KIP-38: ZooKeeper Authentication

Status

Current state: *Accepted*

Discussion thread: *apache mail archive link*

JIRA: *KAFKA-2639, KAFKA-2640, KAFKA-2641*

Released: 0.9.0.0

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

As part of the effort to introduce and strengthen the security features of Kafka, we propose a set of changes to authenticate access to the metadata stored in ZooKeeper. Currently, the metadata stored in ZooKeeper for any given Kafka cluster is open and can be manipulated by any client with access to the ZooKeeper ensemble. The goal of this KIP is to restrict access to authenticated clients by leveraging the SASL authentication feature available in the 3.4 branch of ZooKeeper. With this feature, clients authenticate upon connecting to a ZooKeeper server with security configured and enabled. Once authenticated, the ensemble uses the client credentials to authorize access to znodes that have ACLs configured to restrict access.

The current ACL used with all znodes is open for read and write access, and with this KIP we want to make it unrestricted for authenticated clients, but readable by any client. We assume that no data stored in ZooKeeper is sensitive and can be read by anyone with the ability of connecting to the ensemble, but the metadata stored in it can be used to mount specific attacks against the cluster. For example, the metadata of a broker can be manipulated or a rogue broker can be introduced and start participating in replica sets as any other broker. Note that the feature described here focuses on brokers and does not include the old consumer, since it is becoming deprecated.

Public Interfaces

This proposal assumes that all changes to APIs are internal and there are visible changes to users only through configuration parameters. Users need to setup a JAAS login configuration file and specify it as a system property named `java.security.auth.login.config`. With this property set, Kafka brokers turn security features on and use more strict ACLs rather than the open unsafe one. Specifically, it uses `CREATOR_ALL_ACL` and `READ_ACL_UNSAFE` when the security feature is on, which enable the creator (or anyone with the credentials of the creator) to manipulate the znode while everyone else can read it. That's the only bit that changes to users, everything else happens under the hood.

ZooKeeper currently offers two mechanisms of authentication: `Kerberos` and `DIGEST-MD5`. For more information on the options, check the following online docs:

- Apache ZooKeeper documentation
- Apache ZooKeeper wiki
- Cloudera ZooKeeper security configuration

Paths affected

The following paths will have secure ACLs once this feature is turned on:

```
val BrokerIdsPath = "/brokers/ids"
val BrokerTopicsPath = "/brokers/topics"
val ControllerPath = "/controller"
val ControllerEpochPath = "/controller_epoch"
val ReassignPartitionsPath = "/admin/reassign_partitions"
val DeleteTopicsPath = "/admin/delete_topics"
val PreferredReplicaLeaderElectionPath = "/admin/preferred_replica_election"
val BrokerSequenceIdPath = "/brokers/seqid"
val IsrChangeNotificationPath = "/isr_change_notification"
val EntityConfigPath = "/config"
val EntityConfigChangesPath = "/config/changes"
```

Command-line tools

Command line tools need to have the system property `java.security.auth.login.config` set pointing to a file with the appropriate credentials so that it has write access to znodes.

Proposed Changes

There are three parts to this new feature:

- **ZkUtils**: Pretty much all access to ZooKeeper happens through kafka.utils.ZkUtils and the JIRA for these changes is KAFKA-2639. The main problem with the code before the changes in KAFKA-2639 is the fact that ZkUtils is a singleton object, which makes changing the ACLs from not secure to secure difficult. We need to be able to change ACLs dynamically at least for testing, however. In KAFKA-2639, we make ZkUtils a class and now we can determine if we want to use strict or open ACLs at instantiation.
- **Configuration and upgrades**: The user needs to be able to input configuration to express the interest in running a secure cluster and needs an upgrade path to bring non-secure clusters to secure. The main part of the configuration is in the JAAS login file, and we pass it to a broker via a system property. The upgrade path is discussed below, and this part of the proposal is in KAFKA-2641.
- **Tests**: This task is mainly about changing existing test cases and creating new ones to test new functionality. This part of the work is in KAFKA-2640.

Compatibility, Deprecation, and Migration Plan

Unsecure cluster -> Secure cluster

If a broker all of a sudden changes the ACLs of znodes to secure, then brokers without the proper configuration will not be able to access ZooKeeper data any longer. Consequently, we need a plan to migrate existing clusters without requiring such a cluster to stop to reconfigure. We currently propose the following recipe:

1. Perform a rolling restart setting the JAAS login file, which enables brokers to authenticate. At the end of the rolling restart, brokers are able to manipulate znodes with strict ACLs, but they will not create znodes with those ACLs.
2. Perform a second rolling restart of brokers, this time setting the configuration parameter `zookeeper.set.acl` to true, which enables `ZkUtils` to use secure ACLs when creating znodes.
3. Execute a tool called `ZkSecurityMigrator` (there is a script under `./bin` and the code is under `kafka.admin`). This tool traverses the corresponding sub-trees changing the ACLs of the znodes.

The ZooKeeper ensemble also needs to be configured to accept authenticated clients. We perform a rolling restart of ZooKeeper servers and for each server we need to set two parameters: the authentication provider and the login renew like this:

```
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
jaasLoginRenew=3600000
```

Note that ZooKeeper is not going to drop connections for unauthenticated clients. Such clients are still able to connect and read ZooKeeper content, but they cannot manipulate (`create`, `setData`, or `delete`) znodes with the secure ACL. One ZooKeeper setting of interest on the server side is `zookeeper.allowSaslFailedClients`. If this is false, then clients trying to authenticate with an incorrect configuration will have their connections dropped. Otherwise, such clients will be able to connect successfully, but will not have the right credentials set. Setting it to false prevents clients with an incorrect configuration from making progress.

ZooKeeper also allows users to disable authentication on the client side even in the presence of a JAAS login file with the property `zookeeper.sasl.client`. Setting it to false disables client authentication. Additionally, the context key in the JAAS login file is "Client" by default, but that name can be changed by using setting the property `zookeeper.sasl.clientconfig`.

Secure cluster -> Unsecure cluster

It is desirable to have the ability of bringing a cluster from secure to unsecure, making all znodes open to reads and writes by any client able to connect to the ZooKeeper ensemble. This option might be needed for example when there are issues with security features and they need to be off to allow the cluster to keep operating.

The path to execute this change is the following:

1. Perform a rolling restart setting the JAAS login file, which enables brokers to authenticate, but setting `zookeeper.set.acl` to false. At the end of the rolling restart, brokers stop creating znodes with secure ACLs, but are still able to authenticate and manipulate all znodes.
2. Execute the `ZkSecurityMigrator` (there is a script under `./bin` and the code is under `kafka.admin`). This tool traverses the corresponding sub-trees changing the ACLs of the znodes.
3. Perform a second rolling restart of brokers, this time omitting the system property that sets the JAAS login file.

Step 3 isn't strictly necessary, but it is best to just turn off authentication.

Shared ensemble

The changes should not prevent multiple applications sharing an ensemble. One typical way of sharing a ZooKeeper ensemble is to use the chroot feature to create different sub-trees for applications.

The currentl implementation sets an ACL to the root of the Kafka cluster sub-tree, which restricts operations on znodes like deleting children. If chroot is not used, then the ability of sharing the ensemble is limited and will possibly require manual intervention.

Credentials

It should enable different credentials for admin tools and brokers. Sharing credentials between brokers and tools might not be desirable from an operational perspective and we need to enable different credentials to manipulate the appropriate ZooKeeper state (znodes). This aspect isn't entirely clear and is one feature that is not implemented yet.

Rejected Alternatives

One way to restrict access to a ZooKeeper ensemble is to use firewalls. This approach is reasonable, but difficult to implement in a fine-grained manner, which ends up leaving the metadata exposed all the same. Usin g traffic filtering to complement the feature described here is certainly a recommended option, so this option is not really rejected, but it is deemed insufficient.