Multi-Layer Authorization Framework for a Representative Hadoop Ecosystem Deployment

Maanak Gupta, Farhan Patwa, James Benson and Ravi Sandhu
Institute for Cyber Security and Department of Computer Science
University of Texas at San Antonio, One UTSA Circle, San Antonio, TX, 78249, USA
gmaanakg@yahoo.com,{farhan.patwa,james.benson,ravi.sandhu}@utsa.edu

ABSTRACT

Apache Hadoop is a predominant software framework to store and process vast amount of data, produced in varied formats. Data stored in Hadoop multi-tenant data lake often includes sensitive data such as social security numbers, intelligence sources and medical particulars, which should only be accessed by legitimate users. Apache Ranger and Apache Sentry are important authorization systems providing fine-grained access control across several Hadoop ecosystem services. In this paper, we provide a comprehensive explanation for the authorization framework offered by Hadoop ecosystem, incorporating core Hadoop 2.x native access control features and capabilities offered by Apache Ranger, with prime focus on data services including Apache Hive and Hadoop 2.x core services. A multi-layer authorization system is discussed and demonstrated, reflecting access control for services, data, applications and infrastructure resources inside a representative Hadoop ecosystem instance. A concrete use case is discussed to underline the application of aforementioned access control points. We use Hortonworks Hadoop distribution HDP 2.5 to exhibit this multi-layer access control framework.

CCS CONCEPTS

•Security and privacy → Security requirements; Access control; Authorization;

KEYWORDS

Access Control; Hadoop Ecosystem; Big Data; Data Lake; Role Based; Attributes; Object Tags

ACM Reference format:
Maanak Gupta, Farhan Patwa, James Benson and Ravi Sandhu. 2017. Multi-Layer Authorization Framework for a
Representative Hadoop Ecosystem Deployment. In *Proceedings of SACMAT'17, June 21–23, 2017, Indianapolis, IN, USA, ,* 8 pages.
DOI: http://dx.doi.org/10.1145/3078861.3084173

1 INTRODUCTION

In past several years, enterprises have grown their reliance on Big Data for critical financial and strategic decisions. An estimate in IDC's Digital Universe study, predicts world's data size to reach 44 zettabytes by 2020 [10]. Multi-format Big Data is collected from diverse sources including sensors, tennis rackets, web browsing, social media, power meters etc., to improve organization's operational efficiency, revenue and to offer personalised customer experience. Leveraging full potential and gaining valuable insights of such massive data sets require enormous infrastructure for storage and computation in real time manner.

Apache Hadoop [3] offers a distributed, scalable and cost-efficient open-source framework for storing and analysing structured, unstructured and semi-structured data in variety of formats. Resilient storage and rich analytical capabilities provided by Hadoop and its ecosystem components (Apache HBase, Apache Hive etc.) makes it a prime choice as a Big Data processing system in government and industry. Such wide acceptability of Hadoop ecosystem comes with the responsibility to make it secure against cyber attacks.

The Multi-tenant Hadoop Data lake stores sensitive information including credit card numbers, medical records and social security numbers (SSNs), requiring the cluster to be protected against cyber threats. Unauthorized access to data assets can have serious impact on its confidentiality and integrity. An inside user can masquerade by running malicious code to impersonate Hadoop core services including HDFS NameNode, DataNode or YARN ResourceManager. A nefarious user can also modify, view or delete other users' applications. It is also possible to execute denial of resources attack, where a malicious user can submit lengthy jobs which consume all the cluster resources preventing other users from submitting new jobs. The challenges to mitigate these threats include distributed and partitioned file system and computing, scale of Hadoop cluster, multi-tenant environment and multi-level access of same data elements to different users. Correspondingly, Hadoop ecosystem has deployed several security measures including authentication, authorization, data encryption and network security.

Access Control [13][20] mechanisms are vital in restricting users and applications access to authorized resources. Apache Hadoop deploys a multi-layer authorization framework using Access Control Lists (ACLs) to authorize users to access data, infrastructure resources and services in Hadoop cluster. Apache Ranger [5] and Apache Sentry [6] are two widely deployed systems to enforce fine grainer authorization across several Hadoop ecosystem services. Both systems offer a centralized administration console to store and manage security policies for multiple ecosystem components. They provide plugins which are hooked to ecosystem services to decide and enforce access control, based on the policies pulled from central policy server. Sentry supports role-based authorization model, whereas Ranger assigns permissions to users and groups.

SACMAT'17, June 21–23, 2017, Indianapolis, IN, USA
 ACM ISBN 978-1-4503-4702-0/17/06…$15.00.
DOI: http://dx.doi.org/10.1145/3078861.3084173

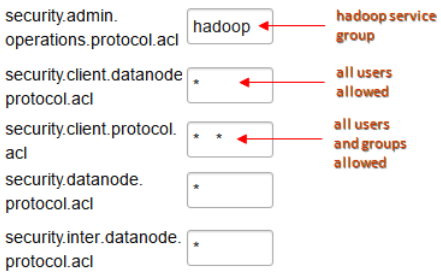

Figure 1: Hadoop Daemons Access Configuration

In this paper we present a unified explanation for authorization mechanisms in Hadoop ecosystem primarily focusing on data access via Apache Hive. We demonstrate the capabilities using multiple Apache projects as a part of Hortonworks Hadoop distribution HDP (Hortonworks Data Platform) 2.5 sandbox [14]. For our purpose we assume data is already ingested in the cluster.

The paper is organized as follows. Section 2 discusses and demonstrates individual access control points offered in Hadoop ecosystem. A complete multi-layer authorization architecture is presented in Section 3. We discuss a concrete use case in Section 4. Section 5 reviews related work, followed by conclusions in Section 6.

2 HADOOP ECOSYSTEM ACCESS CONTROL MECHANISMS

Hadoop ecosystem deploys Defense in Depth approach to enforce access control security mechanisms across the cluster. A user is allowed to access data services and Hadoop daemons in the cluster through service level authorization layer. Data stored in HDFS (Hadoop Distributed File System) is secured by POSIX-style file and directory permissions or extended ACLs. Cluster resources are segregated through YARN capacity (or fair) scheduler queues which restrict applications to use limited resources and also prevent application modifications from unauthorized users. Apache Ranger further offers data masking, filtering and dynamic contextual information to achieve finer-grained authorization. This section discusses and demonstrates these access control mechanisms using Hortonworks HDP 2.5.

2.1 Hadoop and Data Services Access

Once a user is authenticated, the first layer of access control mechanisms is provided by service layer authorization. This layer controls access to ecosystem services (Apache Hive, HDFS, Apache HBase) inside Hadoop cluster, much before data underlying the services is accessed. It also checks if a user is allowed to access Hadoop daemons such as HDFS NameNode, YARN ResourceManager, and ApplicationMaster, to submit applications or to query status. Several Hadoop core services also need communication with each other for task updates or cluster resource status, which is also controlled by this layer. This cross-service authorization prevents rogue processes from impersonating as Hadoop daemons and gaining control to data and resources.

By default, service level authorization is disabled in Hadoop. It is enabled in `core-site.xml` configuration file by setting property `hadoop.security.authorization = true` in

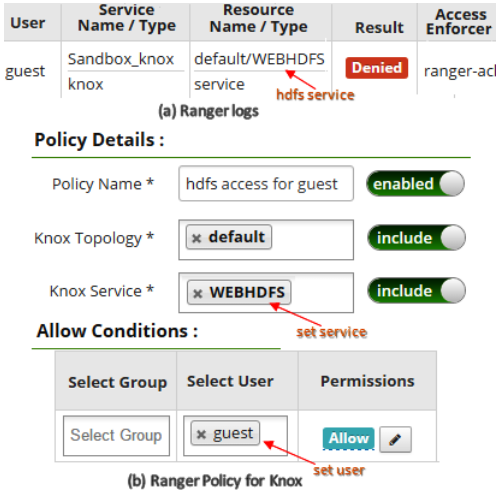

Figure 2: WebHDFS Access via Apache Knox

all the nodes of Hadoop cluster. The ACLs for various Hadoop services (daemons) are set in `hadoop-policy.xml` file. An example Hadoop service access control property is `security.client.datanode.protocol.acl`. This property lists the set of users which can access HDFS DataNode, required for data block recovery. An example of cross Hadoop services ACL property is `security.resourcetracker.protocol.acl`, which is used when YARN ResourceManager and NodeManager communicate with each other for resource monitoring. The recommended value for this ACL is 'yarn' service user. It should be noted that all Hadoop services run under a service user Unix account, and the ACLs for cross service should include permissions for the service users. Using Apache Ambari [1], these properties can be changed under HDFS configuration. Figure 1 shows sample Hadoop services ACL properties. These ACLs include allowed users and groups information, with a special value $*$ including all users or groups. Similar ACLs can also be specified for blocked services or users. Note that some ACLs, for example, `security.job.task.protocol.acl` used by Map and Reduce tasks to communicate with YARN NodeManager, may give permissions to all users ($*$) as the identity of applications running these tasks cannot be enumerated in advance.

Apache Knox [4] offers an API gateway to provide access to several Hadoop ecosystem services (WebHDFS, Hive etc.) to external users. It provides a single access point to Hadoop REST services by intercepting user access requests and enforcing policies to allow or deny users to access services. Internal ports to ecosystem services are not accessible to end users. Apache Knox validates permissions of users to access cluster services much before data or other resource access decisions are made, thereby preventing unauthorized access at early stage of user request lifecycle. As shown in Figure 2 (a) (Apache Ranger logs), a user guest trying to access HDFS NameNode service via Knox by issuing a list files curl command `curl -iku guest:guest-password -X GET 'https://10.x.x.255:8443/gateway/default/webhdfs/v1/?op=LISTSTATUS'` is denied access as the user is not allowed to access HDFS service inside the cluster protected by Apache Knox gateway. Once the policy is set in Apache Ranger (for Knox

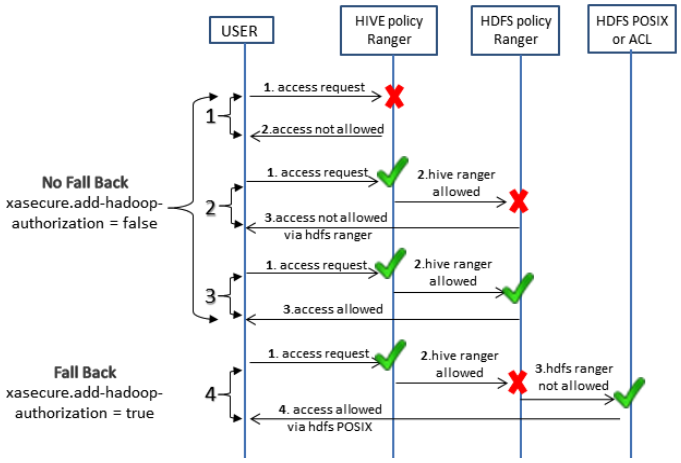

Figure 3: Hive and HDFS Access Configurations

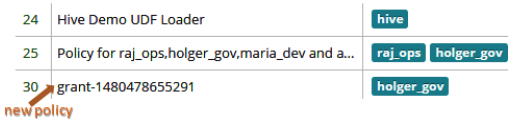

Figure 4: Policy Creation using Grant Command

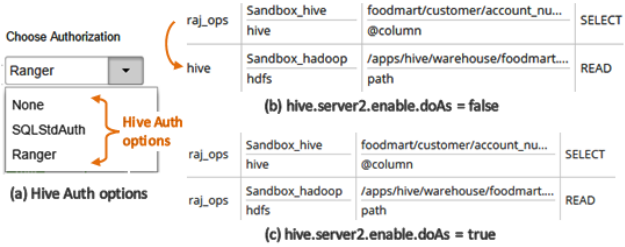

Figure 5: Authorization Options and Impersonation

service) to allow access to WebHDFS service to guest user (Figure 2 (b)), the user will be allowed access to HDFS NameNode. Similar policies can be set for other ecosystem services such as Apache Hive, HBase, YARN ResourceManager or Ambari, to allow external users to access cluster. Hence, the perimeter security offered by Apache Knox enables secure access to several ecosystem services from users outside the cluster. It should be noted that Apache Knox gateway can protect multiple clusters and offer single endpoint to all services across clusters.

2.2 Data Objects Access

The backbone of Hadoop ecosystem is fault-tolerant and distributed file system for storing data, called Hadoop Distributed File System (HDFS). With YARN architecture, same data can be read directly from HDFS or through several data engines (or services) such as Apache Hive or Apache Pig, to perform operations on supported data model. Data layer authorization (using Apache Ranger plugins) ensures access to HDFS directly or via different data engines, to authorized users and applications. HDFS supports both POSIX style permissions (read, write and execute) on files and directories, and extended Access Control Lists. ACLs authorization is disabled by default in HDFS which can be enabled by setting `dfs.namenode.acls.enabled = true` in configuration file `hdfs-site.xml` for HDFS NameNode. ACLs can be set on files and directories using `setfacl`. To check ACLs on a particular file `getfacl` command is used. For example, to add ACL permission read on file marketing for members in group execs and to get the list of permissions on file marketing, following commands are issued:

```
hdfs dfs -setfacl -m group:execs:r-- /marketing
hdfs dfs -getfacl /marketing
group:execs:r--
```

Apache Ranger plugin can be enabled for HDFS service via Ranger configuration in Apache Ambari, which will manage and enforce security policies on HDFS files and directories [15]. Security administrator has option to configure layers of authorization controls to check access to HDFS by setting property `xasecure.add-hadoop-authorization` in `ranger-hdfs-security.xml` under Ranger configuration in Ambari. When the property is set to true, authorization engine will check HDFS ACLs if Ranger policy is not defined or denies access, whereas property set to false will make decision based only on Ranger policies without checking HDFS ACLs.

When a user attempts to access data through data services such as Apache Hive, access policies for both data service and HDFS are checked. As shown in Figure 3, when a user tries to access table using Apache Hive, Hive Ranger policies are first checked followed by HDFS Ranger policies for the corresponding data files. When the value of `xasecure.add-hadoop-authorization` is set to false, only HDFS Ranger policies are checked. If Hive Ranger policy allows access and HDFS Ranger denies, the user is not allowed to operate on the data. Both Hive and HDFS permissions must allow access to data. If `xasecure.add-hadoop-authorization = true` and HDFS Ranger policy does not allow access, HDFS POSIX permissions are further checked to make decision. Apache Ranger audit logs will reflect the policies (HDFS or Ranger ACLs) used to make access decision. Therefore, a user may need to have access authorization at multiple data service levels to perform operations on data items. Security administrator can set policies in Apache Ranger using UI, REST API or SQL grant/revoke commands. Ranger UI offers drop-down menu for several ecosystem services where administrator can select resources, users and actions along with contextual or policy conditions. SQL grant command issued by administrator (using command line tools such as beeline) will be intercepted by Apache Ranger plugin and will create corresponding policies. As shown in Figure 4, admin user raj_ops issuing command `grant select, update on table foodmart.customer to user holger_gov;` will generate policy in Ranger allowing user holger_gov for select and update action on table foodmart.customer.

Most of the services in Hadoop ecosystem have in-built authorization mechanisms, besides the centralized access control framework provided by Hadoop core, Apache Ranger and Apache Sentry. For example, Apache Hive has storage-based authorization, SQL standard based authorization and default-mode authorization models. Storage-based authorization uses the file system permissions

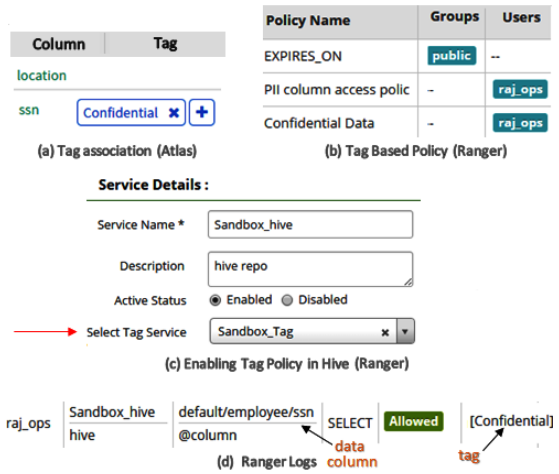

Figure 6: Tag Based Policy in Apache Ranger

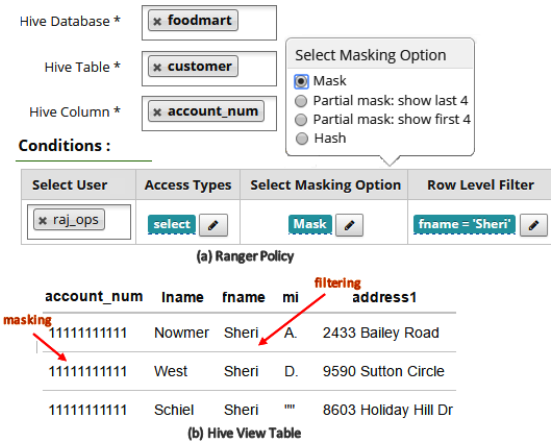

Figure 7: Data Masking and Row Filtering

to be applied for metastore and is primarily used when Hive acts as a table storage layer. This mode only provides authorization at table, database or partition level. SQL standards based authorization provides a fine-grained control to columns, rows or views by using grant/revoke commands, and uses HiveServer2 to enforce controls. Both these modes can be used simultaneously depending on the requirements and use case. In our demonstration, we use Ranger as the authorization provider for all the services in Hortonworks HDP 2.5. As shown in Figure 5(a), Apache Ambari provides drop-down in Hive configuration to select the authorization model which will automatically change the required configuration properties in `hive-site.xml` and `hiveserver2-site.xml`.

As Hadoop data storage layer is managed by HDFS, it is possible that user might be required to access data only via certain data engines such as Apache Hive or Pig, and not directly in HDFS. For example, business analyst users may get access to data using HiveQL via Apache Hive query but may not be allowed to access corresponding data files at HDFS level through MapReduce jobs (as HiveQL query changes to MapReduce job). In such requirements, the end user issuing HiveQL command is changed to 'hive' service user (user running Apache Hive service) for underlying HDFS level data access. Another use-case may need users to have data access at both Apache Hive and HDFS level. Such user impersonation is managed by changing `hive.server2.enable.doAs` property in Ambari Hive configuration, which changes the user running jobs at HDFS level. As shown in Apache Ranger audit logs (Figures 5(b) and (c)), with `hive.server2.enable.doAs` set to false, raj_ops is end user running HiveQL command which changes to 'hive' service user when accessing HDFS data through corresponding MapReduce job. For `hive.server2.enable.doAs = true`, end user raj_ops is accessing data at both Hive level and HDFS [16].

Besides resource-based policies (set on tables, files or queues) offered in Ranger, resource tags (attribute values) can be used to create tag-based policies. Apache Atlas [2] is used to associate attribute values with resources in ecosystem based on content, expiration time or sensitivity of resources. This metadata information (tag) can be used to create tag policy using tag service in Ranger. This tag service is then attached to several data services to use tag-based policies. Once attached, all the resources associated with a particular tag will be secured by corresponding tag-based policy. As shown in Figure 6, confidential tag is created and assigned to column ssn in Apache Atlas, and a tag-based policy on confidential tag is created under tag service in Ranger. Once the policy is set, Apache Hive (or other service) is configured to support tag-based policy by selecting tag service in Hive configuration under Ranger. With this change, Apache Hive will support both resource and tag-based policy. When user raj_ops tries to access object ssn, Ranger audit logs will reflect the tag policy used along with the tag name. In this way, instead of creating separate policies across each data service for same resource, single tag policy can be used for all services. Tags can also have attributes such as expiration time which will reject access to associated object after certain time or date.

Apache Ranger can also set policies to perform column masking and row filtering on data items in Hive for certain users. Column masking helps to conceal data columns having sensitive information from users and applications by replacing data with random characters. Several masking options including complete mask, partial mask, hash mask, and date mask, are available. With row filtering, some rows are hidden from users based on the where clause and conditions set in policies. In Figure 7, column masking and row filtering are demonstrated together. Here, for raj_ops user, masking has been done on account_num column in table customer. Also row filtering has been applied where only rows with fname=Sheri are to be displayed. With these policies, the resultant access when raj_ops issues `select * from customer` command is shown in Figure 7(b). It can be seen that account_num is masked and only rows with fname=Sheri are displayed as a result for user raj_ops.

2.3 Context Enricher and Policy Conditions

Some authorization use-cases may require finer grained access to resources not only based on the subjects and objects, but also on certain environmental or contextual information. For example, it may be required to give access to a user when the user is at a specific location, between a particular time frame or from a designated IP address. Apache Ranger provides context enricher and condition

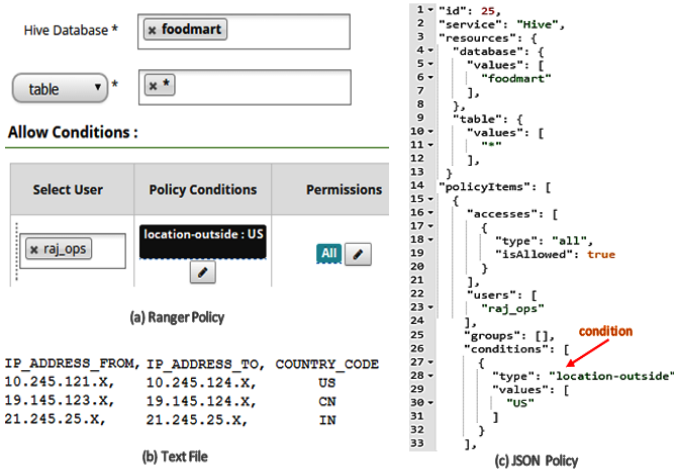

Figure 8: Geo Location Based Ranger Policies

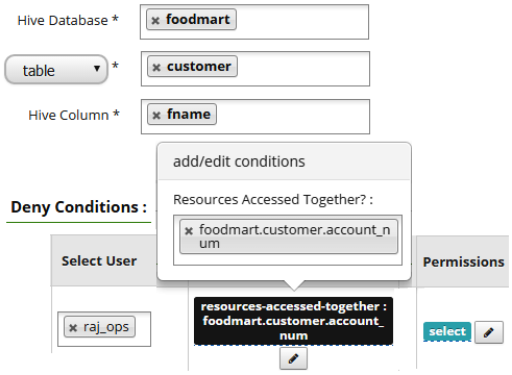

Figure 9: Prohibiting Data Combination

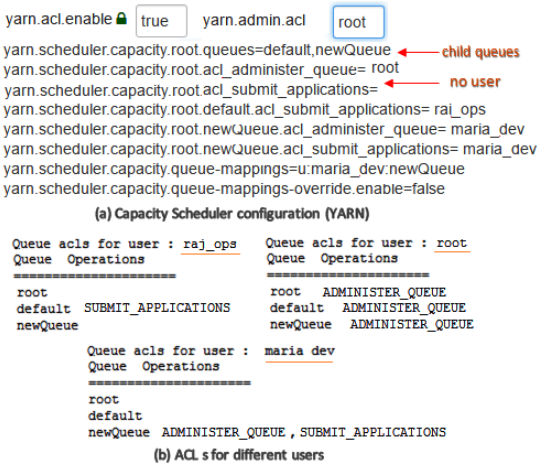

Figure 10: YARN Queues Sample Configuration

evaluators to realize dynamic policies, which change the access results based on contextual information. Context enricher is a java class which is used to enhance user access request by adding extra information such as location or user's IP address, based on mapping stored in a separate text file. Condition evaluators are used to add the access under certain conditions in Ranger policies [9]. To enable context enricher and condition evaluation, service definitions of each Hadoop ecosystem service should be enhanced with enricher and evaluation information. As shown in Figure 8(a), a Ranger policy is created to allow all actions to user raj_ops on all tables in database foodmart when its requesting location is outside US. When raj_ops tries to access table in foodmart database, Ranger Hive plugin will intercept this request and based on raj_ops ip address and the text file (which maps IP address to location and shown in Figure 8(b)), context enricher will add location information of raj_ops to the access request [18]. When Ranger plugin evaluates this request based on the policy (shown in Figure 8(c)) cached from central server, it will check the subjects, objects and operations involved and also if policy condition is satisfied. If the policy condition is not satisfied it will deny the access request even if other parameters are satisfied. Deny conditions can also be specified in Ranger by setting `enableDenyAndExceptionsInPolicies = true` in definitions of ecosystem services. Therefore, a policy can be formulated where a user can be denied access to resources based on specific user locations. Similarly, fine-grained access policies involving attributes can be defined using context enricher and condition evaluator hooks in Ranger. Role-based access control can be implemented using these hooks where a text file can have the current user to roles mapping and the policy condition can require the user to have a certain role to perform actions on resources. In such case, when a user requests access, context enricher will add roles of user to the request and evaluator will check against roles in policy condition to allow or deny access.

Privacy policies may require certain data-sets combinations not revealed to specific users. For example, SSN should not be shown together with name and address to sales users. Such a requirement can be encapsulated using prohibition policies, which can be defined using policy conditions and helps prevent certain data sets combination to be accessed together. Apache Ranger includes `RangerHiveResourcesAccessedTogetherCondition` condition evaluator which can check if the particular data sets can be accessed together. As shown in Figure 9, a deny condition can be set in a policy which prevents user raj_ops from performing a select operation on fname and account_num column together in customer table in foodmart database.

Note that the data-level access controls discussed in subsection 2.2 and 2.3, take effect only after the user is allowed to access Hive service through service-level authorization.

2.4 Cluster Resource and Application Access

Multi-tenant Hadoop cluster requires optimized sharing of resources among several tenants. In Hadoop 2.x, YARN capacity (or fair) scheduler defines queues with resource limits so that users submitting application in one of the queues can access a fraction of total cluster resources. YARN queues further prevent rogue users from submitting application to Hadoop cluster and prevent users from killing or modifying other user applications. Capacity (or fair) scheduler queues have ACLs associated with them which determine

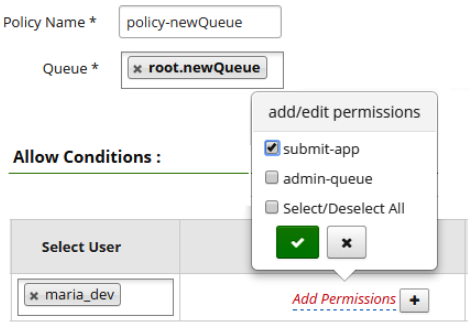

Figure 11: YARN Queues Authorization in Ranger

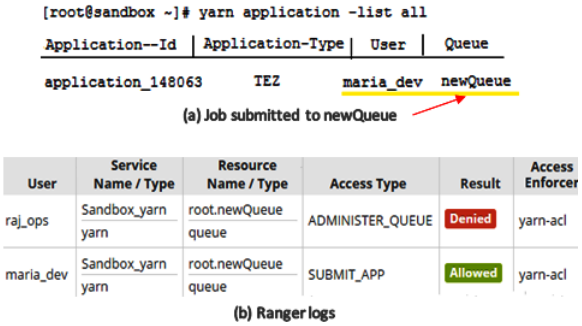

Figure 12: Queue and Job Access Control

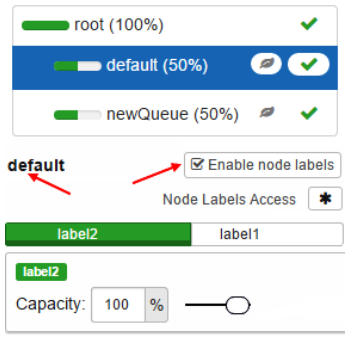

Figure 13: Node Labels and YARN Queues

the set of users who can submit applications or who can administer submitted application. By default, owner of application and queue administrator can modify or view applications submitted to a queue. These queues are hierarchial in nature where the resources allocated to the parent queue are distributed among child queues. ACLs defined at parent level are descended downwards to child queues, while applications are always submitted at leaf queue levels. YARN ACLs can be defined in YARN capacity scheduler configuration in Apache Ambari. Apache Ranger also provides plugin to enforce access control on YARN queues.

As shown in Figure 10(a), YARN ACLs are enabled by setting `yarn.acl.enable = true` under resource manager configuration in Apache Ambari. Also root is set as admin for YARN cluster via `yarn.admin.acl` property. As shown, YARN capacity scheduler configuration has two new child queues created (default and newQueue) under root parent queue. We have set root user as the administrator of root queue. The hierarchial nature of queues will descend the administrative privileges of root user to all the child queues, thereby making user root as administrator for both default and newQueue also. None of the users should be allowed to submit application to root queue (leave blank) as that will allow them to submit in all child queues also. It should be noted that two ACLs, one for administration and other for submit application operation, should be set for each YARN queue. User raj_ops is now set to submit application only to default queue, while maria_dev can administer and submit applications in newQueue child queue. With this capacity scheduler configuration, when each user issues `mapred queue -showacls` command to see the list of queues along with permissions, it can be seen in Figure 10(b) that root is admin for all three queues while raj_ops can only submit to default queue and maria_dev has administrator and submit application permissions on newQueue. Similar access control requirements can be achieved using Apache Ranger via YARN service plugin. As shown in Figure 11, a policy is set where user maria_dev is allowed to submit application in newQueue under root parent queue.

With aforementioned configurations, when user maria_dev tries to access data by issuing a HiveQL command `select min(customer.account_num) from foodmart.customer` using Hive service, its corresponding MapReduce (or Tez) job will be submitted to newQueue child queue (Figure 12(a)). If user raj_ops tries to kill or view the details of the job submitted by maria_dev, raj_ops is not allowed as the user is neither the admin nor the owner of job (shown by Ranger audit logs in Figure 12(b)). ACLs can also be included in the configuration files of application submitted, to restrict access to the application. For MapReduce job, ACLs are first enabled by setting `mapreduce.cluster.acls.enabled = true`. The list of users or groups which can modify or view job is set using `mapreduce.job.acl-modify-job` and `mapreduce.job.acl-view-job` properties respectively.

Cluster nodes can be assigned node labels for restricting node access to certain users and applications. By associating labels, sub-clusters can be created so that user applications can be executed on certain set of nodes, specific to application requirements. These node labels are associated with YARN capacity scheduler queues such that all the users submitting jobs to a particular queue will run their jobs only on nodes with labels that were assigned to the queue. These node labels can be enabled in YARN configuration using Ambari. Once enabled, administrator can issue command to create new labels. For example `sudo -u yarn yarn rmadmin -addToClusterNodeLabels "label1(exclusive=true),label2(exclusive=false)"` will add two labels label1 and label2, which can be verified using `yarn cluster --list-node-labels`. Here "exclusive=true" means only the applications running in the queue associated with the label can use the node. Therefore, if resources are free in that node, these cannot be used by applications in other queues. Once label is created it is associated with node-managers using `yarn rmadmin -replaceLabelsOnNode "<node-address>=label2"`. Node labels and associated nodes can be checked in YARN ResourceManager web interface (port 8088) under node labels tab.

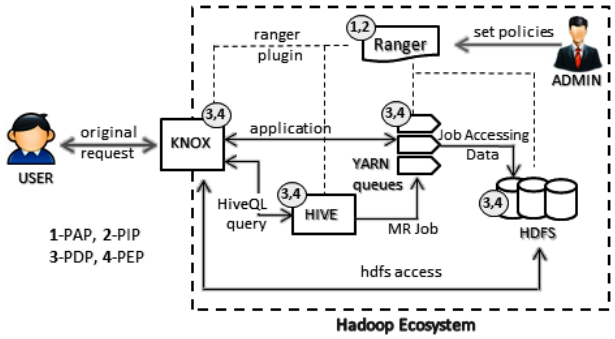

Figure 14: Hadoop Ecosystem Authorization Architecture

After labels are assigned to nodes, queues should be associated with node labels via YARN Queue Manager in Ambari (shown in Figure 13). Here with root as parent queue, and newQueue and default as children, we have assigned node-label label2 to default queue. The capacity specifies percentage of node resources (associated with label) that can be accessed by applications in default queue. As shown in Figure 13, applications in default queue can access all resources of the node which has label label2. YARN capacity scheduler configuration will automatically set `"yarn.scheduler.capacity.root.default.accessible-node-labels=label2"` property. With this configuration, it follows that raj_ops which was previously allowed to submit application in default queue, will run all its applications on node with label label2.

3 MULTI-LAYER AUTHORIZATION ARCHITECTURE

In this section we present a unified authorization architecture in a Hadoop ecosystem. This multi-layer architecture outlines how the individual access control points demonstrated in previous section fit into ecosystem to offer 'Defense in Depth' approach to protect cluster resources. Figure 14 shows several layers of access control decision and enforcement points to authorized access to resources in services such as Apache Hive and HDFS. Apache Ranger provides a centralized Policy Administration and Policy Information Point (1-PAP, 2-PIP) whereas its plugins appended with individual services, periodically cache policies from policy server and provide Policy Decision and Enforcement Points (3-PDP, 4-PEP). The architecture portrays how several Apache projects work in a coherent manner to realize multi-layer authorization in a Hadoop ecosystem.

An authenticated user is first examined by the perimeter security layer offered by Apache Knox, which checks if the user is allowed to access services inside the ecosystem. This external user interacts with all ecosystem services via single access point REST API provided by Apache Knox. When a user issues a request, it is intercepted by Apache Ranger plugin attached to Apache Knox, which checks its policies to decide and enforce access to services inside the ecosystem. If a user wants to submit an application or to check application status, the user should be allowed to access YARN ResourceManager through Apache Knox. If a user issued HiveQL using a client, the user should be allowed access to Apache Hive (HiveServer2) when trying to access HDFS files directly, access to WebHDFS service (REST access to HDFS) should be allowed. This is the first layer which a user must pass through, much before access to data is checked, which helps to restrict users at an early stage.

Once admitted through perimeter layer, if user requires data access at HDFS layer, permissions for files and directories are checked using HDFS Ranger plugin. If a user issues HiveQL command, two layers of data services are checked: one at Apache Hive level and other for the corresponding data files at HDFS level. Both can be done using Apache Ranger plugin attached with each of these services. If a user is submitting a YARN application, the user should be authorized to access YARN queues to submit the application. This is done after YARN ResourceManager access is allowed through Apache Knox. YARN queue policies are enforced either through YARN capacity scheduler configuration or through Ranger plugin. This will prevent unknown users from submitting jobs in the cluster. User's access to YARN queues is also required when HiveQL command is issued, since the command results in a MapReduce or Tez job, which is also submitted to YARN queues. Since these jobs will access data in HDFS, owners of the jobs should have permissions on HDFS files also. Further, data masking or policy conditions can be applied at individual services to get fine-grained access control.

Based on the architecture, it can be understood that several authorization check points come into play to protect unauthorized access to cluster resources. We have not discussed cross-service access between Hadoop core daemons since this does not involve the user directly, and is mainly enforced using core Hadoop service ACLs and not through Ranger plugin. Moreover these daemon processes always run in background and their communication is essential for the proper functioning of Hadoop ecosystem.

4 USE CASE

In this section we present real world use cases to demonstrate the application and configuration of multiple access control mechanisms offered in a Hadoop ecosystem. We assume users are authenticated by some external mechanism and data ingestion is already done.

Suppose an Internet of Things (IoT) provider gathers data from devices assembled in smart homes. The data generated from smart devices is continuously stored in Hadoop Data Lake, which is analysed by the provider to offer better customer experience. As there are multiple IoT providers using the same Hadoop lake for storing and processing their data, security and privacy requirements are extremely critical. Let us say that the provider has two different functional users Alice and Bob, Alice belonging to sales and Bob to data-analyst group. Both users access data in the same Hadoop cluster with different operational and data permissions. Alice can only access data using Apache Hive ecosystem service via beeline client, with no access directly to HDFS data whereas Bob, as a data-analyst, can run YARN applications inside the cluster and may also require access to HDFS data directly. This service level security requirement is achieved by creating Knox policy using Ranger, which will allow Alice to access Hive service from outside the cluster. Another policy will allow Bob to access WebHDFS service and YARN ResourceManager via Knox gateway as discussed in Subsection 2.1.

Based on Hadoop cluster resources and service level agreement, cluster administrator is required to assign set of resources to users from IoT enterprise. The administrator has to ensure that only

authorized users are allowed to submit YARN applications and access data. To accomplish this, cluster administrator creates a `queue-IoT` with a set of resources and allows both Alice and Bob to submit application in `queue-IoT` using Apache Ranger. Further, node labels (nLabel1, nLabel2, nLabel3) are assigned to three worker nodes and `queue-IoT` is associated with these three labels using YARN Queue manager in Ambari. Now, whenever Alice issues a HiveQL command or Bob runs a YARN application inside the cluster, it will only pass through `queue-IoT` and will run on three nodes with set labels. This configuration will also ensure that only Alice and Bob are allowed to submit jobs from IoT enterprise and they are both assigned a set of resources using capacity scheduler configuration and node labels. Further no user is allowed to kill or modify jobs submitted by Alice and Bob. Complete configuration to achieve this use case is discussed in Subsection 2.4.

Data generated from IoT devices is stored in HDFS files. User Bob is allowed to access all data files from IoT enterprise stored under `data-IoT` directory. Alice is only allowed select operation on `address` and `temperature` column on table `table-thermostat-texas` created from file `file-thermostat-texas` in directory `data-IoT`. Further, only `zipcode` part of the `address` should be visible to Alice. It is also required to give data access to Alice only when Alice belongs to Marketing project, since it is possible Alice might be shifted to other department in the organization. To allow Bob to have all permissions on `data-IoT` directory, either HDFS POSIX or ACLs can be used or Apache Ranger plugin can set policy to allow all operations on `data-IoT` directory. This will ensure that all applications run by Bob can access all files in `data-IoT` directory. For user Alice, Apache Hive policies are set to allow select operation on column `address` and `temperature` in table `table-thermostat-texas`. For displaying only `zip-code` field from `address`, data masking is done on the remaining part. Since the requirement denies direct access of user Alice at HDFS level, `hive.server2.enable.doAs` is set to false, which will only allow hive service user to access HDFS files and not Alice end user. Here two layers of data access are checked: for Hive service, access is checked for Alice and for HDFS, access is checked for Hive service user. Therefore policy for Hive service user should be also set in Apache Ranger. Alice's current project membership is required to allow access to table `table-thermostat-texas`. This is ensured by use of context enricher and condition evaluators. The enricher will use a text file with the current user and project mapping. Security administrator will create a policy including marketing as the policy condition. Whenever Alice will try to access table `table-thermostat-texas`, context enricher will add Alice's current project to the access request using the text file, which will be checked against policy condition by the evaluator to allow or deny access. Security administrator can also use tags to create policy. In this case, Apache Atlas will be used to create a tag `tag-IoT` and will be associated with columns `address` and `temperature`. A policy will be created on `tag-IoT` under Tag service in Ranger, which will enforce access to columns `address` and `temperature`. Subsection 2.2 and 2.3 discuss additional details about these configurations.

These use-case requirements clearly illustrate how a layered authorization framework (involving service, data and resource access) is applied and configured to restrict unauthorized resources access.

5 RELATED WORK

Several books, reports and papers have been published [8, 11, 12, 19, 21–23] to discuss security aspects of Hadoop ecosystem and Big Data. Hortonworks HDP (Hortonworks Data Platform) [14] uses Apache Ranger as authorization framework. Cloudera [7] CDH (Cloudera Distribution including Apache Hadoop) offers Apache Sentry [6] as central access control component. MapR Converged Data Platform [17] offers Hadoop with data placement control.

6 CONCLUSION

In this paper we discuss and demonstrate authorization capabilities provided by a representative Hadoop ecosystem deployment. The multi-layer authorization framework offered by Apache Hadoop and Apache Ranger covering services, data, cluster resources and application access is presented. This document can be read as a reference guide to understand access control capabilities and how they are achieved in Hadoop ecosystem using Apache Ranger. We have also discussed real world use-cases which exhibit the application of individual access control points in a coherent manner, including extensions enabled by context enrichers.

ACKNOWLEDGMENTS

This work is partially supported by NSF grants CNS-1111925, CNS-1423481, CNS-1538418, DoD ARL Grant W911NF-15-1-0518 and The Texas Sustainable Energy Research Institute at UTSA.

REFERENCES

[1] *Apache Ambari.* https://ambari.apache.org/.
[2] *Apache Atlas.* http://atlas.apache.org/.
[3] *Apache Hadoop.* http://hadoop.apache.org/.
[4] *Apache Knox.* https://knox.apache.org/.
[5] *Apache Ranger.* http://ranger.apache.org/.
[6] *Apache Sentry.* http://sentry.apache.org/.
[7] Cloudera. *Cloudera Distribution Hadoop.* https://www.cloudera.com/.
[8] Devaraj Das, Owen O'Malley, Sanjay Radia, and Kan Zhang. 2011. Adding Security to Apache Hadoop. *Hortonworks, IBM* (2011).
[9] Balaji Ganeshan and Alok Nath. 2015. Dynamic Policy Hooks in Ranger. https://cwiki.apache.org/confluence/display/RANGER/Dynamic+Policy+Hooks+in+Ranger+-+Configure+and+Use. (2015).
[10] John Gantz et al. 2012. Digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future* (2012).
[11] Maanak Gupta, Farhan Patwa, and Ravi Sandhu. 2017. Object-Tagged RBAC Model for the Hadoop Ecosystem. In *Proc. of IFIP DBSec (To appear).* Springer, 18 Pages.
[12] Maanak Gupta, Farhan Patwa, and Ravi Sandhu. 2017. POSTER: Access Control Model for the Hadoop Ecosystem. In *Proc. of ACM SACMAT (To appear).* ACM, 3 Pages.
[13] Maanak Gupta and Ravi Sandhu. 2016. The $GURA_G$ Administrative Model for User and Group Attribute Assignment. In *Proc. of NSS.* Springer, 318–332.
[14] Hortonworks. *Hortonworks Data Platform.* https://hortonworks.com/.
[15] Robert Hryniewicz. 2016. Best Practices in HDFS Autorization with Apache Ranger. https://hortonworks.com/blog/best-practices-in-hdfs-authorization-with-apache-ranger/. (2016).
[16] Robert Hryniewicz. 2016. Best Practices in Hive Autorization with Apache Ranger. https://hortonworks.com/blog/best-practices-for-hive-authorization-using-apache-ranger -in-hdp-2-2/. (2016).
[17] MapR. *Converged Data Platform.* https://mapr.com/.
[18] Madhan Neethiraj. 2016. Geo-location based policies. https://cwiki.apache.org/confluence/display/RANGER/Geo-location+based+policies. (2016).
[19] Owen O'Malley, Kan Zhang, Sanjay Radia, Ram Marti, and Christopher Harrell. 2009. Hadoop Security Design. *Yahoo, Inc., Tech. Rep* (2009).
[20] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. 1996. Role-based access control models. *IEEE Computer* 29, 2 (1996), 38–47.
[21] Ben Spivey and Joey Echeverria. 2015. *Hadoop Security. Protecting your Platform.* " O'Reilly Media, Inc.".
[22] Tom White. 2012. *Hadoop: The Definitive Guide.* " O'Reilly Media, Inc.".
[23] Chandhu Yalla et al. 2016. Big Data: Intel IT's Secure Hadoop Platform. (2016).