CHAPTER 23
Security

23.0. Introduction

This chapter covers security-related topics:

- The `mysql.user` table that contains MySQL account information
- Statements for managing MySQL user accounts
- Password strength checking and policy
- Password expiration
- Finding and fixing insecure accounts
- Finding and removing anonymous accounts and accounts that permit connections from many hosts

If you like, you can skip over the initial section that describes the `mysql.user` table, but I think you'll find that reading it will help you better understand later sections, which often discuss how SQL operations map onto underlying changes in that table.

Scripts shown in this chapter are located in the *routines* directory of the `recipes` distribution.

Whether you use the MySQL 5.5, 5.6, or 5.7 release series, it is best to use a recent version within the series. Changes to the authentication system occur in early development versions that may produce results that differ from the descriptions here.

www.it-ebooks.info

Many of the techniques shown here require administrative access, such as the ability to modify tables in the `mysql` system database or use statements that require the `SUPER` privilege. For this reason, to carry out the operations described here, connect to the server as `root` rather than as `cbuser`.

23.1. Understanding the mysql.user Table

MySQL stores user account information in tables in the `mysql` system database. The `user` table is the most important because it contains account names and credentials. To see its structure, use this statement:

```
SHOW CREATE TABLE mysql.user;
```

The `user` table columns that concern us here specify account names and authentication information:

- The `User` and `Host` columns identify the account. MySQL account names comprise a combination of username and hostname values. For example, in the `user` table row for a `'cbuser'@'localhost'` account, the `User` and `Host` column values are `cbuser` and `localhost`, respectively. For a `'myuser'@'myhost.example.com'` account, those columns are `myuser` and `myhost.example.com`.
- The `plugin`, `Password`, and `authentication_string` columns store authentication credentials. MySQL does not store literal passwords in the `user` sytem table because that is insecure. Instead, the server computes a hash value from the password and stores the hash string.
  - The `plugin` column indicates which authentication plugin the server uses to check credentials for clients that attempt to use the account. Different plug-ins implement password hashing methods of varying encryption strength. The following table shows the plug-ins this chapter discusses:

Plug-in	Authentication method
`mysql_native_password`	Native password hashing
`mysql_old_password`	"Old" native password hashing (deprecated)
`sha256_password`	SHA-256 password hashing (MySQL 5.6.6 or later)

    MySQL Enterprise, the commercial version of MySQL, includes additional plug-ins for authenticating using PAM or Windows credentials. These enable use of passwords external to MySQL, such as Unix login passwords or native Windows services.
  - The `Password` column is used if the `plugin` column is `mysql_native_password` or `mysql_old_password`. An empty `Password` value means "no password,"

www.it-ebooks.info

which is insecure. A nonempty value represents a hashed password in the format required by the respective plug-in.

— The `authentication_string` column is for use by plug-ins that do not use the `Password` column. For example, `sha256_password` uses `authentication_string` to store SHA-256 password hash values, which are cryptographically superior to native hashing but too long for the `Password` column.

Before MySQL 5.7.2, the server permits the `plugin` value to be empty. As of MySQL 5.7.2, the `plugin` column *must* be nonempty and the server disables any empty-plug-in account until a nonempty plug-in is assigned. However, even before 5.7.2, it's best if every account has a nonempty value:

- If the `plugin` column for an account is empty, the server authenticates clients using either `mysql_native_password` or `mysql_old_password` implicitly, making the choice based on the format of the hash value stored in the `Password` column. A nonempty plug-in makes the authentication method explicit.
- If you're running a version older than 5.7, modifying all accounts now to have a nonempty `plugin` value helps avoid issues when you upgrade to 5.7.

If your `user` table contains accounts that have an empty `plugin` value or use the deprecated `mysql_old_password` plug-in, you can fix them. Recipe 23.8 provides upgrade instructions.

23.2. Managing User Accounts

Problem

You are responsible for setting up accounts on your MySQL server.

Solution

Learn to use the account-management SQL statements.

Discussion

It's possible to modify the grant tables in the `mysql` database directly with SQL statements such as `INSERT` or `UPDATE`, but the MySQL account-management statements are more convenient. This section describes their use and covers these topics:

- Creating accounts (`CREATE USER`, `SET PASSWORD`)
- Assigning and checking privileges (`GRANT`, `REVOKE`, `SHOW GRANTS`)

www.it-ebooks.info

- Removing and renaming accounts (`DROP USER`, `RENAME USER`)

Creating accounts

To create an account, use the `CREATE USER` statement, which creates a row in the `mysql.user` table. But before you do so, decide these three things:

- The account name, expressed in '*user_name*'@'*host_name*' format naming the user and the host from which the user will connect
- The account password
- The authentication plug-in the server should execute when clients attempt to use the account

Authentication plug-ins use hashing to encrypt passwords for storage and transmission. MySQL has several built-in plug-ins from which to choose:

- `mysql_native_password` implements the default password hashing method.
- `mysql_old_password` is similar but uses a hashing method that is less secure and is now deprecated. Avoid choosing this plug-in for new accounts. If your server has existing accounts that use it, Recipe 23.8 discusses how to identify and modify them to use `mysql_native_password` instead.
- `sha256_password` authenticates using SHA-256 password hash values, which are cryptographically more secure than hashes generated by `mysql_native_password`. This plug-in is available as of MySQL 5.6.6. It provides security beyond that afforded by `mysql_native_password`, but additional setup is required to use it. (Clients must connect using SSL or provide an RSA certificate.)

The `CREATE USER` statement is commonly used in one of these forms:

```
CREATE USER 'user_name'@'host_name' IDENTIFIED BY 'password';
CREATE USER 'user_name'@'host_name' IDENTIFIED WITH 'auth_plugin';
```

The first syntax creates the account and sets its password with a single statement. It also assigns an authentication plug-in implicitly, sort of:

- Before MySQL 5.6.6, the statement leaves the `plugin` column empty in the `user` table row for the account. It's preferable for the plug-in to be nonempty, for reasons discussed in Recipe 23.1.
- As of 5.6.6, the statement assigns the plug-in named by the `--default-authentication-plugin` setting (which is `mysql_native_password`, unless you change it at server startup).

www.it-ebooks.info

With the second syntax, you must set the password separately using a subsequent SET PASSWORD statement, but because you specify the plug-in explicitly, it's always clear which one the user table row for the account will contain.

To create an account in a way that works consistently for any version of MySQL from 5.5 or later to ensure a designated nonempty plugin value, use this approach:

1. Create the account using a CREATE USER statement that names the authentication plug-in explicitly. Also, set the old_passwords system variable to select the password hashing method appropriate for the plug-in (this affects the PASSWORD() function in the next step). The following sequences show how to do this for each plug-in:

```
CREATE USER 'user_name'@'host_name' IDENTIFIED WITH 'mysql_native_password';
SET old_passwords = 0;

CREATE USER 'user_name'@'host_name' IDENTIFIED WITH 'mysql_old_password';
SET old_passwords = 1;

CREATE USER 'user_name'@'host_name' IDENTIFIED WITH 'sha256_password';
SET old_passwords = 2;
```

2. Set the account password:

```
SET PASSWORD FOR 'user_name'@'host_name' = PASSWORD('password');
```

   The PASSWORD() function hashes the password according to the old_passwords value just specified.

To assign privileges to the new account, which has none initially, use the GRANT statement described later in this section.

CREATE USER fails if the account already exists.

Writing an account-creation helper procedure

To make it easier to create new accounts, we can write a helper stored procedure named create_user() that does all the work, given the account username, hostname, password, and authentication plug-in:

- It issues the proper CREATE USER statement to specify the plug-in explicitly.
- It sets the password, or, if the password is given as NULL, leaves the password unset. (Presumably you'd specify NULL if you intend to assign the password later.)
- Before setting the password, it takes care of setting the old_passwords system variable to the appropriate value for the specified plug-in. It also saves and restores the current old_passwords value, to leave its value in your session undisturbed.
- To implement a policy that users must select their own password, it uses ALTER USER to expire the password immediately. The procedure skips this part if ALTER USER is

www.it-ebooks.info

not available (the server is older than MySQL 5.6.7) or the account is for an anonymous user (who cannot set the account password to unexpire it). For more information about password expiration, see Recipe 23.5.

To use the procedure, invoke it like this:

```
CALL create_user('user_name','host_name','password','auth_plugin');
```

The procedure definition is shown following. It requires the helper routines `exec_stmt()` and `server_version()` from Recipes 9.9 and 10.9. Scripts to create these routines are located in the *routines* directory of the `recipes` distribution:

```
CREATE PROCEDURE create_user(user TEXT, host TEXT,
                             password TEXT, plugin TEXT)
BEGIN
  DECLARE account TEXT;
  SET account = CONCAT(QUOTE(user),'@',QUOTE(host));
  CALL exec_stmt(CONCAT('CREATE USER ',account,
                        ' IDENTIFIED WITH ',QUOTE(plugin)));
  IF password IS NOT NULL THEN
    BEGIN
      DECLARE saved_old_passwords INT;
      SET saved_old_passwords = @@old_passwords;
      CASE plugin
        WHEN 'mysql_native_password' THEN SET old_passwords = 0;
        WHEN 'mysql_old_password' THEN SET old_passwords = 1;
        WHEN 'sha256_password' THEN SET old_passwords = 2;
        ELSE SIGNAL SQLSTATE 'HY000'
                    SET MYSQL_ERRNO = 1525,
                        MESSAGE_TEXT = 'unhandled auth plugin';
      END CASE;
      CALL exec_stmt(CONCAT('SET PASSWORD FOR ',account,
                            ' = PASSWORD(',QUOTE(PASSWORD),')'));
      SET old_passwords = saved_old_passwords;
    END;
  END IF;
  IF server_version() >= 50607 AND user <> '' THEN
    CALL exec_stmt(CONCAT('ALTER USER ',account,' PASSWORD EXPIRE'));
  END IF;
END;
```

Assigning and checking privileges

Suppose that you have just created an account named `'user1'@'localhost'`. You can assign privileges to it with `GRANT`, remove privileges from it with `REVOKE`, and check its privileges with `SHOW GRANTS`.

`GRANT` has this syntax:

```
GRANT privileges ON scope TO account;
```


www.it-ebooks.info

Here, *account* names the account to be granted the privileges, *privileges* indicates what they are, and *scope* indicates the privilege scope, or level at which they apply. The *privileges* value can be ALL (or ALL PRIVILEGES) to specify all privileges available at the given level, or a comma-separated list of one or more privilege names such as SELECT or CREATE. (For a full discussion of available privileges and GRANT syntax not shown here, see the *MySQL Reference Manual*.)

The following examples illustrate the syntax for granting privileges at each level.

- Granting privileges globally enables the account to perform administrative operations or operations on any database:

```
GRANT FILE ON *.* TO 'user1'@'localhost';
GRANT CREATE TEMPORARY TABLES, LOCK TABLES ON *.* TO 'user1'@'localhost';
```

- Granting privileges at the database level enables the account to perform operations on objects within the named database:

```
GRANT ALL ON cookbook.* TO 'user1'@'localhost';
```

- Granting privileges at the table level enables the account to perform operations on the named table:

```
GRANT SELECT ON mysql.user TO 'user1'@'localhost';
```

- Granting privileges at the column level enables the account to perform operations on the named table column:

```
GRANT SELECT(User,Host), UPDATE(password_expired)
ON mysql.user TO 'user1'@'localhost';
```

- Granting privileges at the procedure level enables the account to perform operations on the named stored procedure:

```
GRANT EXECUTE ON PROCEDURE cookbook.exec_stmt TO 'user1'@'localhost';
```

  Use FUNCTION rather than PROCEDURE if the routine is a stored function.

To verify the privilege assignments, use SHOW GRANTS:

```
mysql> SHOW GRANTS FOR 'user1'@'localhost';
+-----------------------------------------------------------------------+
| Grants for user1@localhost                                            |
+-----------------------------------------------------------------------+
| GRANT FILE, CREATE TEMPORARY TABLES, LOCK TABLES                      |
| ON *.* TO 'user1'@'localhost'                                         |
| GRANT ALL PRIVILEGES ON `cookbook`.* TO 'user1'@'localhost'           |
| GRANT SELECT, SELECT (User, Host), UPDATE (password_expired)          |
| ON `mysql`.`user` TO 'user1'@'localhost'                              |
| GRANT EXECUTE ON PROCEDURE `cookbook`.`exec_stmt` TO 'user1'@'localhost' |
+-----------------------------------------------------------------------+
```

To see your own privileges, omit the FOR clause.

www.it-ebooks.info

REVOKE syntax is generally similar to GRANT but uses FROM rather than TO:

```
REVOKE privileges ON scope FROM account;
```

Thus, to remove the privileges just granted to 'user1'@'localhost', use these REVOKE statements (and SHOW GRANTS to verify that they were removed):

```
mysql> REVOKE FILE ON *.* FROM 'user1'@'localhost';
mysql> REVOKE CREATE TEMPORARY TABLES, LOCK TABLES
    -> ON *.* FROM 'user1'@'localhost';
mysql> REVOKE ALL ON cookbook.* FROM 'user1'@'localhost';
mysql> REVOKE SELECT ON mysql.user FROM 'user1'@'localhost';
mysql> REVOKE SELECT(User,Host), UPDATE(password_expired)
    -> ON mysql.user FROM 'user1'@'localhost';
mysql> REVOKE EXECUTE ON PROCEDURE cookbook.exec_stmt
    -> FROM 'user1'@'localhost';
mysql> SHOW GRANTS FOR 'user1'@'localhost';
+----------------------------------------+
| Grants for user1@localhost             |
+----------------------------------------+
| GRANT USAGE ON *.* TO 'user1'@'localhost' |
+----------------------------------------+
```

Removing accounts

To get rid of an account, use the DROP USER statement:

```
DROP USER 'user1'@'localhost';
```

The statement removes all rows associated with the account in all grant tables; you need not use REVOKE to remove its privileges first. An error occurs if the account does not exist.

Renaming accounts

To change an account name, use RENAME USER, specifying the current and new names:

```
RENAME USER 'currentuser'@'localhost' TO 'newuser'@'localhost';
```

An error occurs if the current account does not exist or the new account already exists.

23.3. Implementing a Password Policy

Problem

You want to ensure that MySQL accounts do not use weak passwords.

www.it-ebooks.info

Solution

Use the `validate_password` plug-in to implement a password policy. New passwords must satisfy the policy, whether those chosen by the DBA for new accounts or by existing users changing their password.

Discussion

This technique requires the `validate_password` plug-in to be enabled. For plug-in installation instructions, see Recipe 22.2.

When `validate_password` is enabled, it exposes a set of system variables that enable you to configure it. These are the default values:

```
mysql> SHOW VARIABLES LIKE 'validate_password%';
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| 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      |
+--------------------------------------+--------+
```

Suppose that you want to implement a policy that enforces these requirements for passwords:

- At least 10 characters long
- Contains uppercase and lowercase characters
- Contains at least two digits
- Contains at least one special (nonalphanumeric) character

To put that policy in place, start the server with options that enable the plug-in and set the values of the system variables that configure the policy requirements. For example, put these lines in your server option file:

```
[mysqld]
plugin-load-add=validate_password.so
validate_password_length=10
validate_password_mixed_case_count=1
validate_password_number_count=2
validate_password_special_char_count=1
```

After starting the server, verify the settings:

www.it-ebooks.info

```
mysql> SHOW VARIABLES LIKE 'validate_password%';
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| validate_password_dictionary_file    |        |
| validate_password_length             | 10     |
| validate_password_mixed_case_count   | 1      |
| validate_password_number_count       | 2      |
| validate_password_policy             | MEDIUM |
| validate_password_special_char_count | 1      |
+--------------------------------------+--------+
```

Now the `validate_password` plug-in prevents assigning passwords too weak for the policy:

```
mysql> SET PASSWORD = PASSWORD('weak-password');
ERROR 1819 (HY000): Your password does not satisfy the current
policy requirements
mysql> SET PASSWORD = PASSWORD('Str0ng-Pa33w@rd');
Query OK, 0 rows affected (0.00 sec)
```

The preceding instructions leave the `validate_password_policy` system variable set to its default value (`MEDIUM`), but you can change it to control how the server tests passwords:

- `MEDIUM` enables tests for password length and the number of numeric, uppercase/lowercase, and special characters.
- To be less rigorous, set the policy to `LOW`, which enables only the length test. To also permit shorter passwords, decrease the required length (`validate_password_length`).
- To be more rigorous, set the policy to `STRONG`, which is like `MEDIUM` but also enables you to have passwords checked against a dictionary file, to prevent use of passwords that match any word in the file. Comparisons are not case sensitive.

  To use a dictionary file, set the value of `validate_password_dictionary_file` to the filename at server startup. The file should contain lowercase words, one per line. MySQL distributions include a *dictionary.txt* file in the *share* directory that you can use, and Unix systems often have a */usr/share/dict/words* file.

Putting a password policy in place has no effect on existing passwords. To require users to choose a new password that satisfies the policy, expire their current password (see Recipe 23.5).

www.it-ebooks.info

23.4. Checking Password Strength

Problem

You want to assign or change a password but verify first that it's not weak.

Solution

Use the `VALIDATE_PASSWORD_STRENGTH()` function.

Discussion

The `validate_password` plug-in not only implements policy for new passwords, it provides a SQL function, `VALIDATE_PASSWORD_STRENGTH()`, that enables strength testing of prospective passwords. Uses for this function include:

- An administrator wants to check passwords to be assigned to new accounts.
- An individual user wants to choose a new password but seeks assurance in advance how strong it is.

To use `VALIDATE_PASSWORD_STRENGTH()`, the `validate_password` plug-in must be enabled. For plug-in installation instructions, see Recipe 22.2.

`VALIDATE_PASSWORD_STRENGTH()` returns a value from 0 (weak) to 100 (strong):

```
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('abc') ;
+----------------------------------+
| VALIDATE_PASSWORD_STRENGTH('abc') |
+----------------------------------+
|                                0 |
+----------------------------------+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('weak-password');
+-------------------------------------------+
| VALIDATE_PASSWORD_STRENGTH('weak-password') |
+-------------------------------------------+
|                                        50 |
+-------------------------------------------+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('Str0ng-Pa33w@rd');
+----------------------------------------------+
| VALIDATE_PASSWORD_STRENGTH('Str0ng-Pa33w@rd') |
+----------------------------------------------+
|                                          100 |
+----------------------------------------------+
```


www.it-ebooks.info

23.5. Expiring Passwords

Problem

You want users to pick a new MySQL password.

Solution

The `ALTER USER` statement expires passwords.

Discussion

MySQL 5.6.7 and up provides an `ALTER USER` statement that enables an administrator to expire an account's password:

```
ALTER USER 'cbuser'@'localhost' PASSWORD EXPIRE;
```

Here are some uses for password expiration:

- You can implement a policy that new users must select a new password when first connecting: immediately expire the password for each new account you create.
- If you impose a stricter policy on acceptable passwords (see Recipe 23.3), you can expire all existing passwords to require each user to choose a new one that meets the more stringent requirements.

`ALTER USER` affects a single account. It works by setting the `password_expired` column to `Y` for the appropriate `mysql.user` row. To "cheat" and expire passwords for all non-anonymous accounts at once, do this (anonymous users cannot reset their password, so expiring those would be unfriendly):

```
UPDATE mysql.user SET password_expired = 'Y' WHERE User <> '';
FLUSH PRIVILEGES;
```

Alternatively, to affect all accounts but avoid modifying the grant tables directly, use a stored procedure that loops through all accounts and executes `ALTER USER` for each:

```
CREATE PROCEDURE expire_all_passwords()
BEGIN
  DECLARE done BOOLEAN DEFAULT FALSE;
  DECLARE account TEXT;
  DECLARE cur CURSOR FOR
    SELECT CONCAT(QUOTE(User),'@',QUOTE(Host)) AS account
    FROM mysql.user WHERE User <> '';
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur;
  expire_loop: LOOP
    FETCH cur INTO account;
    IF done THEN
```


www.it-ebooks.info

```
        LEAVE expire_loop;
      END IF;
      CALL exec_stmt(CONCAT('ALTER USER ',account,' PASSWORD EXPIRE'));
    END LOOP;
    CLOSE cur;
  END;
```

The procedure requires the `exec_stmt()` helper routine (see Recipe 9.9). Scripts to create these routines are located in the *routines* directory of the `recipes` distribution.

23.6. Assigning Yourself a New Password

Problem

You want to change your password.

Solution

Use the `SET PASSWORD` statement.

Discussion

To assign yourself a new password, use the `SET PASSWORD` statement and the `PASSWORD()` function:

```
SET PASSWORD = PASSWORD('my-new-password');
```

`SET PASSWORD` permits a `FOR` clause that enables you to specify which account gets the new password:

```
SET PASSWORD FOR 'user_name'@'host_name' = PASSWORD('my-new-password');
```

This latter syntax is primarily for DBAs because it requires the `UPDATE` privilege for the `mysql` database.

If `SET PASSWORD` complains about the password hash being in the wrong format, try again after setting `old_passwords` to select the hashing method appropriate for the authentication plug-in associated with your account. Recipe 23.2 provides these values.

To check the strength of a password you're considering, use the `VALIDATE_PASSWORD_STRENGTH()` function (see Recipe 23.4).

23.7. Resetting an Expired Password

Problem

You cannot use MySQL because your DBA expired your password.

www.it-ebooks.info

Solution

Assign yourself a new password.

Discussion

If the MySQL administrator has expired your password, MySQL will let you connect, but not do much of anything else:

```
% mysql --user=cbuser --password
Enter password: ******
mysql> SELECT CURRENT_USER();
ERROR 1820 (HY000): You must SET PASSWORD before executing this statement
```

If you see that message, reset your password so that you can work normally again:

```
mysql> SET PASSWORD = PASSWORD('my-new-password');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CURRENT_USER(); -- now you can work again
+------------------+
| CURRENT_USER()   |
+------------------+
| cbuser@localhost |
+------------------+
1 row in set (0.00 sec)
```

Technically, MySQL does not require a *new* password to replace an expired password, so you can assign yourself your current password to unexpire it. The exception is that if the password policy has become more restrictive and your current password no longer satisfies it, a stronger password must be chosen.

For more information about changing your password, see Recipe 23.6.

23.8. Finding and Fixing Insecure Accounts

Problem

Your MySQL installation includes accounts that have no password or use deprecated and insecure password hashing.

Solution

Upgrade those accounts to use a better password hashing method.

Discussion

Security is important and MySQL has improved user account security over time. An early change occurred way back in MySQL 4.1, with the introduction of a better pass-

www.it-ebooks.info

word hashing method than the original pre-4.1 method. (MySQL does not store literal passwords in the `mysql.user` system table because that is insecure. Instead, the server computes a hash value from the password and stores the hash string.) More recent authentication changes include the introduction in MySQL 5.6 of the `sha256_password` plug-in that implements SHA-256 password hashing and the `validate_password` plug-in that implements password policy and password strength assessment. This section describes characteristics of the 4.1 and (less secure) original hashing methods and shows how to upgrade accounts that use the original method so they use the 4.1 method instead. For information about the `sha256_password` and `validate_password` plug-ins, see Recipes 23.2 and 23.4.

For any account with a nonempty `Password` value in its `user` table row, you can tell which hashing method generated it:

- The hashing method introduced in MySQL 4.1 produces 41-character hash values beginning with a `*` character. This is the "4.1" or "native" hashing method. For accounts that have this type of password hash, the server authenticates connection attempts using the `mysql_native_password` plug-in.
- The original hashing method produces 16-character hash values. This is the "pre-4.1" or "old" hashing method. The server authenticates accounts that have this type of password hash using the `mysql_old_password` authentication plug-in.

To see the difference between the two hash formats, generate hash values explicitly:

```
mysql> SET old_passwords = 0;
mysql> SELECT OLD_PASSWORD('mypass') AS old, PASSWORD('mypass') AS new\G
*************************** 1. row ***************************
old: 6f8c114b58f2ce9e
new: *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4
```

The example sets `old_passwords` to 0 because `PASSWORD()` uses the pre-4.1 hashing method and returns the same result as `OLD_PASSWORD()` if `old_passwords` is set to 1.

Administrators should avoid creating accounts that use the older, less secure pre-4.1 hashing method. If your MySQL installation has accounts that have old password hashes, you can upgrade them to use the 4.1 hashing method. (This will become necessary eventually, anyway. Pre-4.1 hashing is deprecated as of MySQL 5.6 and support for it will be dropped at some point.)

Additionally, each account should have a nonempty password.

To identify and upgrade insecure accounts, use this procedure:

1. Determine whether your `user` table contains accounts with weak security. A "weak" account has either of these characteristics:

www.it-ebooks.info

- The `plugin` column is `mysql_native_password` but the `Password` column is empty.
- The `plugin` column is empty or `mysql_old_password`. (If the value is empty, the server authenticates clients using either `mysql_native_password` or `mysql_old_password`, making the choice based on the hash format of the value stored in the `Password` column. To prevent the possibility of implicit authentication using `mysql_old_password`, set the plug-in to `mysql_native_password`.)

Use this query to find weak accounts with those characteristics:

```
SELECT User, Host, plugin, Password FROM mysql.user
WHERE (plugin = 'mysql_native_password' AND Password = '')
      OR plugin IN ('','mysql_old_password');
```

2. Before upgrading a weak account, consider whether the account is even necessary. Perhaps it was created long ago for a project that's no longer used and you can simply remove it:

```
DROP USER 'olduser'@'localhost';
```

   The result is one less account to be protected and one less point of exploit.

3. If a weak account must be retained, upgrade it:

   - If the plug-in is empty or `mysql_old_password`, change it to `mysql_native_password` so that pre-4.1 password hashing cannot be used.
   - If the password is empty or in pre-4.1 hash format, assign a new password using 4.1 hashing.

Suppose that a server's user population includes accounts with the following authentication characteristics, most of which need improvement. (All have a `Host` value of `localhost`, although it's not shown here.)

```
mysql> SELECT User, plugin, Password FROM mysql.user
    -> WHERE User LIKE 'user%' AND Host = 'localhost' ORDER BY User;
+-------+-----------------------+-------------------------------------------+
| User  | plugin                | Password                                  |
+-------+-----------------------+-------------------------------------------+
| user1 | mysql_native_password | *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
| user2 |                       |                                           |
| user3 | mysql_old_password    |                                           |
| user4 |                       | 6f8c114b58f2ce9e                          |
| user5 | mysql_old_password    | 6f8c114b58f2ce9e                          |
| user6 | mysql_native_password |                                           |
| user7 |                       | *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------+-----------------------+-------------------------------------------+
```

The requirements for better security are that each account names the `mysql_native_password` plug-in explicitly and has a nonempty password in 4.1 hash format.

www.it-ebooks.info

Measured against those requirements, only the `user1` account has acceptable values. (It's the only account not selected by the "identify weak accounts" query shown earlier.) Each of the other accounts is deficient in some way. The following instructions describe how to address their weaknesses.

In general, it's preferable to manipulate MySQL accounts using SQL statements intended for that purpose, such as `CREATE USER` or `SET PASSWORD`, and to avoid modifying the grant tables directly using statements such as `INSERT` or `UPDATE`. But some operations are more straightforward using direct manipulation (and sometimes not possible to perform otherwise), so the following instructions include some direct modifications of the `user` table, even though that goes against convention. A consequence of direct manipulation is that `FLUSH PRIVILEGES` is required following `UPDATE`, to ensure that the server refreshes the account information it caches in memory.

For each account for which you reassign the password, you must either know the current password or assign a temporary password. In the latter case, contact the account owner, provide the temporary password, and ask the owner to choose a new one.

Begin by setting the `old_passwords` system variable to 0, to ensure that `PASSWORD()` uses the 4.1 hashing method, not the pre-4.1 method:

```
SET old_passwords = 0;
```

That done, upgrade each account per its particular weaknesses. Note that the `UPDATE` statements specify both `User` and `Host` (not just `User`) to uniquely identify the single account to update:

- `user1` weaknesses: None. The account specifies the native plug-in explicitly and the password is nonempty in 4.1 hash format. Actions: None needed.
- `user2` through `user5` have different weaknesses, but in each case the statements to implement the required security upgrade are the same:
  - `user2` weaknesses: No plug-in named; password is empty. Actions: Specify the native plug-in; assign a password.
  - `user3` weaknesses: Uses the old plug-in; password is empty. Actions: Change to the native plug-in; assign a password.
  - `user4` weaknesses: No plug-in named; password uses pre-4.1 hash. Actions: Specify the native plug-in; upgrade password to 4.1 hash.
  - `user5` weaknesses: Uses the old plug-in; password uses pre-4.1 hash. Actions: Change to the native plug-in; upgrade password to 4.1 hash.

  To address the issues for any of `user2` through `user5`, use the following statements (substituting the proper username for `user2` as necessary):

www.it-ebooks.info

```
UPDATE mysql.user
SET plugin = 'mysql_native_password', Password = PASSWORD('mypass')
WHERE User = 'user2' AND Host = 'localhost';
FLUSH PRIVILEGES;
```

- `user6` weakness: Password is empty. Action: Assign a password.

```
SET PASSWORD FOR 'user6'@'localhost' = PASSWORD('mypass');
```

- `user7` weakness: No plug-in named. Action: Specify the native plug-in.

```
UPDATE mysql.user
SET plugin = 'mysql_native_password'
WHERE User = 'user7' AND Host = 'localhost';
FLUSH PRIVILEGES;
```

23.9. Disabling Use of Accounts with Pre-4.1 Passwords

Problem

The original pre-4.1 hashing method is less secure than other methods and you want to prevent accounts from using it.

Solution

Set the `secure_auth` system variable to prevent such accounts from connecting to the server. To be more user friendly, upgrade affected accounts first.

Discussion

The hashing method used by the `mysql_old_password` authentication plug-in is not as secure as the method used by `mysql_native_password`. In addition, `mysql_old_password` is deprecated and eventually will no longer be supported. To prevent its use and prepare for the day when support for it ceases, take these steps:

1. Identify accounts that use `mysql_old_password` and upgrade them to use `mysql_native_password` (see Recipe 23.8). Do this first so as not to lock out accounts in the next step.
2. Start the server with the `secure_auth` system variable enabled. That's been the default value since MySQL 5.6.5, but you can check whether your server's setting differs:

```
mysql> SELECT @@secure_auth;
+---------------+
| @@secure_auth |
+---------------+
|             0 |
+---------------+
```


www.it-ebooks.info

If the value is 0, enable the variable by starting the server with the value set to 1. For example, use these lines in an option file:

```
[mysqld]
secure_auth=1
```

At this point, accounts that use pre-4.1 password hashes can no longer connect.

23.10. Finding and Removing Anonymous Accounts

Problem

You want to ensure that your MySQL server can be used only by accounts associated with specific usernames.

Solution

Identify and remove anonymous accounts.

Discussion

An "anonymous" account is one that has an empty user part in the account name, such as `''@'localhost'`. An empty user matches any name because the purpose of an anonymous account is to permit anyone who knows its password to connect from the named host (`localhost` in this case). This is a convenience because the DBA need not set up individual accounts for separate users. But there are security implications as well:

- Such accounts often are given no password, enabling their use with no authentication at all.
- You cannot associate database activity with specific users (for example, by checking the server query log or examining `SHOW PROCESSLIST` output), making it more difficult to tell who is doing what.

If the preceding points persuade you that anonymous accounts are not a good thing, use the following instructions to identify and remove them:

1. The `User` column is empty in the `mysql.user` rows for anonymous accounts, so you can identify them like this:

```
mysql> SELECT User, Host FROM mysql.user WHERE User = '';
+------+---------------+
| User | Host          |
+------+---------------+
|      | %.example.com |
|      | localhost     |
+------+---------------+
```


www.it-ebooks.info

2. The SELECT output shows two anonymous accounts. Remove each using a DROP USER statement with the corresponding account name:

```
mysql> DROP USER ''@'localhost';
mysql> DROP USER ''@'%.example.com';
```

23.11. Modifying "Any Host" and "Many Host" Accounts

Problem

You want to ensure that MySQL accounts cannot be used from an overly broad set of hosts.

Solution

Find and fix accounts containing % or _ in the host part.

Discussion

The host part of MySQL account names can contain the SQL pattern characters % and _ (see Recipe 5.8). These names match client connection attempts from any host that matches the pattern. For example, the account `'user1'@'%'` permits `user1` to connect from any host whatsoever, and `'user2'@'%.example.com'` permits `user2` to connect from any host in the `example.com` domain.

Patterns in the host part of account names provide a convenience that enables a DBA to create an account that permits connections from multiple hosts. They correspondingly increase security risks by increasing the number of hosts from which intruders can attempt to connect. If you consider this a concern, identify the accounts and either remove them or change the host part to be more specific.

There are several ways to find accounts with % or _ in the host part. Here are two:

```
WHERE Host LIKE '%\%%' OR Host LIKE '%\_%';
WHERE Host REGEXP '[%_]';
```

The LIKE expression is more complex because we must look for each pattern character separately and escape it to search for literal instances. The REGEXP expression requires no escaping because those characters are not special in regular expressions, and a character class permits both to be found with a single pattern. So let's use that expression:

1. Identify pattern-host accounts in the `mysql.user` table like this:

```
mysql> SELECT User, Host FROM mysql.user WHERE Host REGEXP '[%_]';
+-------+---------------+
| User  | Host          |
+-------+---------------+
```


www.it-ebooks.info

```
| user1 | %            |
| user2 | %.example.com |
| user3 | _.example.com |
+-------+--------------+
```

2. To remove an identified account, use `DROP USER`:

```
mysql> DROP USER 'user1'@'%';
mysql> DROP USER 'user3'@'_.example.com';
```

Alternatively, rename an account to make the host part more specific:

```
mysql> RENAME USER 'user2'@'%.example.com' TO 'user2'@'host17.example.com';
```


www.it-ebooks.info