Goal

Support secure SQL standard based authorization for hive.

Current Status of Hive Authorization:

The default authorization in hive is not designed with the intent to protect against malicious users from accessing data they should not be accessing. It only helps in preventing users from accidentally doing operations they are not supposed to. It is also incomplete because it does not have authorization checks for many operations including the grant statement. The authorization checks happen on the hive client, but as the user is allowed to execute dfs commands, user defined functions and shell commands, it is possible to bypass the client security checks.

Hive also has support for storage based authorization, which is commonly used to add authorization to metastore server api calls. It can now (as of hive 0.12?) be used on the client side as well. While it can protect metastore against changes by malicious users, it does not support fine grained access control (column or row level).

The default authorization model in hive can be used to provide fine grained access control by creating views and granting access to views instead of the underlying table.

Proposal

SQL compliant option will be provide a third option for authorization in hive. This is recommended because it allows Hive to be fully SQL compliant in its authorization model without causing backward compatibility issues for current users. As users migrate to this more secure model, the current default authorization can be deprecated. This authorization mode can be used in conjunction with storage based authorization on the metastore server.

Like the current default authorization in hive, this will also be implemented on the client side. To provide security through this option, the client will have to be secured. This can be done by allowing users access only through hive server2, and by restricting the user code and non sql commands that can be run. The checks will happen against the user who submits the request, but the query will run as the hive server user. The directories and files for input data would have read access for this hive server user.

For users who don’t have the need to protect against malicious users, this can be supported through the hive command line as well.

References

For information on the SQL standard for security see:

- ISO 9075 Part 1 Framework - sections 4.2.6, 4.6.11
- ISO 9075 Part 2 Foundation - sections 4.35 and 12

Privileges

- `SELECT` - privilege gives read access to object
- `INSERT` - privilege gives ability to add data to object (table)
- `UPDATE` - privilege gives ability to run update queries on object (table)
- `DELETE` - privilege gives ability to delete data in object (table)
- `ALL PRIVILEGES` - gives all privileges

Support for INDEX and LOCK privileges will be introduced later.

Objects

- The privileges will apply to database, table and views. The support for table and views will be added first.

Access control statements

1. `CREATE ROLE` must be supported.
    a. Only the superuser role can create roles.
2. `DROP ROLE` must be supported.
    a. Only the superuser role can drop roles.
3. `SET ROLE` must be supported.
    a. There exists a concept of multiple roles that a user is set as belonging to. This can include a SUPERUSER role.
    b. A user can only run `SET ROLE` to switch to one of the roles they have been added into.
    c. In the default case, where a user has not yet run any `SET ROLE` commands, or if they've run `SET ROLE NONE`, they have privileges to do anything any of the roles they belong to allow, with the exception of the SUPERUSER role. SUPERUSER role privileges are never imported by default.
    d. If a user switches to a particular role using `SET ROLE`, they have only privileges accessible to that role. For eg., if user "bob" has access to roles "SALES" and "MARKETING", by default, they will have access to privileges granted by both roles. However, if they explicitly run `SET ROLE SALES` , then they will lose privileges granted by MARKETING or any of their other default privileges and retain only privileges granted by role SALES.
    e. A user can run `SET ROLE NONE` to reset their role.
    f. A user that switches to a SUPERUSER role has all privileges across the board.

4. `SHOW CURRENT ROLES` must be supported. (This is not a SQL standard.) This will list the current roles the user is acting as. If the user is not currently acting in a role, it will return `NONE`.
5. `SHOW ALL ROLES` must be supported. (This is not SQL standard.) This will list all currently existing roles. This will be available only to the superuser.[1]
6. `DESCRIBE ROLE` *`role`* must be supported. (This is not a SQL standard.) This will list all users and roles that participate in a role. This will be available to the superuser and any member of the role that has admin privileges on the role.
7. The role `PUBLIC`, which includes all users, must be supported.
8. A role `SUPERUSER` must be supported. (This is not a SQL standard.)
9. The grant privilege statement must be supported (`GRANT` *`action`* `ON [TABLE]` *`table`* `TO` *`grantee`*). Grantee can be a user or a role.
    a. Including `WITH GRANT OPTION.` As per the SQL specification, this allows users the grantee of this privilege to in turn become a grantor of that privilege on that object to other users or roles.
    b. Including `GRANTED BY.`
        i. Excluding `WITH HIERARCHY OPTION`.
10. The grant role statement must be supported (`GRANT` *`role`* `TO` *`grantee`*). Grantee can be a user or a role
    a. Including `WITH ADMIN OPTION`. As per the SQL specification, this allows users the grantee of this role to in turn become a grantor of that role to other users or roles.
    b. Including `GRANTED BY`.
11. The revoke privilege statement must be supported (`REVOKE` *`action`* `ON [TABLE]` *`table`* `FROM` *`grantee`*). Grantee can be a user or a role.
    a. Including `GRANT OPTION FOR.` As per the SQL specification, revoking this removes the ability of the revokee to grant this privilege to other users or roles, but does not remove the privilege itself from the revokee.
    b. Including `GRANTED BY.`
    c. Excluding `HIERARCHY OPTION FOR`.
    d. Excluding drop behavior (`CASCADE` and `RESTRICT`).
12. The revoke role statement must be supported (`REVOKE` *`role`* `FROM` *`grantee`*). Grantee can be a user or a role.
    a. Including `ADMIN OPTION FOR`. As per the SQL specification, this option removes the ability of the revokee to grant the role to other users or roles, but does not remove the user or role from participation in the role itself.
    b. Including `GRANTED BY.`
    c. Excluding drop behavior (`CASCADE` and `RESTRICT`).
13. `GRANTED BY` records the user or role that will be recorded as granting (or revoking) the privilege to the grantee (or revokee). The grantor may choose to specify his current username (if he has the privilege as that user to grant (or revoke) the privilege being

[1]

granted (or revoked)) or a role in which he participates (if that role has the privilege to grant (or revoke) the the privilege being granted (or revoked)). If the grantor (or revoker) does not include a `GRANTED BY` clause in the grant (or revoke) then the grantor will be recorded following the rules of ISO 9075-2 subsection 12.8, which basically specify that you use the username unless it is unavailable, and then you use the role name. Since username will always be available in our case this will mean it becomes the username.

14. A `SHOW GRANTS` statement must be supported. (This is not SQL standard.) It will give a list of objects, privileges, and who granted the grantee rights on that object
    a. When used by a user who does not participate in the SUPERUSER role, it can be used in three ways
        i. `SHOW GRANTS;` This will show all grants effective for the current user or roles, whichever is currently determining the user's privileges.
        ii. `SHOW GRANTS FOR` *`role`*`;` This will show grants effective for a given role. The user must participate in that role, otherwise permission will be denied.
        iii. `SHOW GRANTS FOR` *`user`*`;` where *user* is the username of the current user. This will show grants effective for the user. Attempts to list grants effective for other users will be denied.
    b. When used by a user who does participate in the SUPERUSER role, it can be used in three ways:
        i. `SHOW GRANTS;` As in the previous section.
        ii. `SHOW GRANTS FOR` *`role`*`;` This will show grants effective for a given role. Any role, regardless of whether the current user participates in it, can be shown.
        iii. `SHOW GRANTS FOR` *`user`*`;` where *user* is a valid username. This will show grants effective for that user. Any user, regardless of whether it is the current user, can be shown.
15. Permission to create a database must be given to any user, with the following caveats.[2]
    a. The user must have HDFS rights to create a directory in the Hive warehouse directory, if no database location is specified;
    b. or the user must own the directory provided in the database location specification (in this case it is recommended but not required that the group of the directory be hive, so that the hive user has access to the data in the database);
    c. and if the database is owned by a role (rather than a user) the directory must be owned by the hive user.
16. A database will be owned by the user or role that the user is acting as when the database is created. It will not be possible to create the database and assign it to another user. (This is not in alignment with the SQL standard.)

---

[2] In SQL, schemas (equivalent to Hive databases) are created with an owner. That owner need not match the creator of the schema, as the owner can be set via the `AUTHORIZATION` clause. Hive will differ from this in that it will not support assigning a database to another user. The creator will always be the owner Instead any user will be allowed to create a database as long as the underlying storage will support the create.

17. All permissions to CREATE, ALTER, and DROP tables are given to the owner of the database the table is in. It is not possible to grant these permissions to other users or roles.[3]
18. Bootstrapping of the superuser role will be done. This can be done through config options specified for metastore server, and it will populate the entries at startup.
19. Hive must be able to store the privileges information in its metastore. This will include:
    a. Roles that have been created.
    b. Users who participate in a given role.
    c. Privileges for each user for each object.
    d. Whether a particular user has grant option for a particular privilege.
    e. Who granted that privilege to that user.

Users

The user will be determined in the same manner as HDFS. In non-secure clusters the user will be determined by the user of the client process (e.g. `whoami` on Unix systems), or in the case of JDBC, WebHCat, or other remote connections as the user configured in the connection. In secure clusters the user will be determined by the kerberos credentials.

Roles

It will be useful to be able to use external systems as additional sources for user to group mapping. An example of external source for this information is the hadoop user to group mapping. This would mean that there needs to be support for multiple name spaces.

So a role in metastore could have a name like “role1@metastore”. A role from hadoop would have a name like “group1@hadoopgroups” . There will be a ‘role authority’ interface that can be implemented to add additional sources of role mapping information. Namespaces won’t need to be specified for the roles from the authority that is configured as default role authority.

Other changes needed for securing this model

1. HIVE-4887 - hive should have an option to disable non sql commands that impose security risk
2. A way for adding trusted user defined functions

[3] This is in line with the SQL specification. It is also the safest in terms of avoiding conflicts in file and directory ownership in HDFS. It also fits well with the Hadoop concepts of resource quotas managed by directories.