An Investigation into the Hive Security Model

Goal:

As per the Hive Authorization specification, the current Hive Security model is designed to prevent well-intentioned users from doing something bad, but it's not designed to keep malicious users/clients at bay. As an exercise, we shall attempt to design a malicious client/user, to see what problems exist with the model so as to see what needs to be prevented to more fully flesh out its security model in a large deployment which sees multiple hive users.

Another important thing to note is that Hive can be run in two modes, a standalone client-side mode, and a mode with a persistent metastore. For the scope of this experiment, we shall delve deeper into the mode where we use a metastore server while breifly introducing what we understand to be the working behaviour of each of these modes:

a) Standalone client-side model : With the standalone client-side model, hive runs entirely within the scope of the client. It is possible to sandbox hive in such a manner that only the user running hive has permissions to do anything with hive (using HDFS permissions), and thus, very secure and private. For simple users, this is easy to set up, and since it stores all state in HDFS, which can be user-protected, and in a database, which can be local to that user, it can be run in a very secure mode.

However, the hive metadata needs to be stored on a database, and as long as you're using hive in an embedded mode, with something like derby being your backing db, security is easy. When we decide to use mysql as a backing db, the db connection credentials are present in the hive-site.xml config that goes with hive accessing it, which is still possible to limit to the user accessing hive, and can use traditional unix security permissions to protect it. However, if you're using this mode, and want to scale out hive accessibility to multiple machines and multiple users, the mysql logon credentials are visible to all that can access hive, and thus, it is possible for a malicious user to talk directly to the mysql db, and cause havoc with the metadata present, even if HDFS permissions prevent the malicious user from directly affecting the data.

It is for this reason, that for the purpose of the scope of this experiment, we limit ourselves to discussing the case where we use a metastore server, where the mysql logon credentials are deployed only on the machine running as the metastore server, and this hive-site.xml is expected to be deployed as being private and restricted on that box.

b) Metastore server model : Hive can also work in a mode where metadata is managed by a separate metastore server, and the hive client communicates with the metastore server using

thrift, and the metastore server talks to a backing db (for eg., implemented in mysql) for metadata storage and persistance.

This model has the potential to scale out quite well, and be managed in a secure fashion. However, even this model, with the current implementation of authorization and authentication in hive, has potential issues.

A primary point of where the issues exist is that authentication and authorization for hive are done on the client-side, with the metastore trusting communication it receives over thrift as being valid to perform. So, fundamentally, as long as we can talk in thrift to the metastore server, we can perform pretty much any metadata operation and have access, which is no better than with the previous case where a person was able to gain access directly to the db. This, however, has the potential to be "fixed" by having authorization run in the metastore as well, so we shall explore this in a bit more detail.

Methodology:

A thing to note is that one need not even go through the trouble of writing another client that talks to the metastore using thrift - one could simply define and implement a custom HiveAuthorizationProvider and a HiveAuthenticationProvider that are wildly permissive and we can instruct Hive to use those for authorization and authentication. That is the approach that we have taken with this experiment - we've implemented a HiveAuthorizationProvider called PermissiveAuthorizationProvider which simply returns successfully for any authorization check asked for by hive, and a HiveAuthenticationProvider called PermissiveAuthenticationProvider, which, when asked for the username of the person performing the task, always returns "vic", our supposed Victim, and when asked for groups, returns "users", "hadoop" and "victims". We'll run most of our tests as another user, "mal", and see how often Mal can compromise security restrictions Vic has placed.

For each test, we shall first observe natural behaviour, with authorization/authentication turned off, then observe it with authorization/authentication turned on, using Hive Defaults, and then observe again with the permissive implementations plugged in.

Configuration:

hive-site.xml has the following configuration entries that we'll be fiddling around with:

i) Auth subsystem toggle:
hive.security.authorization.enabled : Set to true to turn on hive authorization

The default for hive is off, and we'll be testing with it on as well as off.

ii) Implementations of auth subsystems to plug in:
hive.security.authorization.manager : Class that implements HiveAuthorizationManager. Default = org.apache.hadoop.hive.ql.security.authorization.DefaultHiveAuthorizationProvider
hive.security.authenticator.manager : Class that implements HiveAuthenticationManager. Default = org.apache.hadoop.hive.ql.security.HadoopDefaultAuthenticator

We'll be testing this with it off(and thus unused), then testing with default implementations, and then we shall provide
org.apache.hadoop.hive.ql.security.authorization.PermissiveAuthorizationProvider and org.apache.hadoop.hive.ql.security.PermissiveAuthenticationProvider as authorization provider and authentication provider implementations respectively.

iii) Automatic grant privileges:

hive.security.authorization.createtable.user.grants
hive.security.authorization.createtable.group.grants
hive.security.authorization.createtable.role.grants

hive.security.authorization.createtable.owner.grants

These settings specify the default set of privileges automatically granted to users whenever a table gets created. The first three settings specify a series of users, groups and roles that are automatically granted certain privileges as soon as a user creates a table, without explicit use of grant statements. These might be used for reasons such as trying to give an admin role or an audit group access to all tables created, or to a user designated as operating a cleanup tool, for example. For the purpose of this experiment, it was not considered that these three were important to specify.

The fourth setting, however, is important. It specifies what permissions are automatically granted to the owner of the table upon having issued a create table command. By default, this is set to null (or empty), which would result in the owner of the table not having any privileges on the table, so it is important to set this to some reasonable value if we're using Hive Security. We will run our tests with this set to "ALL", which grants all privileges(ALTER,UPDATE,CREATE,DROP,INDEX,LOCK,SELECT,SHOW_DATABASE) to the owner of the table.

In addition to the hive-site.xml configurations, there is usually another aspect to hive configuration - the default warehouse directory, let's say "/user/hive/warehouse" on hdfs. The warehouse directory works as the directory where hive databases are created by default if another hdfs location is not specified during creation. The result of this requirement is that any

and all users that are allowed to create databases, and hence, directories inside the hive warehouse directories must have write permissions for the hive warehouse directory. This is true for the "default" directory inside that which houses all the tables created in the default db that do not specify an external location. Thus, it is strongly recommended that the hive warehouse directory, and the default db directory inside it have a sticky bit set on it in hdfs. Also, since the sticky bit is a new-ish development in hdfs, for security recommendations, it is a good idea to use external locations outside of the central warehouse directory for each user creating a database or table for their own use.

In this experiment, we'll test with both cases where relevant.

Tests : Basic case, no authorization/authentication

vic> show databases;

result : as expected, a list of existing databases

mal> show databases;

result : as expected, a list of existing databases

vic> show tables;

result : as expected, a list of existing tables in the default db

mal> show tables;

result : as expected, a list of existing tables in the default db

vic> create database vsimple

result : database created

vic> create database vsimple2 comment 'I am commenting'

result : database created

mal> show databases

result : able to see vsimple, vsimple2

mal> describe database vsimple2

result : able to see hdfs location and the comment created.

vic> create database v3 comment 'I am commenting' with dbproperties ('a'='1', 'b'='2')

result : database created

mal> describe database v3

result : able to see hdfs location and the comment created. Does not display the dbproperties.

vic> create database v4 comment 'I am commenting' location '/tmp/vic/v4' with dbproperties ('a'='1', 'b'='2')

result : database created

mal> describe database v4

result : able to see external hdfs location and the comment created. Does not display the dbproperties. (But that holds even for running as vic)

vic> create table v3.vtable(a string) stored as textfile

result : table created

mal> use v3;
mal> show tables;
mal> describe extended vtable;

result : mal is able to see vtable and all the metadata associated with it

vic> create table v4.vtable(a string) stored as textfile
vic> create table v4.vtabl2(a string) stored as textfile location '/tmp/vic/vt2'

result : both tables are created ok

mal> use v4;
mal> show tables;
mal> describe extended vtable;
mal> describe extended vtabl2;

result : As with v3's tables, we are able to see all the metadata.

mal> create v3.mtable(a string) stored as textfile

result : FAILED : ParseException line 1:7 Failed to recognize predicate 'v3'. Failed rule: 'kwRole' in create role

mal> use v3;
mal> create mtable(a string) stored as textfile

result : FAILED : MetaException(message: Got exception:
org.apache.hadoop.security.AccessControlException
org.apache.hadoop.security.AccessControlException: Permission denied: user=mal, access=write, inode="v3.db":vic:supergroup:rwxr-xr-x)

mal> create v3.mtable(a string) stored as textfile location '/tmp/mal/mt1'

result : FAILED : ParseException line 1:7 Failed to recognize predicate 'v3'. Failed rule: 'kwRole' in create role

mal> use v3;
mal> create mtable(a string) stored as textfile location '/tmp/mal/mt1'

result : OK - Item DB.A.1 : We were able to create an external table inside another user's database

vic> drop table v3.mtable;

result : OK - Item DB.A.2 : We were able to drop mal's table inside vic's db as vic. This by itself is not a surprise because vic owns the parent db, but still a point to note because of the unusual circumstances around which mal was able to create a table inside vic's db.

mal> use v3;
mal> create mtable(a string) stored as textfile location '/tmp/mal/mt1'

result : OK - repeating DB.A.1 to see if vic can remove it from with a "use" as well.

vic> use v3;
vic> drop table mtable;

result : OK - [a continuation of DB.A.2, we see that this is possible to do as well.]

mal> drop table v3.vtable;

result : OK - Item DB.A.3 : More dangerous than DB.A.2, as mal, we're able to drop an internal table owned by vic inside the DB owned by vic. On observing the hdfs location, we see that the table directory itself was not deleted, but the metadata associated with the table has been dropped. The dir would have also been deleted if we'd deleted it as vic, but deleting it as mal leaves that behind. Not that that's terribly important in the context of an entire table being deleted by a different user, but it's important to note that this happens.

vic> create table v3.vtable(a string) stored as textfile

result : OK , recreating it to test drop one more time

mal> use v3;
mal> drop table vtable;

result : OK - Item DB.A.3 continued, manages to drop again.

mal> drop database v3;

result : FAILURE : InvalidOperationException(mesage:Database v3 is not empty)

mal> drop database v3 cascade;

result : OK - Item DB.A.4 - We're able to outright drop the database, cascading to drop any tables inside. We note that it doesn't drop the db directory itself, which it would have if we'd deleted it as 'vic'

mal> drop database v4 cascade;

result : OK - [continuation of DB.A.4 - We're able to drop an external db as well, cascading to drop any tables inside. No hdfs directories are deleted in the process.]

vic> create database v1
vic> use v1
vic> create table t1(a string) partitioned by (b string);

result : OK - partitioned table created.

mal> describe extended t1;

result : OK - we can see all details in it

mal> alter table t1 set tblproperties ('z'='9')

mal> describe extended t1;

result : OK - parameters show that z=9 is now set. Item T.A.1

mal> alter table t1 set location 'hdfs://localhost:54310/tmp/mal/mtable'

result : OK, as expected, continuation of Item T.A.1, we are even able to modify the location, and it's now possible for Mal to feed erroneous data into Vic's processes that use his table, without directly visible changes.

mal> alter table t1 add partition(b='1') location '/tmp/mal/mtable/b1.part';

result : OK, as expected again, we are able to create a partition inside Vic's table. Item T.A.2

mal> alter table t1 set location 'hdfs://localhost:54310/user/hive/warehouse/v1.db/t1'
vic> alter table t1 add partition(b='2')

result : OK, partition added properly

mal> alter table t1 drop partition(b='2')

result : "Dropping the partition b=2" followed by FAILURE asking to check logs.

mal> show partitions t1;

result : Partition for b=2 does not exist. The above failure, in all likelihood, was caused because a hdfs rmr could not be done on the partition directory that was owned by Vic, but the metadata for the partition was still deleted. Item T.A.3

vic> alter table t1 add partition(b='3')
mal> alter table t1 partition(b='3') set location 'hdfs://localhost:54310/tmp/mal/mtable/b3.part'

result : ok : Mal is able to change location of Vic's partition inside Vic's table. Item P.A.1

mal> alter table t1 drop partition(b='3')

result : ok, with no error. This confirms that the error observed in T.A.3 was probably due to it attempting to delete the associated hdfs directory. With having changed the associated hdfs directory for the partition that Vic created, Mal is able to drop the partition entirely with no error.

Issues: Basic case, no authorization/authentication

DB.A.1 : Mal was able to create a table in Vic's db
DB.A.2 : Vic was able to delete Mal's table inside Vic's db.
DB.A.3 : Mal was able to drop tables created inside Vic's db
DB.A.4 : Mal is able to outright cascade-drop Vic's db.
T.A.1 : Mal is able to alter metadata of Vic's table.
T.A.2 : Mal is able to create a partition inside Vic's table.
T.A.3 : Mal is able to drop partitions that Vic created inside Vic's table.
P.A.1 : Mal is able to alter metadata of Vic's partition inside Vic's table.

Tests : With authorization, without malicious authorization/authentication backing

Retesting aforementioned issues:

vic> create database v1

result : ok

mal> create table v1.mtable (a string) stored as textfile

result : Authorization failed: No privilege 'Create' found for outputs { database:default}. Use show grant to get more details. This looks like the auth system saw the dbname.tablename syntax as being a tablename inside db default. This is itself a bug that should be fixed, but not a security issue in and of itself. [Item DB.B.1]

mal> use v1;
mal> create table mtable(a string) stored as textfile

result : Authorization failed: No privilege 'Create' found for outputs { database:v1}. : Good, this looks like fixes the issue observed by DB.A.1

mal> drop database v1

result : OK : Item DB.B.2 - DB.A.4 was not plugged, we're still able to drop a database created by vic.

vic> use v1;
vic> create table t1(a string) stored as textfile;

result : Authorization failed : No privilege 'Create' found for outputs { database:v1}. Hm., looks like we need to grant permissions to ourselves, and our config setting of "ALL" did not have the expected full impact - that setting was probably only for inside a table, as opposed to inside a db.

vic> grant all on database v1 to user vic with grant option;

result : OK

vic> create table t1(a string) stored as textfile;

result : OK

mal> create table m1(a string) stored as textfile;

result : Authorization failed: No privilege 'Create' found for outputs { database:v1}. : Good, this looks like this does indeed fix the issue observed by DB.A.1.

mal> grant all on database v1 to user mal with grant option;

result : OK : Item DB.B.3 - This looks bad. It looks like user 'mal' can simply go ahead and grant himself access, and then follow that up with doing whatever they want.

mal> use v1;
mal> create table m1(a string) stored as textfile;

result : FAILURE : Got an AccessControlException from hdfs in trying to create a dir inside v1.db.

mal> use v1;
mal> create table m1(a string) stored as textfile location '/tmp/mal/m1'

result : OK : We're able to create an external table using this route, which seems to be DB.A.1 again. But the underlying problem is DB.B.3 - without that, this isn't possible

vic> use v1;
vic> drop table m1;

result : OK : This is similar to DB.A.2, but we do not consider a security issue now, since we have explicitly granted permission to vic to all entries in the database v1, which includes m1.

mal> use v1;

mal> drop table t1;

result : OK : Again, this is bad, but the underlying problem is with DB.B.3, because mal was able to give himself permission to all entries inside v1.

mal> drop database v1;

result : OK : Again, we granted ourselves privileges here, but we also have DB.B.2 as an underlying problem.

vic> create database v1;

result : OK

vic> use v1;
vic> create table t1(a string) stored as textfile;

result : Authorization failed: No privilege 'Create' found for outputs { database:v1}. Okay, so we need to reinitialize permissions each time. That is good, because otherwise, previously created objects that have since been deleted but have leftover rules could affect new objects.

vic> use v1;
vic> grant all on database v1 to user vic with grant option;
vic> create table t1(a string) stored as textfile;

result : OK

mal> use v1;
mal> create table m1(a string) stored as textfile;

result : Authorization failed : Double checking - valid response

mal> use v1;
mal> drop table t1;

result : Authorization failed: No privilege 'Drop' found for outputs { database:v1, table:t1}. Good, this is working as intended.

mal> drop database v1;

result : Failed : InvalidOperationException(message: Database v1 is not empty)

mal> drop database cascade;

result : OK . Item DB.B.4 : This is bad - we're able to do a cascade db drop when we don't have privileges, even when we weren't allowed to drop a table in it specifically.

vic> create database v1;
vic> use v1;
vic> grant all on database v1 to user vic with grant option;
vic> create table t1(a string) partitioned by (b string);

result : OK - partitioned table created.

mal> describe extended t1;

result : OK - we can see all details in it

mal> alter table t1 set tblproperties ('z'='9')
mal> describe extended t1;

result : Authorization failed: No privilege 'Alter' found for inputs { database:v1, table:t1}. Good, this shows us that the authorization is working correctly. However, this potentially suffers from the same problem where mal can simply grant themselves privileges.

mal> alter table t1 set location 'hdfs://localhost:54310/tmp/mal/mtable'

result : As expected, again, authorization failure, this time complaining that 'Update' privilege was not found for the table. Except for the ability for Mal to grant privileges to himself, this is working as intended.

mal> alter table t1 add partition(b='1') location 'hdfs://localhost:54310/tmp/mal/mtable/b1.part';

result: Again, as expected, authorization failure, this time complaining that 'Create' privilege was not found inside the table.

vic> alter table t1 add partition(b='1');
vic> show partitions t1;

result : OK, Vic created a partition inside the table successfuly.

mal> alter table t1 drop partition(b='1')

result : Authorization failure, no 'Drop' privilege available for mal in the required table.

mal> alter table t1 partition(b='1') set location 'hdfs://localhost:54310/tmp/mal/mtable/b1.part'

result : Authorization failure, no 'Update' privilege found for table input.

mal> use v1;
mal> grant all on table t1 to user mal with grant option;

result : OK - same problem as DB.B.3

mal> alter table t1 set tblproperties ('z'='9')
mal> describe extended t1;

result : OK : as expected, DB.B.3 continues to be a problem

mal> alter table t1 set location 'hdfs://localhost:54310/tmp/mal/mtable'

result : OK : Succeeds again - DB.B.3

mal> alter table t1 add partition(b='2') location 'hdfs://localhost:54310/tmp/mal/mtable/b2.part';

result : OK : Success - DB.B.3

mal> alter table t1 drop partition(b='1')

result : Says its droppping partition b=1 and then reports an error, but a subsequent 'show partitions' shows that the partition was dropped. Again, DB.B.3

Issues: With authorization, without malicious authorization/authentication backing

DB.B.1 : Auth check for create table tested for entire 'dbname.tablename' as if it were a table name. This is a bug in the auth subsystem.
DB.B.2 : We're able to drop a db without having any associated permissions
DB.B.3 : A user is able to grant himself any permission he desires.
DB.B.4 : We're able to cascade drop a db, including dropping tables inside the db which we explicitly do not have the permission to drop.

Tests : With authorization/authentication, with malicious

authorization/authentication injected

Retesting aforementioned issues:

vic> create database v1;

result : ok

mal> create table v1.mtable (a string) stored as textfile

mal> use v1;
mal> create table mtable(a string) stored as textfile

result : AccessControlException obtained from trying to create a table in the db. This didn't fail from a failed auth check, it failed from trying to create a table in hdfs. This should be Item DB.C.1.

mal> drop database v1

result : OK : Item DB.C.2 - DB.B.2 & DB.A.4 was not plugged, we're still able to drop a database created by vic.

vic> create database v1;
vic> use v1;
vic> create table t1(a string) stored as textfile;

result : OK. This varies from previous auth case in that it didn't fail us for creating a table without permissions. This is bad if mal can do it too, as an external table.

mal> use v1;
mal> create table mtable(a string) stored as textfile location '/tmp/mal/mtable';

result : OK. Mal was able to create a table inside Vic's db. This actively successfully circumvents the issue prevented by authorization being enabled. This is a further aspect of DB.C.1 which was already noticed.

vic> use v1;
vic> drop table mtable;

result : OK : This is similar to DB.A.2, and is a security issue again, since this varies again with the DB.B cases and is similar to the DB.A case. Item DB.C.3

mal> use v1;
mal> drop table t1;

result : OK : Again, this is expected, given DB.C.3, but a subtly different issue, one that wasn't present in DB.B cases without mal granting himself privileges. Item DB.C.4

vic> create table t1(a string) stored as textfile;
mal> create table mtable(a string) stored as textfile location '/tmp/mal/mtable';
mal> drop database v1;

result : Failed : InvalidOperationException(message: Database v1 is not empty)

mal> drop database cascade;

result : OK . Item DB.C.5 : Expected given DB.B.4 and DB.C.*

vic> create database v1;
vic> use v1;
vic> create table t1(a string) partitioned by (b string);

result : OK - partitioned table created by Vic although Vic didn't explicitly have privileges to the db. Already observed as part of DB.C.1

mal> use v1;
mal> describe extended t1;

result : OK - we can see all details in it

mal> use v1;
mal> alter table t1 set tblproperties ('z'='9');
mal> describe extended t1;

result : OK : alter succeeds, 'Alter' privileges circumvented - Item T.C.1

mal> use v1;
mal> alter table t1 set location 'hdfs://localhost:54310/tmp/mal/mtable';

result : OK : alter succeeds, 'Update' privilege circumvented - Item T.C.2

mal> use v1;
mal> alter table t1 add partition(b='1') location 'hdfs://localhost:54310/tmp/mal/mtable/b1.part';

result : OK : alter succeeds, 'Create' privilege circumvente - Item T.C.3

vic> alter table t1 add partition(b='2');
vic> show partitions t1;

result : OK, Vic created a partition inside the table successfuly.

mal> alter table t1 partition(b='2') set location 'hdfs://localhost:54310/tmp/mal/mtable/b2.part'

result : update succeeds, 'Update' privilege circumvented - Item P.C.1

mal> alter table t1 drop partition(b='2')

result : drop succeeds, but hdfs rm fails, 'Drop' privilege circumvented - Item T.C.4

Issues: With authorization/authentication, with malicious authorization/authentication injected

DB.C.1 - Mal is able to create a table inside Vic's db without any permissions having been explicitly granted.
DB.C.2 - Mal is able to drop Vic's db.
DB.C.3 - Vic is able to drop Mal's table inside Vic's db - similar to DB.A.2 and a secondary effect.
DB.C.4 - Mal is able to drop Vic's table inside Vic's db without any permissions having been explicitly granted.
DB.C.5 - Mal is able to cascade drop Vic's db, deleting any tables inside it, whether or not he has permissions to them.
T.C.1 - Mal is able to alter Vic's table metadata, without any permissions having been explicitly granted.
T.C.2 - Mal is able perform updates to Vic's table, without any permissions having been explicitly granted.
T.C.3 - Mal is able to create partitions inside Vic's table, without any permissions having been explicitly granted.
T.C.4 - Mal is able to drop Vic's partitions inside Vic's table, without any permissions having been explicitly granted.
P.C.1 - Mal is able to update Vic's partition's metadata.

--

Conclusion:

a) If authorization and authentication is done on the client-side, then any client that can be compromised or replaced can lead to extensive metadata damage.
b) A flat grant-granting-grant is dangerous. Either the power to grant must be limited to admins, or we must use a more strict way of auto-determining grants as with models that mimic backing hdfs permissions in determining privileges.
c) A permission check needs to be added in for dropping databases - without a check on that, the effects are far reaching and dangerous.
d) For purposes of the auth subsystem, there seems to be a bug where it does not parse the ‘dbname.tablename’ format correctly, and treats it as a tablename alone.