Implementing Row and Column Level Security in Hive

Rahul Kumar Sharma, Dr. Vivek Kapoor

Abstract— Data security is a major concern in any data management system like a relational database management system (RDBMS) or Big data warehouse system like Hive. It is often required to grant access to a subset of data to users or denying access to some critical data.

The access can be restricted based on the records or rows that the user can access based on his identity and/or role or can be also be based on the columns that are accessible to the user.

The access of data can be related to viewing the data, updating/deleting the data and inserting the data.

Big data warehouse systems like Hive provide access control to allow/deny access to tables/view etc. to the user but there is a need to take the security to next level and enable security and individual row and column level

There is a need to introduce advance security features of row and column level security in Hive.

***Index Terms*— Big data security, Hive security, Row level security, column level security.**

I. INTRODUCTION

Traditionally Relational Database Management Systems (RDBMS) were used for storing and managing the data. RDBMS became popular in 1970s as they were able to connect the physical data to the logical data model easily.

Structured Query Language (SQL) was introduced in 1980 and became the de facto standard programming language for interacting with RDBMS. SQL provides ways to defining and managing the data model, to add/update/delete data, to retrieve data, transaction processing, analytic operations etc.

With huge data generation due to increase in internet traffic
and usage, generation of unstructured or semi structured data and the need to efficiently process and analyze this data, terms like Big Data became popular in 2000.

Big data security [1] has become a major issue and has its own set of challenges [2]. Big data security challenges are due to various factors [3] like Volume, Velocity, Variety, Variability etc.

Hive

Hive [4] is an open-source data warehouse system which supports querying and analyzing huge datasets stored in HDFS. Hive has three main functions: data summarization, query and analysis.

Hive is a now becoming the de-facto standard for SQL queries over huge volumes of data in Hadoop [5]. Hive does provide SQL-like access for data lying in HDFS enabling Hadoop to be used like a data warehouse structure.

Hive supports queries expressed in a language called HiveQL, which automatically translates SQL-like queries into MapReduce jobs executed on Hadoop. HiveQL supports custom MapReduce scripts to be plugged into queries. HQL has semantics and functions very similar to standard SQL in RDBMS so that experienced DBAs can easily get their hands on with HQL as well [6]. Hive is best suited for data warehouse applications, where real-time responsiveness to queries and record-level inserts, updates, and deletes are not required. Hive is also very nice for people who know SQL already. HQL can run on different computing frameworks like – Tex, MapReduce or Spark for better performance.

Hive provides a high-level, table-like structure on top of HDFS and support various data structures including tables, partitions, and buckets. Hive tables actually correspond to HDFS directories and can be partitioned and can be in turn divided into buckets [7].

The security challenges in Big Data warehouse systems like Hive can include Data Leakage [8] where the data is made available to user for which he is not authorized.

II. ROW AND COLUMN LEVEL SECURITY

Generally, when access is provided to users to access record definition using a query, they have access to all the records of data in the table built using the associated record definition. For row level access control, we need to restrict users from seeing some of those data rows.

Row level security is used for tables that hold sensitive data. For row-level security, users can actually have access to a table without having access to all rows on that table.

Row level security enables you to store data for many users in a single database and table, while at the same time restricting row-level access based on a user's identity, role, or execution context.

Column level security is to restrict the access to specific columns for users based on their role and identity. Access can

be granted to users based on user/group so that they can have restricted access to columns in a table. A user can be restricted access, to view a column data

III. Current Security Configurations in Hive

Currently Hive supports integration with Hadoop security. Authentication/Authorization are also supported in Hive. It is currently possible to define users, groups and roles. Privileges can be granted or revoked to a user or a group. As of now the following privileges are supported –

- **ALL -** All the privileges applied at once.
- **ALTER -** The ability to alter tables.
- **CREATE -** The ability to create tables.
- **DROP -** The ability to remove tables or partitions inside of tables.
- **LOCK -** The ability to lock and unlock tables when concurrency is enabled
- **SELECT -** The ability to query a table or partition.
- **SHOW_DATABASE** - The ability to view the available databases.
- **UPDATE -** The ability to load or insert table into table or partition.

Hive Authorization used to verify if a particular user has permission granted to do certain actions, like - creating, reading/writing data or metadata.

There is a need to introduce advance security features of row and column security in Hive. A mechanism is required for configuring row and column level security in Hive where the administrator can configure the security rules at user/group level. The administrator should be able to configure the rules for row level security to allow users to have access to only certain rows in a Hive table, this can be achieved having capability to define a set of conditions on basis of which user should have access to data. If those set of conditions are fulfilled for a row, user should be able to access that row.

The administrator should also be able to configure the column level security to not allow users to see the columns in data for which they don't have access to. If those columns are selected in Hive query, the value of data for those columns should be masked in query results.

IV. Proposed solution for Row and Column Level Security

A security plugin or layer is needed for Hive which should be able to intercept the Hive queries and should be able to add the filters for row level security before the queries are submitted to Hive runtime.

The security plugin should also intercept the query results and should apply the column based rules on the data set returned from query and should mask the values of columns for which user does not have access.

The security plugin should be able to apply the rules on basis of user and group. An administrator GUI is also proposed which should have capability to define the rules on basis of user and group
The following access will be supported at user and group level

Row Level Security

Access to specific rows of table to a user or group based on certain criteria which can be defined by a set of rules based on values of columns in row

Column Level Security

Access to specific columns of table to a user or a group

Following will be the mechanism to define the row/column level access –

Row level access

The row level access can be defined for a user or group. Row level access can be defined by specifying some rules that are based on values of certain columns.

It should be possible to specify combination of one or more rules to grant/deny access to a user or a group. The rules can be based on exact values of column (i.e. using equals or not equals for varchar/string fields) and should also support arithmetic operators like equals ("="), not equal to ("! ="), greater than (">"), less than ("<"), ranges etc. for numeric and date columns. Rules can be combined by logical "AND" and "OR" operators

Column level access

The column level access can be defined for a user or group. It should be possible to specify the columns of a table for which a user or group should be granted or denied access.

Administrator Interface

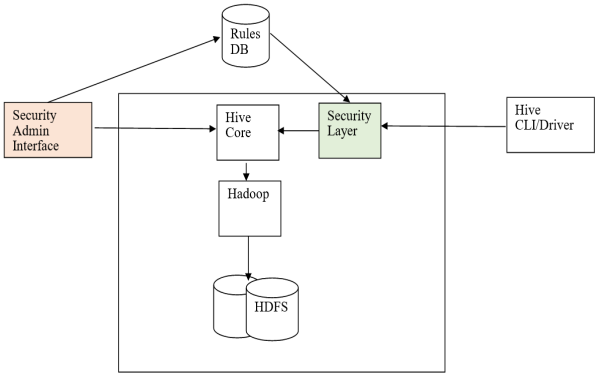
A Web based Graphical User Interface (GUI) should also be provided to facilitate the administrator to create the rules for row and column based access based on user and group level.

V. Proposed Implementation Approach for Row and Column Level Security

For implementing the rules and privileges for row and column level access for user, and group, the following mechanism is proposed to translate the Hive query to include

the rules for row and column level access (based on user or group)

Following diagram represents the high level view of the proposed solution –



The proposed approach is to use Hive pre-execution hooks to intercept the Hive queries and add filters for enabling row and level security and to apply post filters to capture the query results to apply column level security. The pre and post execution logic will be embedded in a new security layer which can be installed as library with Hive setup.

Following is the sequences of action which should be performed on execution of a Hive query –

1. User issues a Hive query to fetch some records from a set of Hive tables. The query includes column from these Hive tables and optionally a set of conditions in where clause. The query may join 2 or more tables

2. The security layer intercepts the query in a Hive pre-execution hook

3. Security layer looks for the row and column level security rules in rules database which are configured for the user and/or group. The administrator has already configured the row level security filters which are stored in rules database

4. Security layer adds additional conditions in the where clause to impose the rules for row level security for user/group to only allow data for which the user and group have access for various Hive tables

5. Security layer forwards the query to Hive to continue the query execution

6. Security layer checks if the query results contain any column for which user has not been granted access at user or group level. If user does not have access to column, security layer will mask the value of that column with null values. The administrator has already configured the column level security rules which are stored in rules database

VI. IMPLEMENTATION RESULTS

This system was implemented and it was verified that the following capabilities are provided –

- Row level security implemented for Hive - Capability to define row level security using a combination of rules defined at user and group level

- Column level security implemented for Hive - Capability to define column level security using rules defined at user and group level

- Administrator GUI having capability to define the rules for row and column level security using a GUI interface

Result Verification

For verification of row and column level security, we created the following data set Employee and queried it using HiveQL with user "hduser" without any row and column level security policies configured -

Beeline version 2.1.1 by Apache Hive

jdbc:hive2://localhost:10000/mydb> **select * from employee**;

```
+--------------+----------------+-------------------+----------------
+--+
| employee.id | employee.name | employee.surname |
employee.dept |
+--------------+----------------+-------------------+----------------
+--+
| 1          | rahul          | sharma           | it           |
| 2          | upendra        | jariya           | it           |
| 3          | manish         | sharma           | cs           |
+--------------+----------------+-------------------+----------------
+--+
```

3 rows selected (0.572 seconds)

As we can see that user hduser has access to all the 3 rows in Employee Hive table and he can see all columns as well.

Now we added a row level security policy to show records for department = 'it' only for user "hduser", the policy was added with following details -

- policy name – Department Row Filter
- hive table – Employee
- user – hduser
- **filter condition – (dept = 'it')**

The dataset Employee was now queried after adding the above policy –

Beeline version 2.1.1 by Apache Hive

jdbc:hive2://localhost:10000/mydb> **select * from employee**;

```
+--------------+----------------+-------------------+----------------
+--+
| employee.id  | employee.name  | employee.surname  |
employee.dept  |
+--------------+----------------+-------------------+----------------
+--+
| 1            | rahul          | sharma            | it             |
| 2            | upendra        | jariya            | it             |
+--------------+----------------+-------------------+----------------
+--+
```

2 rows selected (0.575 seconds)

As we can see that user hduser now has access to only 2 rows in Employee Hive table with dept = 'it' only

Now we added a column level security policy to mask values for id column, the policy was added with following details -

- policy name – Employee ID column Filter
- hive table – Employee
- user – hduser
- **masked column – id (nullify values)**

The dataset Employee was now queried after adding the above policy –

Beeline version 2.1.1 by Apache Hive

: jdbc:hive2://localhost:10000/mydb> **select * from employee;**

```
+--------------+----------------+-------------------+----------------
+--+
| employee.id  | employee.name  | employee.surname  |
employee.dept  |
+--------------+----------------+-------------------+----------------
+--+
| NULL         | rahul          | sharma            | it             |
| NULL         | upendra        | jariya            | it             |
+--------------+----------------+-------------------+----------------
+--+
```

2 rows selected (0.591 seconds)

As we can see that user hduser now has access to only 2 rows in Employee Hive table with dept = 'it' only and cannot see value for "id" column (the values for "id" column are nullified)

VII. FUTURE WORK

The security layer implemented can be integrated with Hive as a standard plugin or extension. This security layer and admin interface can be contributed to open source community

VIII. CONCLUSION

This proposal will enhance the security of Hive/Hadoop and add new dimensions of row and column level security. This will also provide an integrated approach for achieving row and column level security in Hive.

Currently for achieving row and column level security in Hive, views need to be created for specific columns and conditions. With this approach, creating views for achieving row and column level security would be no longer and row/column level security will become an integral part of Hive/Hadoop security

ACKNOWLEDGMENT

REFERENCES

[1] Big data - http://www.nessi-europe.com/Files/Private/NESSI_WhitePaper_BigData.pdf
[2] Raghav Toshniwal, Kanishka Ghosh Dastidar, Asoke Nath: Big Data Security Issues and Challenges. International Journal of Innovative Research in Advanced Engineering Issue 2, Volume 2 (2015)
[3] Kalyani Shirudkar, Dilip Motwani: Big Data Security. International Journal of Advance Research in Computer Science and Software Engineering Issue 3 Volume 5 (2015)
[4] Apache Hive - https://hive.apache.org/
[5] Apache Hadoop - http://hadoop.apache.org/
[6] Dean Wampler, Edward Capriolo, Jason Rutherglen: Programming Hive Second edition, O'Reilly Media (2017)
[7] Dayong Du: Apache Hive Essentials First edition, Packt Publishing (2015)
[8] Charles Schmitt: Security and Privacy in the Era of Big Data. A RENSI/National Consortium for Data Science white paper

Rahul Kumar Sharma - Software engineer by profession, holds Bachelors' degree in I.T. SGSITS, pursuing Masters' degree I.T. with specialization in Information Security from IET DAVV Indore.

Dr. Vivek Kapoor – Senior faculty at Department of Information Technology, DAVV Indore. Holds Bachelors', Masters' degree and PhD. in Computer Science