Hortonworks Technical Report 1

Adding Security to Apache Hadoop

Devaraj Das, Owen O'Malley, Sanjay Radia, Kan Zhang

www.Hortonworks.com
@hortonworks

Adding Security to Apache Hadoop

Devaraj Das*, Owen O'Malley*, Sanjay Radia*, and Kan Zhang**

*Hortonworks
**IBM

Abstract

Hadoop is a distributed system that provides a distributed filesystem and MapReduce batch job processing on large clusters using commodity servers. Although Hadoop is used on private clusters behind an organization's firewalls, Hadoop is often provided as a shared multi-tenant service and is used to store sensitive data; as a result, strong authentication and authorization is necessary to protect private data.

Adding security to Hadoop is challenging because all the interactions do not follow the classic client-server pattern: the file system is partitioned and distributed requiring authorization checks at multiple points; a submitted batch job is executed at a later time on nodes different from the node on which the client authenticated and submitted the job; job tasks from different users are executed on the same compute node; secondary services such as a workflow system access Hadoop on behalf of users; and the system scales to thousands of servers and tens of thousands of concurrent tasks.

To address these challenges, the base Kerberos authentication mechanism is supplemented by delegation and capability-like access tokens and the notion of trust for secondary services.

1 Overview

Apache Hadoop is a distributed system for storing large amounts of data and processing the data in parallel. Hadoop is used as a multi-tenant service internally at Yahoo! and stores sensitive data such as personally identifiable information or financial data. Other organizations, including financial organizations, using Hadoop are beginning to store sensitive data on Hadoop clusters. As a result, strong authentication and authorization is necessary. This paper describes the security issues that arise in Hadoop and how we address them. Hadoop is a complex distributed system that poses a unique set of challenges for adding security. While we rely primarily on Kerberos as our authentication mechanism, we supplement it with delegation tokens, capability-like access tokens and the notion of trust for auxiliary services.

1.1 Hadoop Background

Hadoop has been under development at Yahoo! and a few other organization as an Apache open source project over the last 5 years. It is gaining wide use in the industry. Yahoo!, for example, has deployed tens of Hadoop clusters, each typically with 4,000 nodes and 15 petabytes.

Hadoop contains two main components. The first component, HDFS [SKRC10], is a distributed file system similar to GFS [GGL03]. HDFS contains a metadata server called the NameNode that stores the hierarchical file and directory name space and the corresponding metadata, and a set of DataNodes that stores the individual blocks of each files. Each block, identified by a block id, is replicated at multiple DataNodes. Client perform file metadata operations such as *create file* and *open file*, at the NameNode over an RPC protocol and read/write the data of a file directly to DataNodes using a streaming socket protocol called the data-transfer protocol.

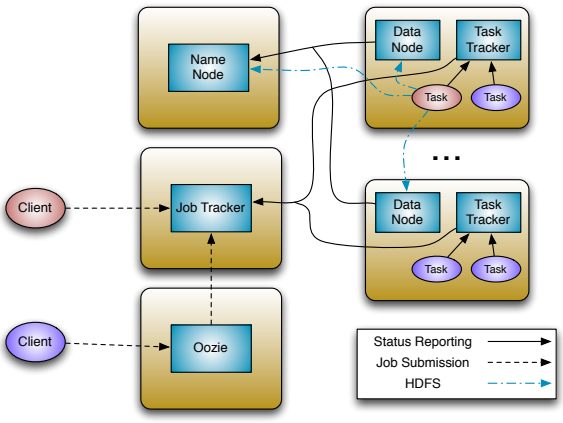

Figure 1: Hadoop High-level Architecture

The second component is a framework for processing large amounts of data in parallel using the MapReduce paradigm [DG04] . HDFS DataNodes also serve as compute nodes for MapReduce to allow computation to be performed close to the data being processed. A user submits a MapReduce job to the JobTracker which schedules the job to be executed on the compute nodes. Each compute node has a small daemon called the TaskTracker that launches map and reduce tasks of a job; the task tracker also serves intermediate data created by map tasks to reduce tasks.

There are additional services deployed with Hadoop, one of which are relevant to the security concerns discussed in this paper. Ooozie is a workflow system that provides a way to schedule and submit a DAG of MapReduce jobs that are triggered for execution by data availability or time.

1.2 Challenges in Adding Security to Hadoop

Adding security to Hadoop is challenging in that not all interactions follow the usual *client-server* pattern where the server authenticates the client and authorizes each operation by checking an ACL:

1. The scale of the system offers its own unique challenges. A 4000 node typical cluster is expected to serve over 100,000 concurrent tasks; we expect the cluster size and the number of tasks to increase over time. Commonly available Kerberos [SNS88] servers will not be able to handle the scale of so many tasks authenticating directly.

2. The file system is partitioned and distributed: the NameNode processes the initial open/create file operations and authorizes by checking file ACLs. The client accesses the data directly from the DataNodes which do not have any ACLs and hence have no way of authorizing accesses.

3. A submitted batch job is executed at a later time on nodes different from the node on which the user authenticated and submitted the job. Hence the users authentication at the job submission computer needs to be propagated for later use when the job is actually executed (note the user has usually disconnected from the system when the job actually gets executed.) This propagated credentials raises issues of trust and offers opportunities for security violation.

4. The tasks of a job need to securely obtain information such as task parameters, intermediate output of tasks, task status, etc. Intermediate Map output is not stored in HDFS; it is stored on the local disk of each compute node and is accessed via an HTTP-based protocol served by the Task-trackers.

5. Tasks from different tenants can be executed on the same machine. This shared environment offers opportunities for security violations via the APIs of the local operating system of the compute node: access to the intermediate output of other tenants, access to concurrent tasks of other jobs and access to the HDFSs local block storage on the node.

6. Users can access the system through auxiliary service such as Oozie, our work flow system. This raises new issues. For example, should the users credentials be passed through these secondary services or should these services be trusted by the rest of the Hadoop system?

Some of these problems are unique to systems like Hadoop. Others, as individual problems, occur in other systems. Where possible we have used standard approaches and combined them with new solutions. The overall set of solutions and how they interact, to our knowledge are fairly unique and instructive to a designer of complex distributed systems. We compare our work with those of others in Section 12.

2 Secure and Non-Secure Deployments of Hadoop

Hadoop is deployed in many different organizations; not all of them require a highly secure deployment. Although, to date, only Yahoo! has deployed Secure Hadoop clusters, many organizations are planning to convert their environment to a secure one. Hence Hadoop needs the options for being configured secure (with strong authentication) or non-secure. The non-secure configuration relies on client-side libraries to send the clients side credentials as determined from the client-side operating system as part of the protocol; while not secure, this configuration sufficient for many deployment that rely on physical security. Authorization checks through ACLs and file permissions are still performed against the client supplied user-id. A secure deployment requires that one configures a Kerberos [SNS88] server; this paper described the mechanisms and policies for such a secure deployment.

2.1 The Physical Environment

Each Hadoop cluster at Yahoo! is independently managed and connected to the wider enterprise network. The security policy of our organization dictates that each cluster has gateways through which jobs can be submitted or from which HDFS can be accessed; the gateways firewall each Hadoop cluster. Note this is a particular policy of our organization and not a restriction of Hadoop: Hadoop itself allows access to HDFS or MapReduce from any client that can reach it via the network.

Each node in the cluster is physically secure and is loaded with the Hadoop software by system administrators; users do not have direct access to the nodes and cannot install any software on these nodes. Users can however login on a cluster's gateway nodes. A user cannot become a superuser (root) on any of the cluster nodes. Further, a user cannot connect a non-cluster node (such as user workstation) to the cluster network and snoop on the network.

Although HDFS and and MapReduce clusters are architecturally separate, the clusters are typically configured to superimpose to allow computation to be performed close to its data.

3 Usage Scenarios

1. **Applications accessing HDFS** An application running on a cluster gateway, a cluster compute node, or any computer that can connect to the HDFS cluster that accesses HDFS files.

2. **User submitting jobs to MapReduce clusters** A user submits jobs to a MapReduce queue of a cluster. The user can disconnect after job submission and may re-connect to get job status. The job, when later executed, accesses HDFS files as in Usage Scenario 1.

3. **User submitting workflows to Oozie** A user submits a workflow to Oozie. Due to data trigger or time trigger, the workflow is later executed as Mapreduce jobs (as in Usage Scenario 2) on behalf of the user that submitted the workflow.

4. **Headless account doing use cases I, 2, 3**

4 Security Threats

Voydock and Kent [VK83] identify three categories of security violation: unauthorized release of information, unauthorized modification of information and denial of resources. We focus on only the first two as part of our security solution. The following identify the related areas of threat in Hadoop:

- An unauthorized client may access an HDFS file via the RPC or via HTTP protocols.
- A unauthorized client may read/write a data block of a file at a DataNode via the pipeline-streaming data-transfer protocol
- A unauthorized user may submit a job to a queue or delete or change priority of the job.
- A unauthorized client may access intermediate data of Map job via its task trackers HTTP shuffle protocol.
- An executing task may use the host operating system interfaces to access other tasks, access local data which include intermediate Map output or the local storage of the DataNode that runs on the same physical node.
- A task or node may masquerade as a Hadoop service component such as a DataNode, NameNode, job tracker, task tracker etc.
- A user may submit a workflow to Oozie as another user.

5 Users, Groups and Login

We wanted to avoid creating user accounts for each Hadoop user in the HDFS and MapReduce subsystems. Instead Hadoop uses the existing user accounts of the hosting organization and Hadoop depends on external user credentials (e.g. OS login, Kerberos credentials, etc).

Hadoop supports single signon. For a non-secure deployment, the user needs to merely login into the computer from Hadoop is accessed. For secure deployment, the system login is supplemented by Kerberos signon. Any subsequent access to Hadoop carries the credentials for authentication.

As part of client authentication to a Hadoop service, the user is identified by a user-id string. A user-id is not converted to a uid number as in traditional Unix or mapped to a "Hadoop-account". During an authorization check, the user-id is compared with those in ACL entries. Operations that create new objects (say files) use the same user-id for object ownership.

Hadoop also supports a notion of group membership; group membership is determined on the server-side through a plugin that maps a user-id to its respective a set of group-ids of the hosting organization.

There is special group identified as a supergroup; its members are administrators who are permitted all operations. There is no separate notion of a superuser. This has the advantage that our audit logs show the actual user-ids of administrators.

MapReduce executes job tasks as the user who submitted the job and hence each compute node needs to have an account for each user (more details below).

Hadoop services themselves (NameNode, DataNode, JobTracker etc) are also configured with suitable credentials to authenticate with each other and with clients so that clients can be sure that they are communicating with Hadoop service component.

6 Authorization and ACLs

HDFS offers Unix-like permission with *owner*, *group* and *other* permission bits for each file and directory. We considered using only ACLs, but decided that our users were more familiar with the Unix model and that we could add ACLs later as in the Andrew File System [Sat89].

MapReduce offers ACLs for job queues; they define which users or groups can submit jobs to a queue and change queue properties.

The *other* permission of a file or job queue ACL would generally mean *any* user. This causes concerns for some deployments since they want to limit the cluster usage to a subset of their employees. We address this concern via the notion of a Service-ACL as first level check for a Hadoop service such as the NameNode or JobTracker. Only users or groups in the Service-ACL of the service can proceed on with a service operation. The Service-ACL effectively defines the set of users denoted by the *other* permission. Deployments that dont need to restrict the set of users or restrict the *other* permission do not need to configure the Service- ACL.

7 Authentication

7.1 Kerberos as the Primary Authentication

We chose Kerberos [SNS88] for the primary authentication in a secure deployment of Hadoop . We also complement it with additional mechanisms as explained later. Another widely used mechanism is SSL. We choose Kerberos over SSL for the following reasons.

1. **Better performance** Kerberos uses symmetric key operations, which are orders of magnitude faster than public key operations used by SSL.
2. **Simpler user management** For example, revoking a user can be done by simply deleting the user from the centrally managed Kerberos Key Distribution Center (KDC). Whereas in SSL, a new certificate revocation list has to be generated and propagated to all servers.

7.2 Tokens as Supplementary Mechanisms

Security Tokens supplement the primary kerberos authentication in Hadoop. Currently, there are three forms of tokens:

1. **Delegation token** is a token that identifies a user to a particular HDFS or MapReduce service. It addresses the problem of propagating the authentication performed at job submission time to when the job is later executed. Details are given in section 8.1
2. **Block access token** is a capability that gives read or write permission to a particular HDFS block. It addresses the problem that a DataNodes do not have any ACLs against which to authorize block access. Details are given in section 8.3
3. A **Job token** identifies a MapReduce job its tasks. Details are given in 9.3.1

Our RPC subsystem, described in 10.1, is pluggable to allow the kerberos credentials or these tokens for authentication.

8 HDFS

Communication between the client and the HDFS service is composed of two halves:

- A RPC connection from the client to the NameNode to, say, open or create a file. The RPC connection can be authenticated via Kerberos or via a delegation token. If the application is running on a computer where the user has logged into Kerberos then Kerberos authentication is sufficient. Delegation token is needed only when an access is required as part of MapReduce job.

  After checking permissions on the file path, NameNode returns blockids, block locations and block access tokens.

- A streaming socket connection is used to read or write the block of a file at a DataNode. A

datanode requires the client to supply a block-access token generated by the NameNode for the block being accessed.

Communications between the NameNode and DataNodes is via RPC and mutual Kerberos authentication is performed. Figure 2 shows the communication paths and the mechanism used to authenticate these paths.

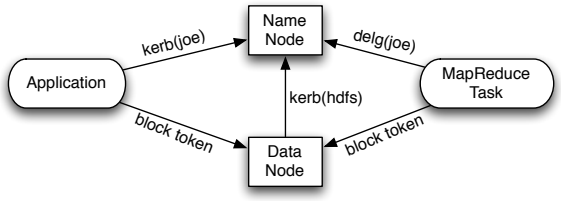

Figure 2: HDFS authentication

8.1 Delegation Token

A user submitting a job authenticates with the Job Tracker using Kerberos. The job is executed later, possibly after the user has disconnected from the system. How do we propagate the credentials? There are several options:

- have the user pass the password to the job tracker
- pass the Kerberos credentials (TGT or service ticket for the NameNode)
- use a special delegation token

Passing the password is clearly unacceptable. we choose to use a special delegation token instead of passing the Keberos credentials (reasons explained below).

After initial authentication to NameNode using Kerberos credentials, a client obtains a delegation token, which is given to a job for subsequent authentication to NameNode. The token is in fact a secret key shared between the client and NameNode and should be protected when passed over insecure channels. Anyone who gets it can impersonate as the user on NameNode. Note that a client can **only** obtain new delegation tokens by authenticating using Kerberos.

The format of delegation token is:

```
TokenID = {ownerID, renewerID, issueDate,
           maxDate, sequenceNumber}
```

When a client obtains a delegation token from NameNode, it specifies a renewer that can renew or cancel the token. By default, delegation tokens are valid for 1 day from when they are issued and may be renewed up to a maximum of 7 days. Because MapReduce jobs may last longer than the validity of the delegation token, the JobTracker is specified as the renewer. This allows the JobTracker to renew the tokens associated with a job once a day until the job completes. When the job completes, the Job Tracker requests the NameNode to cancel the job's delegation token. Renewing a delegation token does not change the token, it just updates its expiration time on the NameNode, and the old delegation token continues to work in the MapReduce tasks.

The NameNode uses a secret key to generate delegation tokens; the secret is stored persistently on the NameNode. The persistent copy of the secret is used when the NameNode restarts. A new secret is rolled every 24 hours and the last 7 days worth of secrets are kept so that previously generated delegation tokens will be accepted. The generated token is also persistently.

8.2 Advantages of the Delegation Token

Delegation tokens have some critical advantages over using the Kerberos TGT or service ticket:

1. **Performance** On a MapReduce cluster, there can be thousands of tasks running at the same time. Many tasks starting at the same time could swamp the KDC creating a bottleneck.

2. **Credential renewal** For tasks to use Kerberos, the task owner's Kerberos TGT or service ticket needs to be delegated and made available to the tasks. Both TGT and service ticket can be renewed for long-running jobs (up to max lifetime set at initial issuing). However, during Kerberos renewal, a new TGT or service ticket will be issued, which needs to be distributed to all running tasks. With delegation tokens, the renewal mechanism is designed so that the validity period of a token can be extended on the NameNode, while the token itself stays the same. Hence, no new tokens need to be issued and pushed to running tasks.

3. **Less damage when credential is compromised** A user's Kerberos TGT may be used to access services other than HDFS. If a delegated TGT is used and compromised, the damage is

greater than using an HDFS-only credential (delegation token). On the other hand, using a delegated service ticket is equivalent to using a delegation token.

Kerberos is a 3-party protocol that solves the hard problem of setting up an authenticated connection between a client and a server that have never communicated with each other before (but they both registered with Kerberos KDC). Our delegation token is also used to set up an authenticated connection between a client and a server (NameNode in this case). The difference is that we assume the client and the server had previously shared a secure connection (via Kerberos), over which a delegation token can be exchanged. Hence, delegation token is essentially a 2-party protocol and much simpler than Kerberos. However, we use Kerberos to bootstrap the initial trust between a client and NameNode in order to exchange the delegation token for later use to set up another secure connection between the client (actually job tasks launched on behalf of the client) and the same NameNode.

8.3 Block Access Token

A block access tokens is a capability that enables its holder to access certain HDFS data blocks. It is issued by NameNode and used on DataNode. Block access tokens are generated in such a way that their authenticity can be verified by DataNode.

Block access tokens are meant to be lightweight and short-lived. They are not renewed, revoked or stored on disk. When a block access token expires, the client simply gets a new one.

A block access token has the following format, where **keyID** identifies the secret key used to generate the token, and **accessModes** can be any combination of READ, or WRITE.

```
TokenID = {expirationDate, keyID, ownerID,
           blockID, accessModes}
```

A block access token is valid on all DataNodes and hence works for all the replicas of a block and even if the block replica is moved. The secret key used to compute token authenticator is randomly chosen by NameNode and sent to DataNodes when they first register with NameNode and periodically updated during heartbeats. The secrets are not persistent and a new one is generated every 10 hours. The block access token is sent from the NameNode to the client when it opens or creates a file. The client sends the entire token to the DataNode as part of a read or write request.

9 MapReduce

A MapReduce job involves the following stages (as depicted in Figure 3:

1. A client connects to the JobTracker to request a job id and an HDFS path to write the job definition files. The MapReduce library code writes the details of the job into the designated HDFS staging directory and acquires the necessary HDFS delegation tokens. All of the job files are visible only to the user, but depend on HDFS security.

2. The JobTracker receives the job and creates a random secret, which is called the job token. The job token is used to authenticate the job's tasks to the MapReduce framework.

3. Jobs are broken down into tasks, each of represents a portion of the work that needs to be done. Since tasks (or the machine that they run on) may fail, there can be multiple attempts for each task. When a task attempt is assigned to a specific TaskTracker, the TaskTracker creates a secure environment for it. Since tasks from different users may run on the same compute node, we have chosen to use the host operating system's protection environment and run the task as the user. This enables use of the local file system and operating system for isolation between users in the cluster. The tokens for the job are stored in the local file system and placed in the task's environment such that the task process and any sub-processes will use the job's tokens.

4. Each running task reports status to its TaskTracker and reduce tasks fetch map output from various TaskTrackers. All of these accesses are authenticated using the job token.

5. When the job completes (or fails), all of the HDFS delegation tokens associated with the job are revoked.

9.1 Job Submission

A client submitting a job or checking the status of a job authenticates with the JobTracker using Kerberos over RPC. For job submission, the client writes the job configuration, the job classes, the input splits, and the meta information about the input splits into a directory, called the Job Staging directory, in their home directory. This directory is protected as read, write, and execute solely by the user.

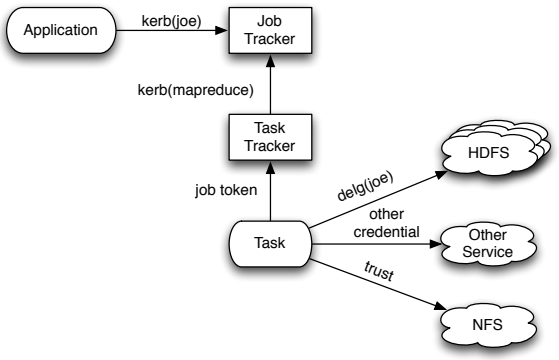

Figure 3: MapReduce authentication

Jobs (via its tasks) may access several different HDFS and other services. Therefore, the job needs to package the security credentials in a way that a task can later look up. The job's delegation tokens are keyed by the NameNode's URL. The other credentials, for example, a username/password combination for a certain HTTP service are stored similarly.

The client then uses RPC to pass the location of the Staging directory and the security credentials to the JobTracker.

The JobTracker reads parts of the job configuration and store it in RAM. In order to read the job configuration, the JobTracker uses the user's delegation token for HDFS. The JobTracker also generates a random sequence of bytes to use as the job token, which is described in section 9.3.1.

The security credentials passed by the client, and, the job token are stored in the JobTracker's system directory in HDFS, which is only readable by the 'mapreduce' user. To ensure that the delegation tokens do not expire, the JobTracker renews them periodically. When the job is finished, all of the delegation tokens are invalidated.

9.2 Job and Task localization

TaskTrackers runs tasks of the jobs, and before running the first task of any job, it needs to set up a secure environment for the same. The TaskTracker launches a small setuid program (a relatively small program written in C) to create the job directory, and make the owner of that job directory the job-owner user. A setuid program is used since root privileges are required to assign ownership of the directory to some other user. The setuid program also copies the credentials file from the JobTracker's system directory into this directory. The setuid program then switches back to the job-owner user, and does the rest of the localization work. That includes - copying of the job files (configuration and the classes) from the job-owner's Staging directory. It uses the user's delegation token for HDFS from the credentials file for these.

As part of the task launch, a task directory is created per task within the job directory. This directory stores the intermediate data of the task (for example, the map output files).

The group ownership on the job directory and the task directories is set to the 'mapreduce' user. The group ownership is set this way so that the TaskTracker can serve things like the map outputs to Reducers later. Note that only the job-owner user and the TaskTracker can read the contents of the job directory.

9.3 Task

The task runs as the user who submitted the job. Since the ability to change user ids is limited to root the setuid program is used here too. It launches the task's JVM as the correct job-owner user. The setuid program also handles killing the JVM if the task is killed. Running with the user's user id ensures that one user's job can not send operating system signals to either the TaskTracker or other user's tasks. It also ensures that local file permissions are sufficient to keep information private.

9.3.1 Job Token

When the job is submitted, the JobTracker creates a secret key that is only used by the tasks of the job when identifying themselves to the framework. As mentioned earlier, this token is stored as part of the credentials file in the JobTracker's system directory on HDFS. This token is used for the RPC via DIGEST-MD5 when the Task communicates with the TaskTracker to requests tasks or report status.

Additionally, this token is used by Pipes tasks, which run as sub-processes of the MapReduce tasks. Using this shared secret, the child and parent can ensure that they both have the secret.

9.4 Shuffle

When a map task finishes, its output is given to the TaskTracker that managed the map task. Each reduce in that job will contact the TaskTracker and fetch its section of the output via HTTP. The framework needs to ensure that other users may not obtain the map outputs. The reduce task will compute the HMAC-SHA1 of the requested URL and the current timestamp and using the job token as the secret. This HMAC-SHA1 will be sent along with the request

and the TaskTracker will only serve the request if the HMAC-SHA1 is the correct one for that URL and the timestamp is within the last N minutes.

To ensure that the TaskTracker hasn't been replaced with a trojan, the response header will include a HMAC-SHA1 generated from the requesting HMAC-SHA1 and secured using the job token. The shuffle in the reduce can verify that the response came from the TaskTracker that it initially contacted.

The advantage of using HMAC-SHA1 over DIGEST-MD5 for the authentication of the shuffle is that it avoids a roundtrip between the server and client. This is an important consideration since there are many shuffle connections, each of which is transferring a small amount of data.

9.5 MapReduce Delegation Token

In some situations, an executing job needs to submit another job; for this it needs to authenticate with the Job Tracker. For this we use a another Delegation token similar to the one generated by the NameNode as described in section 8.1. Its generation, persistence and use is identical to HDFS delegation token, except that the Job-Tracker generates it. It is also obtained at the initial job submission time.

9.6 Web UI

A critical part of MapReduce's user interface is via the JobTracker's web UI. Since the majority of users use this interface, it must also be secured. We implemented a pluggable HTTP user authentication mechanism that allows each deploying organization to configure their own browser based authentication in the Hadoop configuration files. At Yahoo!, that plugin uses Backyard Authentication.

Once the user is authenticated, the servlets will need to check the user name against the owner of the job to determine and enforce the allowable operations. Most of the servlets will remain open to everyone, but the ability view the stdout and stderr of the tasks and to kill jobs and tasks will be limited to the job-owner.

10 Implementation Details

10.1 RPC

Hadoop clients access most Hadoop services via Hadoop's RPC library. In insecure versions of Hadoop, the user's login name is determined from the client OS and sent across as part of the connection setup and are not authenticated; this is insecure because a knowledgeable client that understand the protocol can substitute any user-id. For authenticated clusters, all RPC's connect using Simple Authentication and Security Layer (SASL). SASL negotiates a sub-protocol to use and Hadoop will support either using Kerberos (via GSSAPI) or DIGEST-MD5. Most applications run on the gateways use Kerberos tickets, while tasks in MapReduce jobs use tokens. The advantage of using SASL is that using a new authentication scheme with Hadoop RPC would just require implementing a SASL interface and modifying a little bit of the glue in the RPC code.

The supported mechanisms are:

1. **Kerberos** The user gets a service ticket for the service and authenticates using SASL/GSSAPI. This is the standard Kerberos usage and mutually authenticates the client and the server.

2. **DIGEST-MD5** When the client and server share a secret, they can use SASL/DIGEST-MD5 to authenticate to each other. This is much cheaper than using Kerberos and doesn't require a third party such as the Kerberos KDC. The two uses of DIGEST-MD5 will be the HDFS and MapReduce delegation tokens in sections 8.1 and 9.5 and the MapReduce job tokens in section 9.3.1.

The client loads any Kerberos tickets that are in the user's ticket cache. MapReduce also creates a token cache that is loaded by the task. When the application creates an RPC connection, it uses a token, if an appropriate one is available. Otherwise, it uses the Kerberos credentials.

Each RPC protocol defines the kind(s) of token it will accept. On the client-side, all tokens consist of a binary identifier, a binary password, the kind of the token (delegation, block access, or job), and the particular service for this token (the specific JobTracker or NameNode).

The token identifier is the serialization of a token identifier object on the server that is specific to that kind of token. The password is generated by the server using HMAC-SHA1 on the token identifier and a 20 byte secret key from Java's SecureRandom class. The secret keys are rolled periodically by the server and, if necessary, stored as part of the server's persistent state for use if the server is restarted.

11 Auxiliary Services

There are several Auxiliary services, such as Oozie (a workflow system), that act as proxies for user requests

of the Hadoop services. Workflows can be trigger by time (as in Cron) or data availability. Production workflows are usually configured to run at specific times or based on data triggers. A workflow needs to be executed on behalf of the user that submitted the workflow.

This problem is similar to that of job submission where we used the delegation token. Reusing the delegation token design was problematic because a workflow can be registered in the system for essentially forever, being triggered periodically. This would mean that the delegation token for workflows would need to be renewed perpetually. As a result we instead decided use the notion of trust. Hadoop services such as HDFS and MapReduce allow one to declare specific proxy services as being trusted in the sense that they have authenticated with user and can be trusted to act on behalf of the user. We have other auxiliary services that also act on behalf of users and are trusted by Hadoop.

An obvious question is that given that the notion of trust was added, why didn't we use it instead of the delegation token. We wanted to limit which services are trusted since extending a trust relationship is a serious undertaking. In case of MapReduce jobs, we could practically limit the lifetime of job in queue to 7 days and further could cancel the delegation token when the job completed.

The HDFS and MapReduce services are configured with a list of principals that are *proxy-users*, which are trusted to act as other users. The configuration file also specifies set of IP addresses from which the proxy-user can connect. The proxy service (Oozie in this case) authenticates as itself, but then access functionality as if it is another user. We allow one to specify for each proxy-user principal the group of users on whose behalf the proxy may act.

When the proxy-user makes an RPC connection, the RPC connection will include both the effective and real user of the client. The server rejects connections that: request a user that is not a valid user for that service, request a user that is not in the user group that is allowed for that proxy-user, or originate from an IP address other than the blessed set for that proxy-user.

12 Related Work

Hadoop is a complex distributed system that poses a unique set of challenges for adding security. Some of these problems are unique to systems like Hadoop. Others, as individual problems, occur in other systems. Where possible we have used standard approaches and combined them with new solutions. The overall set of solutions and how they interact, to our knowledge are fairly unique and instructive to a designer of complex distributed systems.

Hadoop's central authentication mechanism is Kerberos; it is supplemented by delegation tokens, capability-like access tokens and the notion of trust for auxiliary services. The delegation token addresses the problem of propagating the user's credentials to an executing job. We compared the mechanisms used to generate, exchange, renew and cancel the delegation token with those of Kerberos and its service ticket in Section 8.1.

To address the shared execution environment we use the security protection offered by the host system on which the job tasks execute; this is common in many parallel execution systems. Since Hadoop offers additional services like HDFS and temporary local storage we cannot simply rely on the authentication mechanisms offered by the host operating system; our delegation and other tokens allow a task to securely access these additional services. Condor, for example, choose to push the delegation problem to the applications themselves.

Like many systems such as Cedar [Bir85], we chose to provide authentication at the RPC transport level. This insulates the application logic from authentication details. It is also a place to plug in alternate authentication mechanisms as is the case with our tokens.

12.1 Condor

The *Condor* system[TTL05] is a distributed HPC system that like MapReduce runs user applications on a cluster of machines. Condor supports a variety of authentication schemes including PKI-based GSI[FKTT98] and Kerberos. Although Kerberos credentials are accepted, those credentials are not passed along to the job's workers. Therefore, jobs that need Kerberos tickets to access resources such as the Andrew File System[HKM+88] need to propagate their Kerberos tickets explicitly.

Condor's authentication library creates and caches session keys for each client to improve performance[MBTS10] to prevent re-authenticating on each client-server connection. Although these session keys fill a similar role to our delegation tokens, they are significantly different. In particular, the session keys are not distributed between clients and do not encode the user's information.

Condor's authentication mechanisms support proxied users between trust domains. This is an important use case for HPC system where grids are of-

ten composed of machines that are owned by different organizations. Hadoop clusters, on the other hand, have very high network bandwidth requirements within the cluster and never span data centers or organizations. Therefore, Hadoop has no requirement to support cross trust domain authentication.

12.2 Torque

Torque, which is another HPC grid solution, uses the host operating system accounts for authentication. To submit jobs, users must connect to the machine running the master scheduler using the standard ssh command. Because the user effectively logs into the master machine, the server can check the user's login name directly.

Alternatively, Torque supports MUNGE authentication from `http://code.google.com/p/munge/`. MUNGE is an authentication service for HPC environments that supports remote detection of another process' user or group ids. Both the client and server host must be part of the same security realm that have a shared secret key.

13 Conclusions

Hadoop, the system and its usage grew over the last 5 years. The early experimental use did not require security and there weren't sufficient resources to design and implement it. However as Hadoop's use grew at Yahoo!, security became critical. Hadoop's elastic allocation of resources and the scale at which it is is typically deployed lends Hadoop to be shared across tenants where security is critical if any stored data is sensitive. Segregating sensitive data and customers into private clusters was not practical or cost effective. As a result, security was recently added to Hadoop in spite of the axiom that states it is best to design security in from the beginning. Yahoo! invested 10 person years over the last two years to design and implement security. The adoption of Hadoop in commercial organizations is beginning and security is a critical requirement.

Secure versions of Hadoop have been deployed at Yahoo! for over a year. In that time, only a handful of security bugs have been discovered and fixed. We were worried that security would add operational burden; fortunately this was not the case. Kerberos has also worked out well for us since our user accounts were already in Kerberos. Our benchmarks show that performance degradation has been less then 3%.

Our very early decision, even prior to adding security, was to use the existing user accounts of the organization in which Hadoop is deployed rather then have a separate notion of Hadoop accounts. This made it easier for someone to quickly try and use Hadoop, a critical factor in the growth and success of Hadoop. For us, the use of existing Kerberos accounts for our users was very convenient.

We were wise to supplement Kerberos with tokens as Kerberos servers would not have scaled to tens of thousands of concurrent Hadoop tasks. During the design of delegation tokens, we had considered the alternative of extending a Kerberos implementation to incorporate mechanisms so that a Kerberos service ticket could be delegated and renewed in the fashion needed for Hadoop. It was a good decision to not do that as it would have been challenging to have a modified Kerberos server adopted for non-Hadoop use in our organization. Further it would have hindered the wider-adoption of secure Hadoop in the industry.

Hadoop is implemented in Java. While Java is known for providing good security mechanisms, they are centered mostly around sandboxing rather then for writing secure services. Our security mechanisms, especially in the RPC layer, are designed to be pluggable; we believe it should be fairly easy to, say replace, our primary authentication mechanism Kerberos with another mechanism. Our approach has the advantage that one could continue to use our tokens to supplement a different primary authentication mechanism.

14 Acknowledgements

We would like to thank Christopher Harrell and Ram Marti of Yahoo! Paranoid team who did a detailed review of Hadoop's security design. We would also like to thank members of the security development team Jakob Homan, Jitendra Pandley, and Boris Shkolnik.

References

[Bir85] Andrew D. Birrell. Secure communication using remote procedure calls. *ACM Trans. Comput. Syst.*, 3:1–14, February 1985.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[FKTT98] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM Conference*

on Computer and Communications Security Conference, pages 83–92, 1998.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[HKM+88] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst.*, 6:51–81, February 1988.

[MBTS10] Zach Miller, Dan Bradley, Todd Tannenbaum, and Igor Sfiligoi. Flexible session management in a distributed environment. *Journal of Physics: Conference Series*, 219(4):042017, 2010.

[Sat89] Mahadev Satyanarayanan. Integrating security in a large distributed system. *ACM Trans. Comput. Syst.*, 7(3):247–280, 1989.

[SKRC10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. *Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference on*, 0:1–10, 2010.

[SNS88] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *in Usenix Conference Proceedings*, pages 191–202, 1988.

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[VK83] Victor L. Voydock and Stephen T. Kent. Security mechanisms in high-level network protocols. *ACM Comput. Surv.*, 15(2):135–171, 1983.