MYRO
My Record Oriented privilege system

Giorgio Calderone
INAF - OA Trieste, Italy

Luciano Nicastro
INAF - OAS Bologna, Italy

September 4, 2019
Ver. 0.3.2-alpha5
https://github.com/lnicastro/MyRO

This page intentionally left blank.

Contents

1 Introduction

2 MyRO: My Record Oriented privilege system

MyRO is the name we use to refer to a technique used to implement a database privilege system on a per-record basis on the MySQL database. Actually all database servers implement a privilege system based on tables or columns, that is if a user has grants to access a certain table (or a table's column) he can access all records of that table (or that table's column). **MyRO** lets you specify grants on a record level so a user can access only those records it is allowed to "select/update/delete". A consequence of this is that different users reading the same table will see different records. The grant mechanism provided by **MyRO** is similar to that of a Unix file system, that is each record belongs to a "owner" and a "group" and has three sets of permissions associated (for the owner, the users belonging to the group and all other users) that specify if that record can be read and/or written.

The software components of **MyRO** are a Perl script used to perform administrative tasks, a C library and a set of MySQL functions. The process of protecting tables is completely transparent to the final user, that is once **MyRO** has been installed and configured by the database administrator, users can access the database without even know that **MyRO** is working.

3 MyRO installation

The **MyRO** software library is distributed in a `tar.gz` package and via GitHub. You can find the latest version at https://github.com/lnicastro/MyRO . To unpack the package simply issue the command:

```
tar xvzf myro-x.y.z.tar.gz
```

where `x`, `y`, `z` are the version number (namely the first number is the major revision, the second number is the version, and third number is the subversion). A directory named `myro-x.y.z` will be created containing all sources code as well as the documentation and the scripts needed to install **MyRO**. Before installing **MyRO** you should check that all mandatory dependencies are satisfied (see Sect. 3.1), then you must follow a three step procedure: Configure, Compile and Installing **MyRO** .

3.1 Dependencies

The only mandatory packages required by **MyRO** are:

- MySQL sources (`http://www.mysql.org`, version 5.1.20 or later);
- Perl (`http://www.perl.com/`);
- The `DBD::mysql` perl module;

The `DBD::mysql` perl module can be easily installed through the `cpan` utility issuing the following command:

```
install DBD::mysql
```

See the `cpan` documentation for further informations. If these package are not already installed in the system you should install them before continuing.

3.2 Installing MyRO

3.2.1 Configure

Configuring **MyRO** means checking your system for compatibilities, search for include files and libraries, and finally produce all necessary Makefiles needed to compile **MyRO**. This is done automatically by the distributed `configure` script. Typically you can use this script without any option, as follows:

```
./configure
```

Anyway `configure` has a lot of options and switches (type `configure --help` for a list) to customize the compilation step. For further documentation see the `INSTALL` file.

3.2.2 Compile

To compile **MyRO**, once the `configure` script has been correctly executed, simply issue the command:

```
make
```

If you got errors while compiling check Sect. 3.2.1 and the `INSTALL` file.

3.2.3 Install

If **MyRO** has been correctly compiled you can install with the command:

```
make install
```

If your account doesn't have the permission to write in the path where it should be installed then you'll get an error. In this case you should login as "root" and retry.

4 How it works

Suppose the table to be protected is called `mytable_data`. **MyRO** adds three fields to the table to store information about the owner and group to which the records belong, and for the permissions. Then it installs three triggers on the table associated with the INSERT, UPDATE and DELETE events, and finally creates a view[1] with a custom name (for example `mytable`) with a simple `SELECT *` statement and a `WHERE` clause that filter the records readable by the user executing the query. This way the next time users will access `mytable` they will access the view, not the underlying table, and they will see only the records they're allowed to read or write. When a user tries to write on a table the corresponding trigger will be activated for each record being modified, to check if the user has grants to modify the record.

4.1 Users, groups and permissions

Users and groups used in **MyRO** are completely analogous to those of a Unix file system. Each MySQL account has a unique user id (`uid`) and can be a member of any number of groups, each associated with a unique group id (`gid`). Furthermore to each MySQL users is associated a flag that tells if the user is a "super-user", in this case all permission checking will be disabled. **MyRO** automatically handles its internal tables where all `uid`, group definition and group membership are stored. Also each record in a table protected by **MyRO** has three fields which contain the `uid` of the owner, the `gid` of the group owning that record, and a permission specification, that is a numerical code that specify if that record can be read and/or written by three categories of users: owner of the record, member of the group to which the record belongs to and all other users. **MyRO** uses all these information to determine if a record can be read and/or written. The permission specification is a sequence of 6 bits whose meaning are as follows (from the least to the most significant bit):

- write permission for owner;
- read permission for owner;
- write permission for members of the group;
- read permission for members of the group;
- write permission for all other users;
- read permission for all other users;

MyRO has a function to properly convert a permission specification to a more readable string representation of the grants:

[1] Views are database objects just like tables, except that they don't require disk space because their data is read from the actual tables.

```
mysql> select myro.fmtPerm(43);
+------------------+
| myro.fmtPerm(43) |
+------------------+
| rwr-r-           |
+------------------+
1 row in set (0.01 sec)
```

in which the two first characters refer to the owner, the third and fourth to the members of the group and the last two characters to all other users. Also **MyRO** automatically creates a group named `anygroup`. Members of this group are automatically members of any other group. This feature is necessary when you are dealing with many groups creation and destroy, and you know that a user must be member of all of this groups. Assigning the user to this group means it is member of any group, even if they are created after the user has been assigned to `anygroup`.

5 MyRO usage

All administrative tasks related to **MyRO**, like protecting tables or manipulating groups, can be performed using the the `myro` script. All available options can be displayed using the command:

```
myro --help
```

Before using **MyRO** it is necessary to install the functions in the MySQL database server (version >= 5.1.20) with the command:

```
myro --install
```

The password of the MySQL root account will be asked. This command also creates a database named `myro` which contains all **MyRO** internal tables.

5.1 Implementing the privilege system

Once **MyRO** has been installed in the database server all MySQL users will already have a `uid` and a default group to which they belong whose name is the same as the account username. Furthermore the group `anygroup` will automatically be created. The steps needed to implement the privilege system are as follows:

- creates all needed groups;
- modify users membership to groups;
- protect tables;
- check that all users' grants are consistent.

These operations are discussed in the next sections.

5.1.1 Handle groups

A new group can be created using the following command:

```
myro --addgroup <GroupName> [<Description>]
```

To delete a group use the command:

```
myro --delgroup <GroupName>
```

To see a list of defined groups issue the following SQL statement from a MySQL terminal:

```
call myro.groups();
```

5.1.2 Modify user's properties

With the following command it is possible to modify the default group to which a user belongs, specify if it is a "super-user", provide a description and an e-mail address:

```
myro --moduser <UserName> [<Group> [<Su> [<Description> [<Email>]]]]
```

To assign a user to a group use the command:

```
myro --assign <UserName> <Group>
```

To delete a user account you should drop the entire MySQL user account. To see the list of users along with their properties issue the following SQL statement from a MySQL terminal:

```
call myro.users();
```

5.1.3 Protect a table

To protect a table with **MyRO** issue the command:

```
myro --protect <DBName> <Table> <View>
```

where the first two arguments are the database which contains the table, and the table name to protect. The third argument is the name of the view that will be used to access the table. This command will add three fields in the table:

- `my_uid`: to store the user ID of record owner;
- `my_gid`: to store the group ID to which the record belongs;
- `my_perm`: to store the record permissions.

These fields are all of type `TINYINT UNSIGNED` so each record will require 3 more bytes on disk. Actually this field type is fixed, thus only 256 users and groups can be defined, but this feature may change in future releases. If the table being protected by myro already contains records the corresponding `my_uid`, `my_gid` and `my_perm` fields will contain NULL values, this means that these records will be accessible only from "super-users". To change the owner of these records open a MySQL terminal as user "root" and issue the following statement:

```
UPDATE <table> SET my_uid  = myro.uid(),
                   my_gid  = myro.defgid(),
                   my_perm = myro.defperm();
```

where `<table>` is the name of the table being protected. For those records that will be inserted after the table has been protected, this fields will be automatically populated with the correct values. The command `myro --protect` will also create a view with the same structure as the underlying table whose purpose is to filter records that can be read (etc.) by a user. Users should now use this view to access the data instead of the real table.

5.1.4 Check users' grants

All grants relative to the protected table should be removed for any users so that the only way to access the data is to use the view created by **MyRO**. To check that all user's grants are compatible with this requirement you can simply execute the `myro` script without any argument, so that if some user can directly access one of the protected tables a warning will be given. On the other hand users should have grants to access **MyRO**'s views. To give users the correct grants to access a **MyRO** view issue the command:

```
myro --grant <UserName> <Host> <DBName> <View>
```

5.2 Extended views

The view created by **MyRO** has the same structure as the underlying table without the `my_uid`, `my_gid` and `my_perm` fields, thus it is impossible to read the information stored in these fields. Typically these information are not needed by the users, however there may be some cases in which it is necessary to show these information. For this purpose **MyRO** can create three additional views which shows these information in various formats. To create this views issue the command:

```
myro --extended <DBName> <Table>
```

6 Reference for functions and procedures

Below there is a list of all functions and procedure created with **MyRO**, with their usage and parameters. Here we'll use some common abbreviations which resemble the ones used in Unix filesystems:

- `uid`: an integer which represent the user ID;
- `gid`: an integer which represent the group ID;
- `defgid`: an integer which represent the default group ID of any newly inserted record for a given user;
- `defperm`: an integer which represent the default permission specification of any newly inserted record for a given user.

6.1 myro.chkPerm

Check if a user can access a record for a read or write operation. The first parameter is the ID of the user who wants to access the record, the next three parameters are the value of the `my_uid`, `my_gid` and `my_perm` fields read from the record and the final parameter specifies which kind of access the user wants to perform.

Syntax:

```
myro.chkPerm(uid TINYINT UNSIGNED, my_uid TINYINT UNSIGNED, my_gid TINYINT UNSIGNED, my_perm TINYINT UNSIGNED, what CHAR(1))
```

Parameters:

1. *uid* `TINYINT UNSIGNED` : user ID;
2. *my_uid* `TINYINT UNSIGNED` : user ID of owner of the record;
3. *my_gid* `TINYINT UNSIGNED` : group ID of the group to which the record belongs to;
4. *my_perm* `TINYINT UNSIGNED` : permission specification;
5. *what* `CHAR(1)` : "r" for read access, "w" for write access.

Return value (`BOOL`):
: 1 if the user can access the record, 0 otherwise.

6.2 myro.defgid

Return the default group ID for the current user.

Syntax:
`myro.defgid()`

Return value (`TINYINT UNSIGNED`):
: default group ID.

6.3 myro.defgrp

Return the default group name for the current user.

Syntax:
`myro.defgrp()`

Return value (`CHAR(50)`):
: default group name.

6.4 myro.defperm

Return the default permission specification for the current user.

Syntax:
`myro.defperm()`

Return value (`TINYINT UNSIGNED`):
: default permission specification.

6.5 myro.fmtPerm

Return a string representation of a permission specification. A read permission is identified by character "r", a write permission by character "w". The string is made up of 6 characters, the first two refer to access for the owner of the record, the third and fourth to access for users belonging to the group, the last two characters for all other users.

Syntax:
`myro.fmtPerm(perm TINYINT UNSIGNED)`

Parameters:

1. *perm* `TINYINT UNSIGNED` : permission specification.

Return value (`CHAR(6)`):
: string representation of permission.

6.6 myro.gid2grp

Return the group name for a given group ID.

Syntax:
`myro.gid2grp(gid TINYINT UNSIGNED)`

Parameters:

1. *gid* `TINYINT UNSIGNED`

Return value (`CHAR(50)`):
: group name, or NULL if the `gid` does not exist.

6.7 myro.grp2gid

Return the group id for a given group name.

Syntax:
`myro.grp2gid(grp CHAR(50))`

Parameters:

1. *grp* `CHAR(50)` : group name.

Return value (`TINYINT UNSIGNED`):
: group ID, or NULL if the group does not exist.

6.8 myro.is_root

Return a flag telling if current user is "root".

Syntax:
`myro.is_root()`

Return value (`BOOL`):
: 1 if the current user is "root", 0 otherwise.

6.9 myro.is_su

Returns the "super user" flag of a given user ID.

Syntax:
`myro.is_su(uid TINYINT UNSIGNED)`

Parameters:

1. *uid* `TINYINT UNSIGNED` : user ID;

Return value (`BOOL`):
: "super user" flag.

6.10 myro.listGroups

Return a list of groups to which a user belongs to, given its user ID.

Syntax:
```
myro.listGroups(uid TINYINT UNSIGNED)
```

Parameters:

1. *uid* `TINYINT UNSIGNED` : user ID.

Return value (`VARCHAR(200)`):
: a string with a list of group names.

6.11 myro.myuser

Return the user name of the user who is calling the function.

Syntax:
```
myro.myuser()
```

Return value (`CHAR(50)`):
: user name of the current user.

6.12 myro.perm

Return a permission specification given its string representation. This function performis the inverse codification of the `myro.fmtPerm` function.

Syntax:
```
myro.perm(perm CHAR(6))
```

Parameters:

1. *perm* `CHAR(6)` : string representation of permission.

Return value (`TINYINT UNSIGNED`):
: permission specification.

6.13 myro.print_priv

Show the list of MySQL grants for a user.

Syntax:

```
CALL myro.print_priv(usr CHAR(50));
```

Parameters:

1. *usr* `CHAR(50)` : user name.

6.14 myro.su

Return the "super user" flag for the current user.

Syntax:

```
myro.su()
```

Return value (`BOOL`):
:"super user" flag.

6.15 myro.uid

Return the uid of the user who is calling the function.

Syntax:

```
myro.uid()
```

Return value (`TINYINT UNSIGNED`):
: user ID of the current user.

6.16 myro.uid2defgid

Return the default group ID of a given user ID.

Syntax:

```
myro.uid2defgid(uid TINYINT UNSIGNED)
```

Parameters:

1. *uid* `TINYINT UNSIGNED` : user ID.

Return value (`TINYINT UNSIGNED`):
: default group ID.

6.17 myro.uid2defperm

Return default permission specification of a given user ID.

Syntax:

```
myro.uid2defperm(uid TINYINT UNSIGNED)
```

Parameters:

1. *uid* `TINYINT UNSIGNED` : user ID.

Return value (`TINYINT UNSIGNED`):
: default permission specification.

6.18 myro.uid2usr

Return the user name of a given user ID.

Syntax:

```
myro.uid2usr(uid TINYINT UNSIGNED)
```

Parameters:

1. *uid* `TINYINT UNSIGNED` : user ID.

Return value (`CHAR(50)`):
: user name, or NULL if the `uid` doesn't exist.

6.19 myro.uid_member_of_anygroup

Check if a user is member of special group "anygroup".

Syntax:

```
myro.uid_member_of_anygroup(uid TINYINT UNSIGNED)
```

Parameters:

1. *uid* `TINYINT UNSIGNED` : user ID.

Return value (`BOOL`):
: 1 if the user is member of "anygroup", 0 otherwise.

6.20 myro.uid_member_of_gid

Check if a user is member of the group identified by `gid`.

Syntax:

```
myro.uid_member_of_gid(uid TINYINT UNSIGNED, gid TINYINT UNSIGNED)
```

Parameters:

1. *uid* `TINYINT UNSIGNED` : user ID;
2. *gid* `TINYINT UNSIGNED` : group ID.

Return value (`BOOL`):
: 1 if the user is member of the group identified by `gid`, 0 otherwise.

6.21 myro.uid_member_of_grp

Check if a user is member of a group.

Syntax:

```
myro.uid_member_of_grp(uid TINYINT UNSIGNED, grp CHAR(50))
```

Parameters:

1. *uid* `TINYINT UNSIGNED` : user ID.
2. *grp* `CHAR(50)` : group name.

Return value (`BOOL`):
: 1 if the user is member of the group, 0 otherwise.

6.22 myro.users

Show a list of all user accounts along with their properties and the groups they belong to.

Syntax:

```
CALL myro.users();
```

6.23 myro.groups

Shows a list of all defined groups with their descriptions.

Syntax:

```
CALL myro.groups();
```

6.24 myro.usr2defgid

Returns the user name of a given user name.

Syntax:
```
myro.usr2defgid(usr CHAR(50))
```

Parameters:

1. *usr* `CHAR(50)` : user name.

Return value (`TINYINT UNSIGNED`):
: default group ID.

6.25 myro.usr2uid

Return the user ID of a given user name.

Syntax:
```
myro.usr2uid(usr CHAR(50))
```

Parameters:

1. *usr* `CHAR(50)` : user name.

Return value (`TINYINT UNSIGNED`):
: user ID, or NULL if the user doesn't exist.

6.26 myro.usr_descr

Return the description of a given user name.

Syntax:
```
myro.usr_descr(usr CHAR(50))
```

Parameters:

1. *usr* `CHAR(50)` : user name.

Return value (`CHAR(50)`):
: description if any has been given, NULL otherwise.

6.27 myro.usr_email

Return the e-mail address of a given user name.

Syntax:

```
myro.usr_email(usr CHAR(50))
```

Parameters:

1. *usr* `CHAR(50)` : user name.

Return value (`CHAR(50)`):

: e-mail address if any has been given, NULL otherwise.