Storage Based Authorization

Table of contents

1 Default Authorization Model of Hive

The default authorization model of Hive supports a traditional RDBMS style of authorization based on users, groups and roles and granting them permissions to do operations on database or table. It is described in more detail in Hive Authorization.

This RDBMS style of authorization is not very suitable for the typical use cases in Hadoop because of the following differences in implementation:

1. Unlike a traditional RDBMS, Hive is not in complete control of all data underneath it. The data is stored in a number of files, and the file system has an independent authorization system.
2. Also unlike a traditional RDBMS which doesn't allow other programs to access the data directly, people tend to use other applications that read or write directly into files or directories that get used with Hive.

This creates problem scenarios like:

1. You grant permissions to a user, but the user can't access the database or file system because they don't have file system permissions.
2. You remove permissions for a user, but the user can still access the data directly through the file system, because they have file system permissions.

2 Storage-System Based Authorization Model

The Hive community realizes that there might not be a one-size-fits-all authorization model, so it has support for alternative authorization models to be plugged in.

In the HCatalog package, we have introduced implementation of an authorization interface that uses the permissions of the underlying file system (or in general, the storage backend) as the basis of permissions on each database, table or partition.

In Hive, when a file system is used for storage, there is a directory corresponding to a database or a table. With this authorization model, the read/write permissions a user or group has for this directory determine the permissions a user has on the database or table. In the case of other storage systems such as HBase, the authorization of equivalent entities in the system will be done using the system's authorization mechanism to determine the permissions in Hive.

For example, an alter table operation would check if the user has permissions on the table directory before allowing the operation, even if it might not change anything on the file system.

A user would need write access to the corresponding entity on the storage system to do any type of action that can modify the state of the database or table. The user needs read access to be able to do any non-modifying action on the database or table.

When the database or table is backed by a file system that has a Unix/POSIX-style permissions model (like HDFS), there are read(r) and write(w) permissions you can set for the owner user, group and 'other'. The file system's logic for determining if a user has permission on the directory or file will be used by Hive.

Details of HDFS permissions are given here: HDFS Permissions Guide.

2.1 Minimum Permissions

The following table shows the **minimum** permissions required for Hive operations under this authorization model:

Operation	Database Read Access	Database Write Access	Table Read Access	Table Write Access
LOAD				X
EXPORT			X	
IMPORT				X
CREATE TABLE		X		
CREATE TABLE AS SELECT		X	X source table	
DROP TABLE		X		
SELECT			X	
ALTER TABLE				X
SHOW TABLES	X			

Caution: Hive's current implementation of this authorization model does not prevent malicious users from doing bad things. See the Known Issues section below.

2.2 Unused DDL for Permissions

DDL statements that manage permissions for Hive's default authorization model do not have any effect on permissions in the storage-based model.

All GRANT and REVOKE statements for users, groups, and roles are ignored. See the Known Issues section below.

3 Configuring Storage-System Based Authorization

The implementation of the file-system based authorization model is available in the HCatalog package. (Support for this is likely to be added to the Hive package in the future.) So using this implementation requires installing the HCatalog package along with Hive.

The HCatalog jar needs to be added to the Hive classpath. You can add the following to hive-env.sh to ensure that it gets added:

```
export HIVE_AUX_JARS_PATH=<path to hcatalog jar>
```

The following entries need to be added to hive-site.xml to enable authorization:

```
  <property>
    <name>hive.security.authorization.enabled</name>
    <value>true</value>
    <description>enable or disable the hive client authorization</description>
  </property>

  <property>
    <name>hive.security.authorization.manager</name>
    <value>org.apache.hcatalog.security.HdfsAuthorizationProvider</value>
    <description>the hive client authorization manager class name.
    The user defined authorization class should implement interface
    org.apache.hadoop.hive.ql.security.authorization.HiveAuthorizationProvider.
    </description>
  </property>
```

To disable authorization, set `hive.security.authorization.enabled` to false. To use the default authorization model of Hive, don't set the `hive.security.authorization.manager` property.

4 Creating New Tables or Databases

To create new tables or databases with appropriate permissions, you can either use the Hive command line to create the table/database and then modify the permissions using a file system operation, or use the HCatalog command line (`hcat`) to create the database/table.

The HCatalog command line tool uses the same syntax as Hive, and will create the table or database with a corresponding directory being owned by the user creating it, and a group corresponding to the "-g" argument and permissions specified in the "-p" argument.

5 Known Issues

1. Some metadata operations (mostly read operations) do not check for authorization. See https://issues.apache.org/jira/browse/HIVE-3009.
2. The current implementation of Hive performs the authorization checks in the client. This means that malicious users can circumvent these checks.

3. A different authorization provider (StorageDelegationAuthorizationProvider) needs to be used for working with HBase tables as well. But that is not well tested.
4. Partition files and directories added by a Hive query don't inherit permissions from the table. This means that even if you grant permissions for a group to access a table, new partitions will have read permissions only for the owner, if the default umask for the cluster is configured as such. See https://issues.apache.org/jira/browse/HIVE-3094. A separate "`hdfs chmod`" command will be necessary to modify the permissions.
5. Although DDL statements for managing permissions have no effect in storage-based authorization, currently they do not return error messages. See https://issues.apache.org/jira/browse/HIVE-3010.