 0,
  "shutdown_data": { "server_id": 1 } }
```

When the audit log plugin is uninstalled as a result of server shutdown (as opposed to being disabled at runtime), `connection_id` is set to 0, and `account` and `login` are not present.

Connect or change-user event:

```
{ "timestamp": "2019-10-03 14:23:18",
  "id": 1,
  "class": "connection",
  "event": "connect",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "connection_data": { "connection_type": "ssl",
                       "status": 0,
                       "db": "test",
                       "connection_attributes": {
                         "_pid": "43236",
                         ...
                         "program_name": "mysqladmin"
                       } }
}
```

Disconnect event:

```
{ "timestamp": "2019-10-03 14:24:45",
  "id": 3,
  "class": "connection",
  "event": "disconnect",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "connection_data": { "connection_type": "ssl" } }
```

Query event:

```
{ "timestamp": "2019-10-03 14:23:35",
  "id": 2,
  "class": "general",
  "event": "status",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "general_data": { "command": "Query",
                    "sql_command": "show_variables",
                    "query": "SHOW VARIABLES",
                    "status": 0 } }
```

Query event with optional query statistics for outlier detection:

```
{ "timestamp": "2022-01-28 13:09:30",
  "id": 0,
  "class": "general",
  "event": "status",
  "connection_id": 46,
  "account": { "user": "user", "host": "localhost" },
  "login": { "user": "user", “os": "", “ip": "127.0.0.1", “proxy": "" },
  "general_data": { "command": "Query",
                    "sql_command": "insert",
              "query": "INSERT INTO audit_table VALUES(4)",
              "status": 1146 }
  "query_statistics": { "query_time": 0.116250,
                        "bytes_sent": 18384,
                        "bytes_received": 78858,
                        "rows_sent": 3,
                        "rows_examined": 20878 } }
```

Table access event (read, delete, insert, update):

```
{ "timestamp": "2019-10-03 14:23:41",
  "id": 0,
  "class": "table_access",
  "event": "insert",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "127.0.0.1", "proxy": "" },
  "table_access_data": { "db": "test",
                         "table": "t1",
                         "query": "INSERT INTO t1 (i) VALUES(1),(2),(3)",
                         "sql_command": "insert" } }
```

The items in the following list appear at the top level of JSON-format audit records: Each item value is either a scalar or a `JSON` hash. For items that have a hash value, the description lists only the item names within that hash. For more complete descriptions of second-level hash items, see later in this section.

- `account`

  The MySQL account associated with the event. The value is a hash containing these items equivalent to the value of the `CURRENT_USER()` function within the section: `user`, `host`.

  Example:

```
"account": { "user": "root", "host": "localhost" }
```

- `class`

  A string representing the event class. The class defines the type of event, when taken together with the `event` item that specifies the event subclass.

  Example:

  ```
  "class": "connection"
  ```

  The following table shows the permitted combinations of `class` and `event` values.

  Table 6.34 Audit Log Class and Event Combinations

Class Value	Permitted Event Values
`audit`	`startup`, `shutdown`
`connection`	`connect`, `change_user`, `disconnect`
`general`	`status`
`table_access_data`	`read`, `delete`, `insert`, `update`

- `connection_data`

  Information about a client connection. The value is a hash containing these items: `connection_type`, `status`, `db`, and possibly `connection_attributes`. This item occurs only for audit records with a `class` value of `connection`.

  Example:

  ```
  "connection_data": { "connection_type": "ssl",
                       "status": 0,
                       "db": "test" }
  ```

  As of MySQL 8.0.19, events with a `class` value of `connection` and `event` value of `connect` may include a `connection_attributes` item to display the connection attributes passed by the client at connect time. (For information about these attributes, which are also exposed in Performance Schema tables, see Section 27.12.9, “Performance Schema Connection Attribute Tables”.)

  The `connection_attributes` value is a hash that represents each attribute by its name and value.

  Example:

  ```
  "connection_attributes": {
    "_pid": "43236",
    "_os": "macos0.14",
    "_platform": "x86_64",
    "_client_version": "8.0.19",
    "_client_name": "libmysql",
    "program_name": "mysqladmin"
  }
  ```

  If no connection attributes are present in the event, none are logged and no `connection_attributes` item appears. This can occur if the connection attempt is unsuccessful, the client passes no attributes, or the connection occurs internally such as during server startup or when initiated by a plugin.

- `connection_id`

  An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

  Example:

  ```
  "connection_id": 5
  ```

- `event`

  A string representing the subclass of the event class. The subclass defines the type of event, when taken together with the `class` item that specifies the event class. For more information, see the `class` item description.

  Example:

  ```
  "event": "connect"
  ```

- `general_data`

  Information about an executed statement or command. The value is a hash containing these items: `command`, `sql_command`, `query`, `status`. This item occurs only for audit records with a `class` value of `general`.

  Example:

  ```
  "general_data": { "command": "Query",
                    "sql_command": "show_variables",
                    "query": "SHOW VARIABLES",
                    "status": 0 }
  ```

- `id`

  An unsigned integer representing an event ID.

  Example:

  ```
  "id": 2
  ```

  For audit records that have the same `timestamp` value, their `id` values distinguish them and form a sequence. Within the audit log, `timestamp`/`id` pairs are unique. These pairs are bookmarks that identify event locations within the log.

- `login`

  Information indicating how a client connected to the server. The value is a hash containing these items: `user`, `os`, `ip`, `proxy`.

  Example:

  ```
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" }
  ```

- `query_statistics`

  Optional query statistics for outlier detection. The value is a hash containing these items: `query_time`, `rows_sent`, `rows_examined`, `bytes_received`, `bytes_sent`. For instructions to set up the query statistics, see Adding Query Statistics for Outlier Detection.

  Example:

  ```
  "query_statistics": { "query_time": 0.116250,
                        "bytes_sent": 18384,
                        "bytes_received": 78858,
                        "rows_sent": 3,
  ```

```
                      "rows_examined": 20878 }
```

- `shutdown_data`

  Information pertaining to audit log plugin termination. The value is a hash containing these items: `server_id` This item occurs only for audit records with `class` and `event` values of `audit` and `shutdown`, respectively.

  Example:

  ```
  "shutdown_data": { "server_id": 1 }
  ```

- `startup_data`

  Information pertaining to audit log plugin initialization. The value is a hash containing these items: `server_id`, `os_version`, `mysql_version`, `args`. This item occurs only for audit records with `class` and `event` values of `audit` and `startup`, respectively.

  Example:

  ```
  "startup_data": { "server_id": 1,
                    "os_version": "i686-Linux",
                    "mysql_version": "5.7.21-log",
                    "args": ["/usr/local/mysql/bin/mysqld",
                             "--loose-audit-log-format=JSON",
                             "--log-error=log.err",
                             "--pid-file=mysqld.pid",
                             "--port=3306" ] }
  ```

- `table_access_data`

  Information about an access to a table. The value is a hash containing these items: `db`, `table`, `query`, `sql_command`, This item occurs only for audit records with a `class` value of `table_access`.

  Example:

  ```
  "table_access_data": { "db": "test",
                         "table": "t1",
                         "query": "INSERT INTO t1 (i) VALUES(1),(2),(3)",
                         "sql_command": "insert" }
  ```

- `time`

  This field is similar to that in the `timestamp` field, but the value is an integer and represents the UNIX timestamp value indicating the date and time when the audit event was generated.

  Example:

  ```
  "time" : 1618498687
  ```

  The `time` field occurs in JSON-format log files only if the `audit_log_format_unix_timestamp` system variable is enabled.

- `timestamp`

  A string representing a UTC value in *`YYYY-MM-DD hh:mm:ss`* format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL

statement received from a client has a `timestamp` value occurring after the statement finishes, not when it was received.

Example:

```
"timestamp": "2019-10-03 13:50:01"
```

For audit records that have the same `timestamp` value, their `id` values distinguish them and form a sequence. Within the audit log, `timestamp`/`id` pairs are unique. These pairs are bookmarks that identify event locations within the log.

These items appear within hash values associated with top-level items of JSON-format audit records:

- `args`

  An array of options that were given on the command line or in option files when the MySQL server was started. The first option is the path to the server executable.

  Example:

  ```
  "args": ["/usr/local/mysql/bin/mysqld",
           "--loose-audit-log-format=JSON",
           "--log-error=log.err",
           "--pid-file=mysqld.pid",
           "--port=3306" ]
  ```

- `bytes_received`

  The number of bytes received from the client. This item is part of the optional query statistics. For this field to be populated, the system variable `log_slow_extra` must be set to `ON`.

  Example:

  ```
  "bytes_received": 78858
  ```

- `bytes_sent`

  The number of bytes sent to the client. This item is part of the optional query statistics. For this field to be populated, the system variable `log_slow_extra` must be set to `ON`.

  Example:

  ```
  "bytes_sent": 18384
  ```

- `command`

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example:

  ```
  "command": "Query"
  ```

- `connection_type`

  The security state of the connection to the server. Permitted values are `tcp/ip` (TCP/IP connection established without encryption), `ssl` (TCP/IP connection established with encryption), `socket` (Unix socket file connection), `named_pipe` (Windows named pipe connection), and `shared_memory` (Windows shared memory connection).

  Example:

  ```
  "connection_type": "tcp/tcp"
  ```

- `db`

A string representing a database name. For `connection_data`, it is the default database. For `table_access_data`, it is the table database.

Example:

```
"db": "test"
```

- `host`

  A string representing the client host name.

  Example:

  ```
  "host": "localhost"
  ```

- `ip`

  A string representing the client IP address.

  Example:

  ```
  "ip": "::1"
  ```

- `mysql_version`

  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable.

  Example:

  ```
  "mysql_version": "5.7.21-log"
  ```

- `os`

  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this attribute is empty. The value is the same as that of the `external_user` system variable. See Section 6.2.19, “Proxy Users”.

  Example:

  ```
  "os": "jeffrey"
  ```

- `os_version`

  A string representing the operating system on which the server was built or is running.

  Example:

  ```
  "os_version": "i686-Linux"
  ```

- `proxy`

  A string representing the proxy user (see Section 6.2.19, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example:

  ```
  "proxy": "developer"
  ```

- `query`

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character),

so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

Example:

```
"query": "DELETE FROM t1"
```

- `query_time`

  The query execution time in microseconds (if the `longlong` data type is selected) or seconds (if the `double` data type is selected). This item is part of the optional query statistics.

  Example:

  ```
  "query_time": 0.116250
  ```

- `rows_examined`

  The number of rows accessed during the query. This item is part of the optional query statistics.

  Example:

  ```
  "rows_examined": 20878
  ```

- `rows_sent`

  The number of rows sent to the client as a result. This item is part of the optional query statistics.

  Example:

  ```
  "rows_sent": 3
  ```

- `server_id`

  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example:

  ```
  "server_id": 1
  ```

- `sql_command`

  A string that indicates the SQL statement type.

  Example:

  ```
  "sql_command": "insert"
  ```

  The values correspond to the `statement/sql/xxx` command counters. For example, *`xxx`* is `drop_table` and `select` for `DROP TABLE` and `SELECT` statements, respectively. The following statement displays the possible names:

  ```
  SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
  FROM performance_schema.events_statements_summary_global_by_event_name
  WHERE EVENT_NAME LIKE 'statement/sql/%'
  ORDER BY name;
  ```

- `status`

  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function.

  The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Server Error Message Reference.

Warnings are not logged.

Example:

```
"status": 1051
```

- `table`

  A string representing a table name.

  Example:

  ```
  "table": "t1"
  ```

- `user`

  A string representing a user name. The meaning differs depending on the item within which `user` occurs:

  - Within `account` items, `user` is a string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking.
  - Within `login` items, `user` is a string representing the user name sent by the client.

  Example:

  ```
  "user": "root"
  ```

6.4.5.5 Configuring Audit Logging Characteristics

This section describes how to configure audit logging characteristics, such as the file to which the audit log plugin writes events, the format of written events, whether to enable log file compression and encryption, and space management.

- Naming Conventions for Audit Log Files
- Selecting Audit Log File Format
- Adding Query Statistics for Outlier Detection
- Compressing Audit Log Files
- Encrypting Audit Log Files
- Manually Uncompressing and Decrypting Audit Log Files
- Audit Log File Encryption Prior to MySQL 8.0.17
- Space Management of Audit Log Files
- Write Strategies for Audit Logging

Note

Encryption capabilities described here apply as of MySQL 8.0.17, with the exception of the section that compares current encryption capabilities to the previous more-limited capabilities; see Audit Log File Encryption Prior to MySQL 8.0.17.

For additional information about the functions and system variables that affect audit logging, see Audit Log Functions, and Audit Log Options and Variables.

The audit log plugin can also control which audited events are written to the audit log file, based on event content or the account from which events originate. See Section 6.4.5.7, “Audit Log Filtering”.

Naming Conventions for Audit Log Files

To configure the audit log file name, set the `audit_log_file` system variable at server startup. The default name is `audit.log` in the server data directory. For best security, write the audit log to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log.

The plugin interprets the `audit_log_file` value as composed of an optional leading directory name, a base name, and an optional suffix. If compression or encryption are enabled, the effective file name (the name actually used to create the log file) differs from the configured file name because it has additional suffixes:

- If compression is enabled, the plugin adds a suffix of `.gz`.
- If encryption is enabled, the plugin adds a suffix of `.pwd_id.enc`, where *`pwd_id`* indicates which encryption password to use for log file operations. The audit log plugin stores encryption passwords in the keyring; see Encrypting Audit Log Files.

The effective audit log file name is the name resulting from the addition of applicable compression and encryption suffixes to the configured file name. For example, if the configured `audit_log_file` value is `audit.log`, the effective file name is one of the values shown in the following table.

Enabled Features	Effective File Name
No compression or encryption	`audit.log`
Compression	`audit.log.gz`
Encryption	`audit.log.`*`pwd_id`*`.enc`
Compression, encryption	`audit.log.gz.`*`pwd_id`*`.enc`

`pwd_id` indicates the ID of the password used to encrypt or decrypt a file. *`pwd_id`* format is *`pwd_timestamp-seq`*, where:

- *`pwd_timestamp`* is a UTC value in *`YYYYMMDDThhmmss`* format indicating when the password was created.
- *`seq`* is a sequence number. Sequence numbers start at 1 and increase for passwords that have the same *`pwd_timestamp`* value.

Here are some example *`pwd_id`* password ID values:

```
20190403T142359-1
20190403T142400-1
20190403T142400-2
```

To construct the corresponding keyring IDs for storing passwords in the keyring, the audit log plugin adds a prefix of `audit_log-` to the *`pwd_id`* values. For the example password IDs just shown, the corresponding keyring IDs are:

```
audit_log-20190403T142359-1
audit_log-20190403T142400-1
audit_log-20190403T142400-2
```

The ID of the password currently used for encryption by the audit log plugin is the one having the largest *`pwd_timestamp`* value. If multiple passwords have that *`pwd_timestamp`* value, the current password ID is the one with the largest sequence number. For example, in the preceding set of password IDs, two of them have the largest timestamp, `20190403T142400`, so the current password ID is the one with the largest sequence number (`2`).

The audit log plugin performs certain actions during initialization and termination based on the effective audit log file name:

- During initialization, the plugin checks whether a file with the audit log file name already exists and renames it if so. (In this case, the plugin assumes that the previous server invocation exited unexpectedly with the audit log plugin running.) The plugin then writes to a new empty audit log file.

- During termination, the plugin renames the audit log file.
- File renaming (whether during plugin initialization or termination) occurs according to the usual rules for automatic size-based log file rotation; see Manual Audit Log File Rotation (Before MySQL 8.0.31).

Selecting Audit Log File Format

To configure the audit log file format, set the `audit_log_format` system variable at server startup. These formats are available:

- `NEW`: New-style XML format. This is the default.
- `OLD`: Old-style XML format.
- `JSON`: JSON format. Writes the audit log as a JSON array. Only this format supports the optional query time and size statistics, which are available from MySQL 8.0.30.

For details about each format, see Section 6.4.5.4, “Audit Log File Formats”.

Adding Query Statistics for Outlier Detection

In MySQL 8.0.30 and later, you can extend log files in JSON format with optional data fields to show the query time, the number of bytes sent and received, the number of rows returned to the client, and the number of rows examined. This data is available in the slow query log for qualifying queries, and in the context of the audit log it similarly helps to detect outliers for activity analysis. The extended data fields can be added only when the audit log is in JSON format (`audit_log_format=JSON`), which is not the default setting.

The query statistics are delivered to the audit log through component services that you set up as an audit log filtering function. The services are named `mysql_audit_print_service_longlong_data_source` and `mysql_audit_print_service_double_data_source`. You can choose either data type for each output item. For the query time, `longlong` outputs the value in microseconds, and `double` outputs the value in seconds.

You add the query statistics using the `audit_log_filter_set_filter()` audit log function, as the `service` element of the JSON filtering syntax, as follows:

```
SELECT audit_log_filter_set_filter('QueryStatistics',
                                   '{ "filter": { "class": { "name": "general", "event": { "name": “status'
                                   '{ "service": { "implementation": "mysql_server", "tag": "query_statisti
                                   '{ "name": "query_time",     "type": "double" }, '
                                   '{ "name": "bytes_sent",     "type": "longlong" }, '
                                   '{ "name": "bytes_received", "type": "longlong" }, '
                                   '{ "name": "rows_sent",      "type": "longlong" }, '
                                   '{ "name": "rows_examined",  "type": "longlong" } ] } } } } } }');
```

For the `bytes_sent` and `bytes_received` fields to be populated, the system variable `log_slow_extra` must be set to `ON`. If the system variable is value is `OFF`, a null value is written to the log file for these fields.

If you want to stop collecting the query statistics, use the `audit_log_filter_set_filter()` audit log function to remove the filter, for example:

```
SELECT audit_log_filter_remove_filter('QueryStatistics');
```

Compressing Audit Log Files

Audit log file compression can be enabled for any logging format.

To configure audit log file compression, set the `audit_log_compression` system variable at server startup. Permitted values are `NONE` (no compression; the default) and `GZIP` (GNU Zip compression).

If both compression and encryption are enabled, compression occurs before encryption. To recover the original file manually, first decrypt it, then uncompress it. See Manually Uncompressing and Decrypting Audit Log Files.

Encrypting Audit Log Files

Audit log file encryption can be enabled for any logging format. Encryption is based on user-defined passwords (with the exception of the initial password that the audit log plugin generates). To use this feature, the MySQL keyring must be enabled because audit logging uses it for password storage. Any keyring component or plugin can be used; for instructions, see Section 6.4.4, “The MySQL Keyring”.

To configure audit log file encryption, set the `audit_log_encryption` system variable at server startup. Permitted values are `NONE` (no encryption; the default) and `AES` (AES-256-CBC cipher encryption).

To set or get an encryption password at runtime, use these audit log functions:

- To set the current encryption password, invoke `audit_log_encryption_password_set()`. This function stores the new password in the keyring. If encryption is enabled, it also performs a log file rotation operation that renames the current log file, and begins a new log file encrypted with the password. File renaming occurs according to the usual rules for automatic size-based log file rotation; see Manual Audit Log File Rotation (Before MySQL 8.0.31).

  If the `audit_log_password_history_keep_days` system variable is nonzero, invoking `audit_log_encryption_password_set()` also causes expiration of old archived audit log encryption passwords. For information about audit log password history, including password archiving and expiration, see the description of that variable.

- To get the current encryption password, invoke `audit_log_encryption_password_get()` with no argument. To get a password by ID, pass an argument that specifies the keyring ID of the current password or an archived password.

  To determine which audit log keyring IDs exist, query the Performance Schema `keyring_keys` table:

```
mysql> SELECT KEY_ID FROM performance_schema.keyring_keys
       WHERE KEY_ID LIKE 'audit_log%'
       ORDER BY KEY_ID;
+-----------------------------+
| KEY_ID                      |
+-----------------------------+
| audit_log-20190415T152248-1 |
| audit_log-20190415T153507-1 |
| audit_log-20190416T125122-1 |
| audit_log-20190416T141608-1 |
+-----------------------------+
```

For additional information about audit log encryption functions, see Audit Log Functions.

When the audit log plugin initializes, if it finds that log file encryption is enabled, it checks whether the keyring contains an audit log encryption password. If not, the plugin automatically generates a random initial encryption password and stores it in the keyring. To discover this password, invoke `audit_log_encryption_password_get()`.

If both compression and encryption are enabled, compression occurs before encryption. To recover the original file manually, first decrypt it, then uncompress it. See Manually Uncompressing and Decrypting Audit Log Files.

Manually Uncompressing and Decrypting Audit Log Files

Audit log files can be uncompressed and decrypted using standard tools. This should be done only for log files that have been closed (archived) and are no longer in use, not for the log file that the audit log plugin is currently writing. You can recognize archived log files because they have been renamed by the audit log plugin to include a timestamp in the file name just after the base name.

For this discussion, assume that `audit_log_file` is set to `audit.log`. In that case, an archived audit log file has one of the names shown in the following table.

Enabled Features	Archived File Name
No compression or encryption	`audit.timestamp.log`
Compression	`audit.timestamp.log.gz`
Encryption	`audit.timestamp.log.pwd_id.enc`
Compression, encryption	`audit.timestamp.log.gz.pwd_id.enc`

As discussed in Naming Conventions for Audit Log Files, *`pwd_id`* format is *`pwd_timestamp-seq`*. Thus, the names of archived encrypted log files actually contain two timestamps. The first indicates file rotation time, and the second indicates when the encryption password was created.

Consider the following set of archived encrypted log file names:

```
audit.20190410T205827.log.20190403T185337-1.enc
audit.20190410T210243.log.20190403T185337-1.enc
audit.20190415T145309.log.20190414T223342-1.enc
audit.20190415T151322.log.20190414T223342-2.enc
```

Each file name has a unique rotation-time timestamp. By contrast, the password timestamps are not unique:

- The first two files have the same password ID and sequence number (`20190403T185337-1`). They have the same encryption password.
- The second two files have the same password ID (`20190414T223342`) but different sequence numbers (`1`, `2`). These files have different encryption passwords.

To uncompress a compressed log file manually, use `gunzip`, `gzip -d`, or equivalent command. For example:

```
gunzip -c audit.timestamp.log.gz > audit.timestamp.log
```

To decrypt an encrypted log file manually, use the `openssl` command. For example:

```
openssl enc -d -aes-256-cbc -pass pass:password -md sha256
    -in audit.timestamp.log.pwd_id.enc
    -out audit.timestamp.log
```

To execute that command, you must obtain *`password`*, the encryption password. To do this, use `audit_log_encryption_password_get()`. For example, if the audit log file name is `audit.20190415T151322.log.20190414T223342-2.enc`, the password ID is `20190414T223342-2` and the keyring ID is `audit-log-20190414T223342-2`. Retrieve the keyring password like this:

```
SELECT audit_log_encryption_password_get('audit-log-20190414T223342-2');
```

If both compression and encryption are enabled for audit logging, compression occurs before encryption. In this case, the file name has `.gz` and `.pwd_id.enc` suffixes added, corresponding to the order in which those operations occur. To recover the original file manually, perform the operations in reverse. That is, first decrypt the file, then uncompress it:

```
openssl enc -d -aes-256-cbc -pass pass:password -md sha256
    -in audit.timestamp.log.gz.pwd_id.enc
    -out audit.timestamp.log.gz
gunzip -c audit.timestamp.log.gz > audit.timestamp.log
```

Audit Log File Encryption Prior to MySQL 8.0.17

This section covers the differences in audit log file encryption capabilities prior to and as of MySQL 8.0.17, which is when password history was implemented (which includes password archiving and

expiration). It also indicates how the audit log plugin handles upgrades to MySQL 8.0.17 or higher from versions lower than 8.0.17.

Feature	Prior to MySQL 8.0.17	As of MySQL 8.0.17
Number of passwords	Single password only	Multiple passwords permitted
Encrypted log file names	`.enc` suffix	`.pwd_id.enc` suffix
Password keyring ID	`audit_log`	`audit_log-pwd_id`
Password history	No	Yes

Prior to MySQL 8.0.17, there is no password history, so setting a new password makes the old password inaccessible, rendering MySQL Enterprise Audit unable to read log files encrypted with the old password. Should you anticipate a need to decrypt those files manually, you must maintain a record of previous passwords.

If audit log file encryption is enabled when you upgrade to MySQL 8.0.17 or higher from a lower version, the audit log plugin performs these upgrade actions:

- During plugin initialization, the plugin checks for an encryption password with a keyring ID of `audit_log`. If it finds one, the plugin duplicates the password using a keyring ID in `audit_log-pwd_id` format and uses it as the current encryption password. (For details about *`pwd_id`* syntax, see Naming Conventions for Audit Log Files.)
- Existing encrypted log files have a suffix of `.enc`. The plugin does not rename these to have a suffix of `.pwd_id.enc`, but can read them as long as the key with the ID of `audit_log` remains in the keyring.
- When password cleanup occurs, if the plugin expires any password with a keyring ID in `audit_log-pwd_id` format, it also expires the password with a keyring ID of `audit_log`, if it exists. (At this point, encrypted log files that have a suffix of `.enc` rather than `.pwd_id.enc` become unreadable by the plugin, so it is assumed that you no longer need them.)

Space Management of Audit Log Files

The audit log file has the potential to grow quite large and consume a great deal of disk space. If you are collecting the optional query time and size statistics, which are available from MySQL 8.0.30, this increases the space requirements. The query statistics are only supported with JSON format.

To manage the space used, employ these methods:

- Log file rotation. This involves rotating the current log file by renaming it, then opening a new current log file using the original name. Rotation can be performed manually, or configured to occur automatically.
- Pruning of rotated JSON-format log files, if automatic rotation is enabled. Pruning can be performed based on log file age (as of MySQL 8.0.24), or combined log file size (as of MySQL 8.0.26).

To configure audit log file space management, use the following system variables:

- If `audit_log_rotate_on_size` is 0 (the default), automatic log file rotation is disabled.
  - No rotation occurs unless performed manually.
  - To rotate the current file, use one of the following methods:
    - Before MySQL 8.0.31, manually rename the file, then enable `audit_log_flush` to close it and open a new current log file using the original name. This file rotation method and the `audit_log_flush` variable are deprecated in MySQL 8.0.31.

      With this file rotation method, pruning of rotated JSON-format log files does not occur; `audit_log_max_size` and `audit_log_prune_seconds` have no effect.

- From MySQL 8.0.31, run `SELECT audit_log_rotate();` to rename the file and open a new audit log file using the original name.

  With this file rotation method, pruning of rotated JSON-format log files occurs if `audit_log_max_size` or `audit_log_prune_seconds` has a value greater than 0.

See Manual Audit Log File Rotation (Before MySQL 8.0.31).

- If `audit_log_rotate_on_size` is greater than 0, automatic audit log file rotation is enabled:
  - Automatic rotation occurs when a write to the current log file causes its size to exceed the `audit_log_rotate_on_size` value, as well as under certain other conditions; see Automatic Audit Log File Rotation. When automatic rotation occurs, the audit log plugin renames the current log file and opens a new current log file using the original name.
  - Pruning of rotated JSON-format log files occurs if `audit_log_max_size` or `audit_log_prune_seconds` has a value greater than 0.
  - `audit_log_flush` has no effect.

Note

For JSON-format log files, rotation also occurs when the value of the `audit_log_format_unix_timestamp` system variable is changed at runtime. However, this does not occur for space-management purposes, but rather so that, for a given JSON-format log file, all records in the file either do or do not include the `time` field.

Note

Rotated (renamed) log files are not removed automatically. For example, with size-based log file rotation, renamed log files have unique names and accumulate indefinitely. They do not rotate off the end of the name sequence. To avoid excessive use of space:

- As of MySQL 8.0.24 (for JSON-format log files): Enable log file pruning as described in Audit Log File Pruning.
- Otherwise (for non-JSON files, or prior to MySQL 8.0.24 for all log formats): Remove old files periodically, backing them up first as necessary. If backed-up log files are encrypted, also back up the corresponding encryption passwords to a safe place, should you need to decrypt the files later.

The following sections describe log file rotation and pruning in greater detail.

- Manual Audit Log File Rotation (Before MySQL 8.0.31)
- Manual Audit Log File Rotation (From MySQL 8.0.31)
- Automatic Audit Log File Rotation
- Audit Log File Pruning

Manual Audit Log File Rotation (Before MySQL 8.0.31)

Note

From MySQL 8.0.31, the `audit_log_flush` variable and this method of audit log file rotation are deprecated; expect support to be removed in a future version of MySQL.

If `audit_log_rotate_on_size` is 0 (the default), no log rotation occurs unless performed manually. In this case, the audit log plugin closes and reopens the log file when the `audit_log_flush` value changes from disabled to enabled. Log file renaming must be done externally to the server. Suppose that the log file name is `audit.log` and you want to maintain the three most recent log files, cycling through the names `audit.log.1` through `audit.log.3`. On Unix, perform rotation manually like this:

1. From the command line, rename the current log files:

   ```
   mv audit.log.2 audit.log.3
   mv audit.log.1 audit.log.2
   mv audit.log audit.log.1
   ```

   This strategy overwrites the current `audit.log.3` contents, placing a bound on the number of archived log files and the space they use.

2. At this point, the plugin is still writing to the current log file, which has been renamed to `audit.log.1`. Connect to the server and flush the log file so the plugin closes it and reopens a new `audit.log` file:

   ```
   SET GLOBAL audit_log_flush = ON;
   ```

   `audit_log_flush` is special in that its value remains `OFF` so that you need not disable it explicitly before enabling it again to perform another flush.

Note

If compression or encryption are enabled, log file names include suffixes that signify the enabled features, as well as a password ID if encryption is enabled. If file names include a password ID, be sure to retain the ID in the name of any files you rename manually so that the password to use for decryption operations can be determined.

Note

For JSON-format logging, renaming audit log files manually makes them unavailable to the log-reading functions because the audit log plugin can no longer determine that they are part of the log file sequence (see Section 6.4.5.6, “Reading Audit Log Files”). Consider setting `audit_log_rotate_on_size` greater than 0 to use size-based rotation instead.

Manual Audit Log File Rotation (From MySQL 8.0.31)

If `audit_log_rotate_on_size` is 0 (the default), no log rotation occurs unless performed manually.

To rotate the audit log file manually, run `SELECT audit_log_rotate();` to rename the current audit log file and open a new audit log file. Files are renamed according to the conventions described in Naming Conventions for Audit Log Files.

The `AUDIT_ADMIN` privilege is required to use the `audit_log_rotate()` function.

Managing the number of archived log files (the files that have been renamed) and the space they use is a manual task that involves removing archived audit log files that are no longer needed from your file system.

The content of audit log files that are renamed using the `audit_log_rotate()` function can be read by `audit_log_read()` function.

Automatic Audit Log File Rotation

If `audit_log_rotate_on_size` is greater than 0, setting `audit_log_flush` has no effect. Instead, whenever a write to the current log file causes its size to exceed the `audit_log_rotate_on_size`

value, the audit log plugin automatically renames the current log file and opens a new current log file using the original name.

Automatic size-based rotation also occurs under these conditions:

- During plugin initialization, if a file with the audit log file name already exists (see Naming Conventions for Audit Log Files).
- During plugin termination.
- When the `audit_log_encryption_password_set()` function is called to set the encryption password, if encryption is enabled. (Rotation does not occur if encryption is disabled.)

The plugin renames the original file by inserting a timestamp just after its base name. For example, if the file name is `audit.log`, the plugin renames it to a value such as `audit.20210115T140633.log`. The timestamp is a UTC value in *`YYYYMMDDThhmmss`* format. For XML logging, the timestamp indicates rotation time. For JSON logging, the timestamp is that of the last event written to the file.

If log files are encrypted, the original file name already contains a timestamp indicating the encryption password creation time (see Naming Conventions for Audit Log Files). In this case, the file name after rotation contains two timestamps. For example, an encrypted log file named `audit.log.20210110T130749-1.enc` is renamed to a value such as `audit.20210115T140633.log.20210110T130749-1.enc`.

Audit Log File Pruning

The audit log plugin supports pruning of rotated JSON-format audit log files, if automatic log file rotation is enabled. To use this capability:

- Set `audit_log_format` to `JSON`. (In addition, consider also changing `audit_log_file`; see Selecting Audit Log File Format.)
- Set `audit_log_rotate_on_size` greater than 0 to specify the size in bytes at which automatic log file rotation occurs.
- By default, no pruning of automatically rotated JSON-format log files occurs. To enable pruning, set one of these system variables to a value greater than 0:
  - Set `audit_log_max_size` greater than 0 to specify the limit in bytes on the combined size of rotated log files above which the files become subject to pruning. `audit_log_max_size` is available as of MySQL 8.0.26.
  - Set `audit_log_prune_seconds` greater than 0 to specify the number of seconds after which rotated log files become subject to pruning. `audit_log_prune_seconds` is available as of MySQL 8.0.24.

  Nonzero values of `audit_log_max_size` take precedence over nonzero values of `audit_log_prune_seconds`. If both are set greater than 0 at plugin initialization, a warning is written to the server error log. If a client sets both greater than 0 at runtime, a warning is returned to the client.

Note

Warnings to the error log are written as Notes, which are information messages. To ensure that such messages appear in the error log and are not discarded, make sure that error-logging verbosity is sufficient to include information messages. For example, if you are using priority-based log filtering, as described in Section 5.4.2.5, “Priority-Based Error Log Filtering (log_filter_internal)”, set the `log_error_verbosity` system variable to a value of 3.

Pruning of JSON-format log files, if enabled, occurs as follows:

- When automatic rotation takes place; for the conditions under which this happens, see Automatic Audit Log File Rotation.
- When the global `audit_log_max_size` or `audit_log_prune_seconds` system variable is set at runtime.

For pruning based on combined rotated log file size, if the combined size is greater than the limit specified by `audit_log_max_size`, the audit log plugin removes the oldest files until their combined size does not exceed the limit.

For pruning based on rotated log file age, the pruning point is the current time minus the value of `audit_log_prune_seconds`. In rotated JSON-format log files, the timestamp part of each file name indicates the timestamp of the last event written to the file. The audit log plugin uses file name timestamps to determine which files contain only events older than the pruning point, and removes them.

Write Strategies for Audit Logging

The audit log plugin can use any of several strategies for log writes. Regardless of strategy, logging occurs on a best-effort basis, with no guarantee of consistency.

To specify a write strategy, set the `audit_log_strategy` system variable at server startup. By default, the strategy value is `ASYNCHRONOUS` and the plugin logs asynchronously to a buffer, waiting if the buffer is full. It's possible to tell the plugin not to wait (`PERFORMANCE`) or to log synchronously, either using file system caching (`SEMISYNCHRONOUS`) or forcing output with a `sync()` call after each write request (`SYNCHRONOUS`).

For asynchronous write strategy, the `audit_log_buffer_size` system variable is the buffer size in bytes. Set this variable at server startup to change the buffer size. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin does not allocate this buffer for nonasynchronous write strategies.

Asynchronous logging strategy has these characteristics:

- Minimal impact on server performance and scalability.
- Blocking of threads that generate audit events for the shortest possible time; that is, time to allocate the buffer plus time to copy the event to the buffer.
- Output goes to the buffer. A separate thread handles writes from the buffer to the log file.

With asynchronous logging, the integrity of the log file may be compromised if a problem occurs during a write to the file or if the plugin does not shut down cleanly (for example, in the event that the server host exits unexpectedly). To reduce this risk, set `audit_log_strategy` to use synchronous logging.

A disadvantage of `PERFORMANCE` strategy is that it drops events when the buffer is full. For a heavily loaded server, the audit log may have events missing.

6.4.5.6 Reading Audit Log Files

The audit log plugin supports functions that provide an SQL interface for reading JSON-format audit log files. (This capability does not apply to log files written in other formats.)

When the audit log plugin initializes and is configured for JSON logging, it uses the directory containing the current audit log file as the location to search for readable audit log files. The plugin determines the file location, base name, and suffix from the value of the `audit_log_file` system variable, then looks for files with names that match the following pattern, where `[...]` indicates optional file name parts:

```
basename[.timestamp].suffix[.gz][[.pwd_id].enc]
```

If a file name ends with `.enc`, the file is encrypted and reading its unencrypted contents requires a decryption password obtained from the keyring. The audit log plugin determines the keyring ID of the decryption password as follows:

- If `.enc` is preceded by *`pwd_id`*, the keyring ID is `audit_log-`*`pwd_id`*.
- If `.enc` is not preceded by *`pwd_id`*, the file has an old name from before audit log encryption password history was implemented. The keyring ID is `audit_log`.

For more information about encrypted audit log files, see Encrypting Audit Log Files.

The plugin ignores files that have been renamed manually and do not match the pattern, and files that were encrypted with a password no longer available in the keyring. The plugin opens each remaining candidate file, verifies that the file actually contains `JSON` audit events, and sorts the files using the timestamps from the first event of each file. The result is a sequence of files that are subject to access using the log-reading functions:

- `audit_log_read()` reads events from the audit log or closes the reading process.
- `audit_log_read_bookmark()` returns a bookmark for the most recently written audit log event. This bookmark is suitable for passing to `audit_log_read()` to indicate where to begin reading.

`audit_log_read()` takes an optional `JSON` string argument, and the result returned from a successful call to either function is a `JSON` string.

To use the functions to read the audit log, follow these principles:

- Call `audit_log_read()` to read events beginning from a given position or the current position, or to close reading:
  - To initialize an audit log read sequence, pass an argument that indicates the position at which to begin. One way to do so is to pass the bookmark returned by `audit_log_read_bookmark()`:

    ```
    SELECT audit_log_read(audit_log_read_bookmark());
    ```

  - To continue reading from the current position in the sequence, call `audit_log_read()` with no position specified:

    ```
    SELECT audit_log_read();
    ```

  - To explicitly close the read sequence, pass a `JSON` `null` argument:

    ```
    SELECT audit_log_read('null');
    ```

    It is unnecessary to close reading explicitly. Reading is closed implicitly when the session ends or a new read sequence is initialized by calling `audit_log_read()` with an argument that indicates the position at which to begin.
- A successful call to `audit_log_read()` to read events returns a `JSON` string containing an array of audit events:
  - If the final value of the returned array is not a `JSON` `null` value, there are more events following those just read and `audit_log_read()` can be called again to read more of them.
  - If the final value of the returned array is a `JSON` `null` value, there are no more events left to be read in the current read sequence.

  Each non-`null` array element is an event represented as a `JSON` hash. For example:

  ```
  [
    {
  ```

```
    "timestamp": "2020-05-18 13:39:33", "id": 0,
    "class": "connection", "event": "connect",
    ...
  },
  {
    "timestamp": "2020-05-18 13:39:33", "id": 1,
    "class": "general", "event": "status",
    ...
  },
  {
    "timestamp": "2020-05-18 13:39:33", "id": 2,
    "class": "connection", "event": "disconnect",
    ...
  },
  null
]
```

For more information about the content of JSON-format audit events, see JSON Audit Log File Format.

- An `audit_log_read()` call to read events that does not specify a position produces an error under any of these conditions:
  - A read sequence has not yet been initialized by passing a position to `audit_log_read()`.
  - There are no more events left to be read in the current read sequence; that is, `audit_log_read()` previously returned an array ending with a `JSON null` value.
  - The most recent read sequence has been closed by passing a `JSON null` value to `audit_log_read()`.

  To read events under those conditions, it is necessary to first initialize a read sequence by calling `audit_log_read()` with an argument that specifies a position.

To specify a position to `audit_log_read()`, include an argument that indicates where to begin reading. For example, pass a bookmark, which is a `JSON` hash containing `timestamp` and `id` elements that uniquely identify a particular event. Here is an example bookmark, obtained by calling the `audit_log_read_bookmark()` function:

```
mysql> SELECT audit_log_read_bookmark();
+------------------------------------------------+
| audit_log_read_bookmark()                      |
+------------------------------------------------+
| { "timestamp": "2020-05-18 21:03:44", "id": 0 } |
+------------------------------------------------+
```

Passing the current bookmark to `audit_log_read()` initializes event reading beginning at the bookmark position:

```
mysql> SELECT audit_log_read(audit_log_read_bookmark());
+-------------------------------------------------------------------------+
| audit_log_read(audit_log_read_bookmark())                               |
+-------------------------------------------------------------------------+
| [ {"timestamp":"2020-05-18 22:41:24","id":0,"class":"connection", ... |
+-------------------------------------------------------------------------+
```

The argument to `audit_log_read()` is optional. If present, it can be a `JSON null` value to close the read sequence, or a `JSON` hash.

Within a hash argument to `audit_log_read()`, items are optional and control aspects of the read operation such as the position at which to begin reading or how many events to read. The following items are significant (other items are ignored):

- `start`: The position within the audit log of the first event to read. The position is given as a timestamp and the read starts from the first event that occurs on or after the timestamp value. The `start` item has this format, where *`value`* is a literal timestamp value:

```
"start": { "timestamp": "value" }
```

The `start` item is permitted as of MySQL 8.0.22.

- `timestamp`, `id`: The position within the audit log of the first event to read. The `timestamp` and `id` items together comprise a bookmark that uniquely identify a particular event. If an `audit_log_read()` argument includes either item, it must include both to completely specify a position or an error occurs.
- `max_array_length`: The maximum number of events to read from the log. If this item is omitted, the default is to read to the end of the log or until the read buffer is full, whichever comes first.

To specify a starting position to `audit_log_read()`, pass a hash argument that includes either a `start` item or a bookmark consisting of `timestamp` and `id` items. If a hash argument includes both a `start` item and a bookmark, an error occurs.

If a hash argument specifies no starting position, reading continues from the current position.

If a timestamp value includes no time part, a time part of `00:00:00` is assumed.

Example arguments accepted by `audit_log_read()`:

- Read events starting with the first event that occurs on or after the given timestamp:

```
audit_log_read('{ "start": { "timestamp": "2020-05-24 12:30:00" } }')
```

- Like the previous example, but read at most 3 events:

```
audit_log_read('{ "start": { "timestamp": "2020-05-24 12:30:00" }, "max_array_length": 3 }')
```

- Read events starting with the first event that occurs on or after `2020-05-24 00:00:00` (the timestamp includes no time part, so `00:00:00` is assumed):

```
audit_log_read('{ "start": { "timestamp": "2020-05-24" } }')
```

- Read events starting with the event that has the exact timestamp and event ID:

```
audit_log_read('{ "timestamp": "2020-05-24 12:30:00", "id": 0 }')
```

- Like the previous example, but read at most 3 events:

```
audit_log_read('{ "timestamp": "2020-05-24 12:30:00", "id": 0, "max_array_length": 3 }')
```

- Read events from the current position in the read sequence:

```
audit_log_read()
```

- Read at most 5 events beginning at the current position in the read sequence:

```
audit_log_read('{ "max_array_length": 5 }')
```

- Close the current read sequence:

```
audit_log_read('null')
```

A `JSON` string returned from either log-reading function can be manipulated as necessary. Suppose that a call to obtain a bookmark produces this value:

```
mysql> SET @mark := audit_log_read_bookmark();
mysql> SELECT @mark;
+-----------------------------------------------+
| @mark                                         |
+-----------------------------------------------+
| { "timestamp": "2020-05-18 16:10:28", "id": 2 } |
```

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

Calling `audit_log_read()` with that argument can return multiple events. To limit `audit_log_read()` to reading at most *`N`* events, add to the string a `max_array_length` item with that value. For example, to read a single event, modify the string as follows:

```
mysql> SET @mark := JSON_SET(@mark, '$.max_array_length', 1);
mysql> SELECT @mark;
+-------------------------------------------------------------------------+
| @mark                                                                   |
+-------------------------------------------------------------------------+
| {"id": 2, "timestamp": "2020-05-18 16:10:28", "max_array_length": 1} |
+-------------------------------------------------------------------------+
```

The modified string, when passed to `audit_log_read()`, produces a result containing at most one event, no matter how many are available.

Prior to MySQL 8.0.19, string return values from audit log functions are binary strings. To use a binary string with functions that require a nonbinary string (such as functions that manipulate `JSON` values), convert it to a nonbinary string. For example, before passing a bookmark to `JSON_SET()`, convert it to `utf8mb4` as follows:

```
SET @mark = CONVERT(@mark USING utf8mb4);
```

That statement can be used even for MySQL 8.0.19 and higher; for those versions, it is essentially a no-op and is harmless.

If an audit log function is invoked from within the `mysql` client, binary string results display using hexadecimal notation, depending on the value of the `--binary-as-hex`. For more information about that option, see Section 4.5.1, “mysql — The MySQL Command-Line Client”.

To set a limit on the number of bytes that `audit_log_read()` reads, set the `audit_log_read_buffer_size` system variable. As of MySQL 8.0.12, this variable has a default of 32KB and can be set at runtime. Each client should set its session value of `audit_log_read_buffer_size` appropriately for its use of `audit_log_read()`.

Each call to `audit_log_read()` returns as many available events as fit within the buffer size. Events that do not fit within the buffer size are skipped and generate warnings. Given this behavior, consider these factors when assessing the proper buffer size for an application:

- There is a tradeoff between number of calls to `audit_log_read()` and events returned per call:
  - With a smaller buffer size, calls return fewer events, so more calls are needed.
  - With a larger buffer size, calls return more events, so fewer calls are needed.
- With a smaller buffer size, such as the default size of 32KB, there is a greater chance for events to exceed the buffer size and thus to be skipped.

Prior to MySQL 8.0.12, `audit_log_read_buffer_size` has a default of 1MB, affects all clients, and can be changed only at server startup.

For additional information about audit log-reading functions, see Audit Log Functions.

6.4.5.7 Audit Log Filtering

Note

For audit log filtering to work as described here, the audit log plugin *and the accompanying audit tables and functions* must be installed. If the plugin is installed without the accompanying audit tables and functions needed for rule-based filtering, the plugin operates in legacy filtering mode, described in

Section 6.4.5.10, “Legacy Mode Audit Log Filtering”. Legacy mode is filtering behavior as it was prior to MySQL 5.7.13; that is, before the introduction of rule-based filtering.

- Properties of Audit Log Filtering
- Constraints on Audit Log Filtering Functions
- Using Audit Log Filtering Functions

Properties of Audit Log Filtering

The audit log plugin has the capability of controlling logging of audited events by filtering them:

- Audited events can be filtered using these characteristics:
  - User account
  - Audit event class
  - Audit event subclass
  - Audit event fields such as those that indicate operation status or SQL statement executed
- Audit filtering is rule based:
  - A filter definition creates a set of auditing rules. Definitions can be configured to include or exclude events for logging based on the characteristics just described.
  - Filter rules have the capability of blocking (aborting) execution of qualifying events, in addition to existing capabilities for event logging.
  - Multiple filters can be defined, and any given filter can be assigned to any number of user accounts.
  - It is possible to define a default filter to use with any user account that has no explicitly assigned filter.

  Audit log filtering is used to implement component services from MySQL 8.0.30. To get the optional query statistics available from that release, you set them up as a filter using the service component, which implements the services that write the statistics to the audit log. For instructions to set this filter up, see Adding Query Statistics for Outlier Detection.

  For information about writing filtering rules, see Section 6.4.5.8, “Writing Audit Log Filter Definitions”.
- Audit filters can be defined, displayed, and modified using an SQL interface based on function calls.
- Audit filter definitions are stored in the tables in the `mysql` system database.
- Within a given session, the value of the read-only `audit_log_filter_id` system variable indicates whether a filter is assigned to the session.

Note

By default, rule-based audit log filtering logs no auditable events for any users. To log all auditable events for all users, use the following statements, which create a simple filter to enable logging and assign it to the default account:

```
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
SELECT audit_log_filter_set_user('%', 'log_all');
```

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

As previously mentioned, the SQL interface for audit filtering control is function based. The following list briefly summarizes these functions:

- `audit_log_filter_set_filter()`: Define a filter.
- `audit_log_filter_remove_filter()`: Remove a filter.
- `audit_log_filter_set_user()`: Start filtering a user account.
- `audit_log_filter_remove_user()`: Stop filtering a user account.
- `audit_log_filter_flush()`: Flush manual changes to the filter tables to affect ongoing filtering.

For usage examples and complete details about the filtering functions, see Using Audit Log Filtering Functions, and Audit Log Functions.

Constraints on Audit Log Filtering Functions

Audit log filtering functions are subject to these constraints:

- To use any filtering function, the `audit_log` plugin must be enabled or an error occurs. In addition, the audit tables must exist or an error occurs. To install the `audit_log` plugin and its accompanying functions and tables, see Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”.
- To use any filtering function, a user must possess the `AUDIT_ADMIN` `SUPER` privilege or an error occurs. To grant one of these privileges to a user account, use this statement:

  ```
  GRANT privilege ON *.* TO user;
  ```

  Alternatively, should you prefer to avoid granting the `AUDIT_ADMIN` or `SUPER` privilege while still permitting users to access specific filtering functions, “wrapper” stored programs can be defined. This technique is described in the context of keyring functions in Using General-Purpose Keyring Functions; it can be adapted for use with filtering functions.

- The `audit_log` plugin operates in legacy mode if it is installed but the accompanying audit tables and functions are not created. The plugin writes these messages to the error log at server startup:

  ```
  [Warning] Plugin audit_log reported: 'Failed to open the audit log filter tables.'
  [Warning] Plugin audit_log reported: 'Audit Log plugin supports a filtering,
  which has not been installed yet. Audit Log plugin will run in the legacy
  mode, which will be disabled in the next release.'
  ```

  In legacy mode, filtering can be done based only on event account or status. For details, see Section 6.4.5.10, “Legacy Mode Audit Log Filtering”.

- It is theoretically possible for a user with sufficient permissions to mistakenly create an “abort” item in the audit log filter that prevents themselves and other administrators from accessing the system. From MySQL 8.0.28, the `AUDIT_ABORT_EXEMPT` privilege is available to permit a user account’s queries to always be executed even if an “abort” item would block them. Accounts with this privilege can therefore be used to regain access to a system following an audit misconfiguration. The query is still logged in the audit log, but instead of being rejected, it is permitted due to the privilege.

  Accounts created in MySQL 8.0.28 or later with the `SYSTEM_USER` privilege have the `AUDIT_ABORT_EXEMPT` privilege assigned automatically when they are created. The `AUDIT_ABORT_EXEMPT` privilege is also assigned to existing accounts with the `SYSTEM_USER` privilege when you carry out an upgrade procedure with MySQL 8.0.28 or later, if no existing accounts have that privilege assigned.

Using Audit Log Filtering Functions

Before using the audit log functions, install them according to the instructions provided in Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”. The `AUDIT_ADMIN` or `SUPER` privilege is required to use any of these functions.

The audit log filtering functions enable filtering control by providing an interface to create, modify, and remove filter definitions and assign filters to user accounts.

Filter definitions are `JSON` values. For information about using `JSON` data in MySQL, see Section 11.5, “The JSON Data Type”. This section shows some simple filter definitions. For more information about filter definitions, see Section 6.4.5.8, “Writing Audit Log Filter Definitions”.

When a connection arrives, the audit log plugin determines which filter to use for the new session by searching for the user account name in the current filter assignments:

- If a filter is assigned to the user, the audit log uses that filter.
- Otherwise, if no user-specific filter assignment exists, but there is a filter assigned to the default account (`%`), the audit log uses the default filter.
- Otherwise, the audit log selects no audit events from the session for processing.

If a change-user operation occurs during a session (see mysql_change_user()), filter assignment for the session is updated using the same rules but for the new user.

By default, no accounts have a filter assigned, so no processing of auditable events occurs for any account.

Suppose that you want to change the default to be to log only connection-related activity (for example, to see connect, change-user, and disconnect events, but not the SQL statements users execute while connected). To achieve this, define a filter (shown here named `log_conn_events`) that enables logging only of events in the `connection` class, and assign that filter to the default account, represented by the `%` account name:

```
SET @f = '{ "filter": { "class": { "name": "connection" } } }';
SELECT audit_log_filter_set_filter('log_conn_events', @f);
SELECT audit_log_filter_set_user('%', 'log_conn_events');
```

Now the audit log uses this default account filter for connections from any account that has no explicitly defined filter.

To assign a filter explicitly to a particular user account or accounts, define the filter, then assign it to the relevant accounts:

```
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
SELECT audit_log_filter_set_user('user1@localhost', 'log_all');
SELECT audit_log_filter_set_user('user2@localhost', 'log_all');
```

Now full logging is enabled for `user1@localhost` and `user2@localhost`. Connections from other accounts continue to be filtered using the default account filter.

To disassociate a user account from its current filter, either unassign the filter or assign a different filter:

- To unassign the filter from the user account:

  ```
  SELECT audit_log_filter_remove_user('user1@localhost');
  ```

  Filtering of current sessions for the account remains unaffected. Subsequent connections from the account are filtered using the default account filter if there is one, and are not logged otherwise.

- To assign a different filter to the user account:

  ```
  SELECT audit_log_filter_set_filter('log_nothing', '{ "filter": { "log": false } }');
  SELECT audit_log_filter_set_user('user1@localhost', 'log_nothing');
  ```

  Filtering of current sessions for the account remains unaffected. Subsequent connections from the account are filtered using the new filter. For the filter shown here, that means no logging for new connections from `user1@localhost`.

For audit log filtering, user name and host name comparisons are case-sensitive. This differs from comparisons for privilege checking, for which host name comparisons are not case-sensitive.

To remove a filter, do this:

```
SELECT audit_log_filter_remove_filter('log_nothing');
```

Removing a filter also unassigns it from any users to whom it is assigned, including any current sessions for those users.

The filtering functions just described affect audit filtering immediately and update the audit log tables in the `mysql` system database that store filters and user accounts (see Audit Log Tables). It is also possible to modify the audit log tables directly using statements such as `INSERT`, `UPDATE`, and `DELETE`, but such changes do not affect filtering immediately. To flush your changes and make them operational, call `audit_log_filter_flush()`:

```
SELECT audit_log_filter_flush();
```

Warning

`audit_log_filter_flush()` should be used only after modifying the audit tables directly, to force reloading all filters. Otherwise, this function should be avoided. It is, in effect, a simplified version of unloading and reloading the `audit_log` plugin with `UNINSTALL PLUGIN` plus `INSTALL PLUGIN`.

`audit_log_filter_flush()` affects all current sessions and detaches them from their previous filters. Current sessions are no longer logged unless they disconnect and reconnect, or execute a change-user operation.

To determine whether a filter is assigned to the current session, check the session value of the read-only `audit_log_filter_id` system variable. If the value is 0, no filter is assigned. A nonzero value indicates the internally maintained ID of the assigned filter:

```
mysql> SELECT @@audit_log_filter_id;
+-----------------------+
| @@audit_log_filter_id |
+-----------------------+
|                     2 |
+-----------------------+
```

6.4.5.8 Writing Audit Log Filter Definitions

Filter definitions are `JSON` values. For information about using `JSON` data in MySQL, see Section 11.5, “The JSON Data Type”.

Filter definitions have this form, where *`actions`* indicates how filtering takes place:

```
{ "filter": actions }
```

The following discussion describes permitted constructs in filter definitions.

- Logging All Events
- Logging Specific Event Classes
- Logging Specific Event Subclasses
- Inclusive and Exclusive Logging
- Testing Event Field Values
- Blocking Execution of Specific Events
- Logical Operators

- Referencing Predefined Variables
- Referencing Predefined Functions
- Replacement of Event Field Values
- Replacing a User Filter

Logging All Events

To explicitly enable or disable logging of all events, use a `log` item in the filter:

```
{
  "filter": { "log": true }
}
```

The `log` value can be either `true` or `false`.

The preceding filter enables logging of all events. It is equivalent to:

```
{
  "filter": { }
}
```

Logging behavior depends on the `log` value and whether `class` or `event` items are specified:

- With `log` specified, its given value is used.
- Without `log` specified, logging is `true` if no `class` or `event` item is specified, and `false` otherwise (in which case, `class` or `event` can include their own `log` item).

Logging Specific Event Classes

To log events of a specific class, use a `class` item in the filter, with its `name` field denoting the name of the class to log:

```
{
  "filter": {
    "class": { "name": "connection" }
  }
}
```

The `name` value can be `connection`, `general`, or `table_access` to log connection, general, or table-access events, respectively.

The preceding filter enables logging of events in the `connection` class. It is equivalent to the following filter with `log` items made explicit:

```
{
  "filter": {
    "log": false,
    "class": { "log": true,
               "name": "connection" }
  }
}
```

To enable logging of multiple classes, define the `class` value as a `JSON` array element that names the classes:

```
{
  "filter": {
    "class": [
      { "name": "connection" },
      { "name": "general" },
      { "name": "table_access" }
    ]
  }
}
```

Note

When multiple instances of a given item appear at the same level within a filter definition, the item values can be combined into a single instance of that item within an array value. The preceding definition can be written like this:

```
{
  "filter": {
    "class": [
      { "name": [ "connection", "general", "table_access" ] }
    ]
  }
}
```

Logging Specific Event Subclasses

To select specific event subclasses, use an `event` item containing a `name` item that names the subclasses. The default action for events selected by an `event` item is to log them. For example, this filter enables logging for the named event subclasses:

```
{
  "filter": {
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect" },
          { "name": "disconnect" }
        ]
      },
      { "name": "general" },
      {
        "name": "table_access",
        "event": [
          { "name": "insert" },
          { "name": "delete" },
          { "name": "update" }
        ]
      }
    ]
  }
}
```

The `event` item can also contain explicit `log` items to indicate whether to log qualifying events. This `event` item selects multiple events and explicitly indicates logging behavior for them:

```
"event": [
  { "name": "read", "log": false },
  { "name": "insert", "log": true },
  { "name": "delete", "log": true },
  { "name": "update", "log": true }
]
```

The `event` item can also indicate whether to block qualifying events, if it contains an `abort` item. For details, see Blocking Execution of Specific Events.

Table 6.35, “Event Class and Subclass Combinations” describes the permitted subclass values for each event class.

Table 6.35 Event Class and Subclass Combinations

Event Class	Event Subclass	Description
`connection`	`connect`	Connection initiation (successful or unsuccessful)
`connection`	`change_user`	User re-authentication with different user/password during session

Event Class	Event Subclass	Description
connection	disconnect	Connection termination
general	status	General operation information
message	internal	Internally generated message
message	user	Message generated by audit_api_message_emit_udf()
table_access	read	Table read statements, such as SELECT or INSERT INTO ... SELECT
table_access	delete	Table delete statements, such as DELETE or TRUNCATE TABLE
table_access	insert	Table insert statements, such as INSERT or REPLACE
table_access	update	Table update statements, such as UPDATE

Table 6.36, “Log and Abort Characteristics Per Event Class and Subclass Combination” describes for each event subclass whether it can be logged or aborted.

Table 6.36 Log and Abort Characteristics Per Event Class and Subclass Combination

Event Class	Event Subclass	Can be Logged	Can be Aborted
connection	connect	Yes	No
connection	change_user	Yes	No
connection	disconnect	Yes	No
general	status	Yes	No
message	internal	Yes	Yes
message	user	Yes	Yes
table_access	read	Yes	Yes
table_access	delete	Yes	Yes
table_access	insert	Yes	Yes
table_access	update	Yes	Yes

Inclusive and Exclusive Logging

A filter can be defined in inclusive or exclusive mode:

- Inclusive mode logs only explicitly specified items.
- Exclusive mode logs everything but explicitly specified items.

To perform inclusive logging, disable logging globally and enable logging for specific classes. This filter logs connect and disconnect events in the connection class, and events in the general class:

```
{
  "filter": {
    "log": false,
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect", "log": true },
          { "name": "disconnect", "log": true }
        ]
      },
      { "name": "general", "log": true }
```

```
    ]
  }
}
```

To perform exclusive logging, enable logging globally and disable logging for specific classes. This filter logs everything except events in the `general` class:

```
{
  "filter": {
    "log": true,
    "class":
      { "name": "general", "log": false }
  }
}
```

This filter logs `change_user` events in the `connection` class, `message` events, and `table_access` events, by virtue of *not* logging everything else:

```
{
  "filter": {
    "log": true,
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect", "log": false },
          { "name": "disconnect", "log": false }
        ]
      },
      { "name": "general", "log": false }
    ]
  }
}
```

Testing Event Field Values

To enable logging based on specific event field values, specify a `field` item within the `log` item that indicates the field name and its expected value:

```
{
  "filter": {
    "class": {
    "name": "general",
      "event": {
        "name": "status",
        "log": {
          "field": { "name": "general_command.str", "value": "Query" }
        }
      }
    }
  }
}
```

Each event contains event class-specific fields that can be accessed from within a filter to perform custom filtering.

An event in the `connection` class indicates when a connection-related activity occurs during a session, such as a user connecting to or disconnecting from the server. Table 6.37, “Connection Event Fields” indicates the permitted fields for `connection` events.

Table 6.37 Connection Event Fields

Field Name	Field Type	Description
`status`	integer	Event status: 0: OK Otherwise: Failed

Field Name	Field Type	Description
`connection_id`	unsigned integer	Connection ID
`user.str`	string	User name specified during authentication
`user.length`	unsigned integer	User name length
`priv_user.str`	string	Authenticated user name (account user name)
`priv_user.length`	unsigned integer	Authenticated user name length
`external_user.str`	string	External user name (provided by third-party authentication plugin)
`external_user.length`	unsigned integer	External user name length
`proxy_user.str`	string	Proxy user name
`proxy_user.length`	unsigned integer	Proxy user name length
`host.str`	string	Connected user host
`host.length`	unsigned integer	Connected user host length
`ip.str`	string	Connected user IP address
`ip.length`	unsigned integer	Connected user IP address length
`database.str`	string	Database name specified at connect time
`database.length`	unsigned integer	Database name length
`connection_type`	integer	Connection type: 0 or `"::undefined"`: Undefined 1 or `"::tcp/ip"`: TCP/IP 2 or `"::socket"`: Socket 3 or `"::named_pipe"`: Named pipe 4 or `"::ssl"`: TCP/IP with encryption 5 or `"::shared_memory"`: Shared memory

The `"::xxx"` values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

An event in the `general` class indicates the status code of an operation and its details. Table 6.38, “General Event Fields” indicates the permitted fields for `general` events.

Table 6.38 General Event Fields

Field Name	Field Type	Description
`general_error_code`	integer	Event status: 0: OK Otherwise: Failed

Field Name	Field Type	Description
general_thread_id	unsigned integer	Connection/thread ID
general_user.str	string	User name specified during authentication
general_user.length	unsigned integer	User name length
general_command.str	string	Command name
general_command.length	unsigned integer	Command name length
general_query.str	string	SQL statement text
general_query.length	unsigned integer	SQL statement text length
general_host.str	string	Host name
general_host.length	unsigned integer	Host name length
general_sql_command.str	string	SQL command type name
general_sql_command.length	unsigned integer	SQL command type name length
general_external_user.str	string	External user name (provided by third-party authentication plugin)
general_external_user.length	unsigned integer	External user name length
general_ip.str	string	Connected user IP address
general_ip.length	unsigned integer	Connection user IP address length

general_command.str indicates a command name: Query, Execute, Quit, or Change user.

A general event with the general_command.str field set to Query or Execute contains general_sql_command.str set to a value that specifies the type of SQL command: alter_db, alter_db_upgrade, admin_commands, and so forth. The available general_sql_command.str values can be seen as the last components of the Performance Schema instruments displayed by this statement:

```
mysql> SELECT NAME FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'statement/sql/%' ORDER BY NAME;
+---------------------------------------+
| NAME                                  |
+---------------------------------------+
| statement/sql/alter_db                |
| statement/sql/alter_db_upgrade        |
| statement/sql/alter_event             |
| statement/sql/alter_function          |
| statement/sql/alter_instance          |
| statement/sql/alter_procedure         |
| statement/sql/alter_server            |
...
```

An event in the table_access class provides information about a specific type of access to a table. Table 6.39, “Table-Access Event Fields” indicates the permitted fields for table_access events.

Table 6.39 Table-Access Event Fields

Field Name	Field Type	Description
connection_id	unsigned integer	Event connection ID
sql_command_id	integer	SQL command ID
query.str	string	SQL statement text
query.length	unsigned integer	SQL statement text length
table_database.str	string	Database name associated with event

Field Name	Field Type	Description
`table_database.length`	unsigned integer	Database name length
`table_name.str`	string	Table name associated with event
`table_name.length`	unsigned integer	Table name length

The following list shows which statements produce which table-access events:

- `read` event:
  - `SELECT`
  - `INSERT ... SELECT` (for tables referenced in `SELECT` clause)
  - `REPLACE ... SELECT` (for tables referenced in `SELECT` clause)
  - `UPDATE ... WHERE` (for tables referenced in `WHERE` clause)
  - `HANDLER ... READ`
- `delete` event:
  - `DELETE`
  - `TRUNCATE TABLE`
- `insert` event:
  - `INSERT`
  - `INSERT ... SELECT` (for table referenced in `INSERT` clause)
  - `REPLACE`
  - `REPLACE ... SELECT` (for table referenced in `REPLACE` clause
  - `LOAD DATA`
  - `LOAD XML`
- `update` event:
  - `UPDATE`
  - `UPDATE ... WHERE` (for tables referenced in `UPDATE` clause)

Blocking Execution of Specific Events

`event` items can include an `abort` item that indicates whether to prevent qualifying events from executing. `abort` enables rules to be written that block execution of specific SQL statements.

Important

It is theoretically possible for a user with sufficient permissions to mistakenly create an `abort` item in the audit log filter that prevents themselves and other administrators from accessing the system. From MySQL 8.0.28, the `AUDIT_ABORT_EXEMPT` privilege is available to permit a user account's queries to always be executed even if an `abort` item would block them. Accounts with this privilege can therefore be used to regain access to a system following an audit misconfiguration. The query is still logged in the audit log, but instead of being rejected, it is permitted due to the privilege.

> Accounts created in MySQL 8.0.28 or later with the `SYSTEM_USER` privilege have the `AUDIT_ABORT_EXEMPT` privilege assigned automatically when they are created. The `AUDIT_ABORT_EXEMPT` privilege is also assigned to existing accounts with the `SYSTEM_USER` privilege when you carry out an upgrade procedure with MySQL 8.0.28 or later, if no existing accounts have that privilege assigned.

The `abort` item must appear within an `event` item. For example:

```
"event": {
  "name": qualifying event subclass names
  "abort": condition
}
```

For event subclasses selected by the `name` item, the `abort` action is true or false, depending on *`condition`* evaluation. If the condition evaluates to true, the event is blocked. Otherwise, the event continues executing.

The *`condition`* specification can be as simple as `true` or `false`, or it can be more complex such that evaluation depends on event characteristics.

This filter blocks `INSERT`, `UPDATE`, and `DELETE` statements:

```
{
  "filter": {
    "class": {
      "name": "table_access",
      "event": {
        "name": [ "insert", "update", "delete" ],
        "abort": true
      }
    }
  }
}
```

This more complex filter blocks the same statements, but only for a specific table (`finances.bank_account`):

```
{
  "filter": {
    "class": {
      "name": "table_access",
      "event": {
        "name": [ "insert", "update", "delete" ],
        "abort": {
          "and": [
            { "field": { "name": "table_database.str", "value": "finances" } },
            { "field": { "name": "table_name.str", "value": "bank_account" } }
          ]
        }
      }
    }
  }
}
```

Statements matched and blocked by the filter return an error to the client:

```
ERROR 1045 (28000): Statement was aborted by an audit log filter
```

Not all events can be blocked (see Table 6.36, “Log and Abort Characteristics Per Event Class and Subclass Combination”). For an event that cannot be blocked, the audit log writes a warning to the error log rather than blocking it.

For attempts to define a filter in which the `abort` item appears elsewhere than in an `event` item, an error occurs.

Logical Operators

Logical operators (`and`, `or`, `not`) permit construction of complex conditions, enabling more advanced filtering configurations to be written. The following `log` item logs only `general` events with `general_command` fields having a specific value and length:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "or": [
            {
              "and": [
                { "field": { "name": "general_command.str",    "value": "Query" } },
                { "field": { "name": "general_command.length", "value": 5 } }
              ]
            },
            {
              "and": [
                { "field": { "name": "general_command.str",    "value": "Execute" } },
                { "field": { "name": "general_command.length", "value": 7 } }
              ]
            }
          ]
        }
      }
    }
  }
}
```

Referencing Predefined Variables

To refer to a predefined variable in a `log` condition, use a `variable` item, which takes `name` and `value` items and tests equality of the named variable against a given value:

```
"variable": {
  "name": "variable_name",
  "value": comparison_value
}
```

This is true if *`variable_name`* has the value *`comparison_value`*, false otherwise.

Example:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "variable": {
            "name": "audit_log_connection_policy_value",
            "value": "::none"
          }
        }
      }
    }
  }
}
```

Each predefined variable corresponds to a system variable. By writing a filter that tests a predefined variable, you can modify filter operation by setting the corresponding system variable, without having to redefine the filter. For example, by writing a filter that tests the value of the `audit_log_connection_policy_value` predefined variable, you can modify filter operation by changing the value of the `audit_log_connection_policy` system variable.

The `audit_log_xxx_policy` system variables are used for the legacy mode audit log (see Section 6.4.5.10, “Legacy Mode Audit Log Filtering”). With rule-based audit log filtering, those variables remain visible (for example, using `SHOW VARIABLES`), but changes to them have no effect unless you write filters containing constructs that refer to them.

The following list describes the permitted predefined variables for `variable` items:

- `audit_log_connection_policy_value`

  This variable corresponds to the value of the `audit_log_connection_policy` system variable. The value is an unsigned integer. Table 6.40, “audit_log_connection_policy_value Values” shows the permitted values and the corresponding `audit_log_connection_policy` values.

  Table 6.40 audit_log_connection_policy_value Values

Value	Corresponding audit_log_connection_policy Value
`0` or `"::none"`	`NONE`
`1` or `"::errors"`	`ERRORS`
`2` or `"::all"`	`ALL`

  The `"::xxx"` values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

- `audit_log_policy_value`

  This variable corresponds to the value of the `audit_log_policy` system variable. The value is an unsigned integer. Table 6.41, “audit_log_policy_value Values” shows the permitted values and the corresponding `audit_log_policy` values.

  Table 6.41 audit_log_policy_value Values

Value	Corresponding audit_log_policy Value
`0` or `"::none"`	`NONE`
`1` or `"::logins"`	`LOGINS`
`2` or `"::all"`	`ALL`
`3` or `"::queries"`	`QUERIES`

  The `"::xxx"` values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

- `audit_log_statement_policy_value`

  This variable corresponds to the value of the `audit_log_statement_policy` system variable. The value is an unsigned integer. Table 6.42, “audit_log_statement_policy_value Values” shows the permitted values and the corresponding `audit_log_statement_policy` values.

  Table 6.42 audit_log_statement_policy_value Values

Value	Corresponding audit_log_statement_policy Value
`0` or `"::none"`	`NONE`
`1` or `"::errors"`	`ERRORS`
`2` or `"::all"`	`ALL`

  The `"::xxx"` values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

Referencing Predefined Functions

To refer to a predefined function in a `log` condition, use a `function` item, which takes `name` and `args` items to specify the function name and its arguments, respectively:

```
"function": {
  "name": "function_name",
  "args": arguments
}
```

The `name` item should specify the function name only, without parentheses or the argument list.

The `args` item must satisfy these conditions:

- If the function takes no arguments, no `args` item should be given.
- If the function does take arguments, an `args` item is needed, and the arguments must be given in the order listed in the function description. Arguments can refer to predefined variables, event fields, or string or numeric constants.

If the number of arguments is incorrect or the arguments are not of the correct data types required by the function an error occurs.

Example:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "function": {
            "name": "find_in_include_list",
            "args": [ { "string": [ { "field": "user.str" },
                                    { "string": "@"},
                                    { "field": "host.str" } ] } ]
          }
        }
      }
    }
  }
}
```

The preceding filter determines whether to log `general` class `status` events depending on whether the current user is found in the `audit_log_include_accounts` system variable. That user is constructed using fields in the event.

The following list describes the permitted predefined functions for `function` items:

- `audit_log_exclude_accounts_is_null()`

  Checks whether the `audit_log_exclude_accounts` system variable is `NULL`. This function can be helpful when defining filters that correspond to the legacy audit log implementation.

  Arguments:

  None.

- `audit_log_include_accounts_is_null()`

  Checks whether the `audit_log_include_accounts` system variable is `NULL`. This function can be helpful when defining filters that correspond to the legacy audit log implementation.

  Arguments:

None.

- `debug_sleep(millisec)`

  Sleeps for the given number of milliseconds. This function is used during performance measurement.

  `debug_sleep()` is available for debug builds only.

  Arguments:

  - *`millisec`*: An unsigned integer that specifies the number of milliseconds to sleep.

- `find_in_exclude_list(account)`

  Checks whether an account string exists in the audit log exclude list (the value of the `audit_log_exclude_accounts` system variable).

  Arguments:

  - *`account`*: A string that specifies the user account name.

- `find_in_include_list(account)`

  Checks whether an account string exists in the audit log include list (the value of the `audit_log_include_accounts` system variable).

  Arguments:

  - *`account`*: A string that specifies the user account name.

- `query_digest([str])`

  This function has differing behavior depending on whether an argument is given:

  - With no argument, `query_digest` returns the statement digest value corresponding to the statement literal text in the current event.
  - With an argument, `query_digest` returns a Boolean indicating whether the argument is equal to the current statement digest.

  Arguments:

  - *`str`*: This argument is optional. If given, it specifies a statement digest to be compared against the digest for the statement in the current event.

  Examples:

  This `function` item includes no argument, so `query_digest` returns the current statement digest as a string:

  ```
  "function": {
    "name": "query_digest"
  }
  ```

  This `function` item includes an argument, so `query_digest` returns a Boolean indicating whether the argument equals the current statement digest:

  ```
  "function": {
    "name": "query_digest",
    "args": "SELECT ?"
  }
  ```

  This function was added in MySQL 8.0.26.

- `string_find(text, substr)`

  Checks whether the `substr` value is contained in the `text` value. This search is case-sensitive.

  Arguments:

  - *`text`*: The text string to search.
  - *`substr`*: The substring to search for in *`text`*.

Replacement of Event Field Values

As of MySQL 8.0.26, audit filter definitions support replacement of certain audit event fields, so that logged events contain the replacement value rather than the original value. This capability enables logged audit records to include statement digests rather than literal statements, which can be useful for MySQL deployments for which statements may expose sensitive values.

Field replacement in audit events works like this:

- Field replacements are specified in audit filter definitions, so audit log filtering must be enabled as described in Section 6.4.5.7, “Audit Log Filtering”.
- Not all fields can be replaced. Table 6.43, “Event Fields Subject to Replacement” shows which fields are replaceable in which event classes.

  Table 6.43 Event Fields Subject to Replacement

Event Class	Field Name
`general`	`general_query.str`
`table_access`	`query.str`

- Replacement is conditional. Each replacement specification in a filter definition includes a condition, enabling a replaceable field to be changed, or left unchanged, depending on the condition result.
- If replacement occurs, the replacement specification indicates the replacement value using a function that is permitted for that purpose.

As Table 6.43, “Event Fields Subject to Replacement” shows, currently the only replaceable fields are those that contain statement text (which occurs in events of the `general` and `table_access` classes). In addition, the only function permitted for specifying the replacement value is `query_digest`. This means that the only permitted replacement operation is to replace statement literal text by its corresponding digest.

Because field replacement occurs at an early auditing stage (during filtering), the choice of whether to write statement literal text or digest values applies regardless of log format written later (that is, whether the audit log plugin produces XML or JSON output).

Field replacement can take place at differing levels of event granularity:

- To perform field replacement for all events in a class, filter events at the class level.
- To perform replacement on a more fine-grained basis, include additional event-selection items. For example, you can perform field replacement only for specific subclasses of a given event class, or only in events for which fields have certain characteristics.

Within a filter definition, specify field replacement by including a `print` item, which has this syntax:

```
"print": {
  "field": {
    "name": "field_name",
    "print": condition,
```

```
    "replace": replacement_value
  }
}
```

Within the `print` item, its `field` item takes these three items to indicate how whether and how replacement occurs:

- `name`: The field for which replacement (potentially) occurs. *`field_name`* must be one of those shown in Table 6.43, “Event Fields Subject to Replacement”.
- `print`: The condition that determines whether to retain the original field value or replace it:
  - If *`condition`* evaluates to `true`, the field remains unchanged.
  - If *`condition`* evaluates to `false`, replacement occurs, using the value of the `replace` item.

  To unconditionally replace a field, specify the condition like this:

  ```
  "print": false
  ```

- `replace`: The replacement value to use when the `print` condition evaluates to `false`. Specify *`replacement_value`* using a `function` item.

For example, this filter definition applies to all events in the `general` class, replacing the statement literal text with its digest:

```
{
  "filter": {
    "class": {
      "name": "general",
      "print": {
        "field": {
          "name": "general_query.str",
          "print": false,
          "replace": {
            "function": {
              "name": "query_digest"
            }
          }
        }
      }
    }
  }
}
```

The preceding filter uses this `print` item to unconditionally replace the statement literal text contained in `general_query.str` by its digest value:

```
"print": {
  "field": {
    "name": "general_query.str",
    "print": false,
    "replace": {
      "function": {
        "name": "query_digest"
      }
    }
  }
}
```

`print` items can be written different ways to implement different replacement strategies. The `replace` item just shown specifies the replacement text using this `function` construct to return a string representing the current statement digest:

```
"function": {
  "name": "query_digest"
}
```

The `query_digest` function can also be used in another way, as a comparator that returns a Boolean, which enables its use in the `print` condition. To do this, provide an argument that specifies a comparison statement digest:

```
"function": {
  "name": "query_digest",
  "args": "digest"
}
```

In this case, `query_digest` returns `true` or `false` depending on whether the current statement digest is the same as the comparison digest. Using `query_digest` this way enables filter definitions to detect statements that match particular digests. The condition in the following construct is true only for statements that have a digest equal to `SELECT ?`, thus effecting replacement only for statements that do not match the digest:

```
"print": {
  "field": {
    "name": "general_query.str",
    "print": {
      "function": {
        "name": "query_digest",
        "args": "SELECT ?"
      }
    },
    "replace": {
      "function": {
        "name": "query_digest"
      }
    }
  }
}
```

To perform replacement only for statements that do match the digest, use `not` to invert the condition:

```
"print": {
  "field": {
    "name": "general_query.str",
    "print": {
      "not": {
        "function": {
          "name": "query_digest",
          "args": "SELECT ?"
        }
      }
    },
    "replace": {
      "function": {
        "name": "query_digest"
      }
    }
  }
}
```

Suppose that you want the audit log to contain only statement digests and not literal statements. To achieve this, you must perform replacement on all events that contain statement text; that is, events in the `general` and `table_access` classes. An earlier filter definition showed how to unconditionally replace statement text for `general` events. To do the same for `table_access` events, use a filter that is similar but changes the class from `general` to `table_access` and the field name from `general_query.str` to `query.str`:

```
{
  "filter": {
    "class": {
      "name": "table_access",
      "print": {
        "field": {
          "name": "query.str",
          "print": false,
```

```
          "replace": {
            "function": {
              "name": "query_digest"
            }
          }
        }
      }
    }
  }
}
```

Combining the `general` and `table_access` filters results in a single filter that performs replacement for all statement text-containing events:

```
{
  "filter": {
    "class": [
      {
        "name": "general",
        "print": {
          "field": {
            "name": "general_query.str",
            "print": false,
            "replace": {
              "function": {
                "name": "query_digest"
              }
            }
          }
        }
      },
      {
        "name": "table_access",
        "print": {
          "field": {
            "name": "query.str",
            "print": false,
            "replace": {
              "function": {
                "name": "query_digest"
              }
            }
          }
        }
      }
    ]
  }
}
```

To perform replacement on only some events within a class, add items to the filter that indicate more specifically when replacement occurs. The following filter applies to events in the `table_access` class, but performs replacement only for `insert` and `update` events (leaving `read` and `delete` events unchanged):

```
{
  "filter": {
    "class": {
      "name": "table_access",
      "event": {
        "name": [
          "insert",
          "update"
        ],
        "print": {
          "field": {
            "name": "query.str",
            "print": false,
            "replace": {
              "function": {
                "name": "query_digest"
              }
```

```
            }
          }
        }
      }
    }
  }
}
```

This filter performs replacement for `general` class events corresponding to the listed account-management statements (the effect being to hide credential and data values in the statements):

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "print": {
          "field": {
            "name": "general_query.str",
            "print": false,
            "replace": {
              "function": {
                "name": "query_digest"
              }
            }
          }
        },
        "log": {
          "or": [
            {
              "field": {
                "name": "general_sql_command.str",
                "value": "alter_user"
              }
            },
            {
              "field": {
                "name": "general_sql_command.str",
                "value": "alter_user_default_role"
              }
            },
            {
              "field": {
                "name": "general_sql_command.str",
                "value": "create_role"
              }
            },
            {
              "field": {
                "name": "general_sql_command.str",
                "value": "create_user"
              }
            }
          ]
        }
      }
    }
  }
}
```

For information about the possible `general_sql_command.str` values, see Testing Event Field Values.

Replacing a User Filter

In some cases, the filter definition can be changed dynamically. To do this, define a `filter` configuration within an existing `filter`. For example:

```
{
```

```
  "filter": {
    "id": "main",
    "class": {
      "name": "table_access",
      "event": {
        "name": [ "update", "delete" ],
        "log": false,
        "filter": {
          "class": {
            "name": "general",
            "event" : { "name": "status",
                        "filter": { "ref": "main" } }
          },
          "activate": {
            "or": [
              { "field": { "name": "table_name.str", "value": "temp_1" } },
              { "field": { "name": "table_name.str", "value": "temp_2" } }
            ]
          }
        }
      }
    }
  }
}
```

A new filter is activated when the `activate` item within a subfilter evaluates to `true`. Using `activate` in a top-level `filter` is not permitted.

A new filter can be replaced with the original one by using a `ref` item inside the subfilter to refer to the original filter `id`.

The filter shown operates like this:

- The `main` filter waits for `table_access` events, either `update` or `delete`.
- If the `update` or `delete` `table_access` event occurs on the `temp_1` or `temp_2` table, the filter is replaced with the internal one (without an `id`, since there is no need to refer to it explicitly).
- If the end of the command is signalled (`general` / `status` event), an entry is written to the audit log file and the filter is replaced with the `main` filter.

The filter is useful to log statements that update or delete anything from the `temp_1` or `temp_2` tables, such as this one:

```
UPDATE temp_1, temp_3 SET temp_1.a=21, temp_3.a=23;
```

The statement generates multiple `table_access` events, but the audit log file contains only `general` / `status` entries.

Note

Any `id` values used in the definition are evaluated with respect only to that definition. They have nothing to do with the value of the `audit_log_filter_id` system variable.

6.4.5.9 Disabling Audit Logging

The `audit_log_disable` variable, introduced in MySQL 8.0.28, permits disabling audit logging for all connecting and connected sessions. The `audit_log_disable` variable can be set in a MySQL Server option file, in a command-line startup string, or at runtime using a `SET` statement; for example:

```
SET GLOBAL audit_log_disable = true;
```

Setting `audit_log_disable` to true disables the audit log plugin. The plugin is re-enabled when `audit_log_disable` is set back to `false`, which is the default setting.

Starting the audit log plugin with `audit_log_disable = true` generates a warning (`ER_WARN_AUDIT_LOG_DISABLED`) with the following message: `Audit Log is disabled. Enable it with audit_log_disable = false`. Setting `audit_log_disable` to false also generates warning. When `audit_log_disable` is set to true, audit log function calls and variable changes generate a session warning.

Setting the runtime value of `audit_log_disable` requires the `AUDIT_ADMIN` privilege, in addition to the `SYSTEM_VARIABLES_ADMIN` privilege (or the deprecated `SUPER` privilege) normally required to set a global system variable runtime value.

6.4.5.10 Legacy Mode Audit Log Filtering

Note

This section describes legacy audit log filtering, which applies if the `audit_log` plugin is installed without the accompanying audit tables and functions needed for rule-based filtering.

The audit log plugin can filter audited events. This enables you to control whether audited events are written to the audit log file based on the account from which events originate or event status. Status filtering occurs separately for connection events and statement events.

- Legacy Event Filtering by Account
- Legacy Event Filtering by Status

Legacy Event Filtering by Account

To filter audited events based on the originating account, set one (not both) of the following system variables at server startup or runtime. These variables apply only for legacy audit log filtering.

- `audit_log_include_accounts`: The accounts to include in audit logging. If this variable is set, only these accounts are audited.
- `audit_log_exclude_accounts`: The accounts to exclude from audit logging. If this variable is set, all but these accounts are audited.

The value for either variable can be `NULL` or a string containing one or more comma-separated account names, each in *`user_name@host_name`* format. By default, both variables are `NULL`, in which case, no account filtering is done and auditing occurs for all accounts.

Modifications to `audit_log_include_accounts` or `audit_log_exclude_accounts` affect only connections created subsequent to the modification, not existing connections.

Example: To enable audit logging only for the `user1` and `user2` local host accounts, set the `audit_log_include_accounts` system variable like this:

```
SET GLOBAL audit_log_include_accounts = 'user1@localhost,user2@localhost';
```

Only one of `audit_log_include_accounts` or `audit_log_exclude_accounts` can be non-`NULL` at a time:

- If you set `audit_log_include_accounts`, the server sets `audit_log_exclude_accounts` to `NULL`.
- If you attempt to set `audit_log_exclude_accounts`, an error occurs unless `audit_log_include_accounts` is `NULL`. In this case, you must first clear `audit_log_include_accounts` by setting it to `NULL`.

```
-- This sets audit_log_exclude_accounts to NULL
SET GLOBAL audit_log_include_accounts = value;
```