KIP-515: Enable ZK client to use the new TLS supported authentication

Status

Current state: *"Adopted" (in version 2.5)*

Discussion thread: *here*

JIRA: ⚠ Unable to render Jira issues macro, execution error.

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

Motivation

ZooKeeper TLS Functionality

Since the merge KAFKA-8634 (https://github.com/apache/kafka/commit/d67495d6a7f4c5f7e8736a25d6a11a1c1bef8d87) in trunk, Apache Kafka ships with Apache Zookeeper supporting TLS and Dynamic Reconfiguration (AK 2.4 ultimately shipped with ZooKeeper version 3.5.6 rather than 3.5.5, but the general functionality is the same). When doing a deployment in a security-minded environment the desire is to use TLS to encrypt communication in transit.

Note that the current version of ZooKeeper (3.5.6 as of this writing and the version shipped with Apache Kafka 2.4) only supports **mutual** certificate authentication. There is a sever-side config "`ssl.clientAuth`" that the ZooKeeper code recognizes (case-insensitively: `want`/`need`/`none` are the valid options), but this config has no effect in 3.5.6 (⚠ Unable to render Jira issues macro, execution error.). This is fixed in version 3.5.7, which is the ZooKeeper version that will ship with Apache Kafka version 2.5.

Note also that ZooKeeper will associate multiple identities with any session that successfully authenticates multiple ways (e.g. both client certificate and SASL). The X.509 identity is the full Distinguished Name from the client's certificate, and this can be changed (i.e. use just a part of the DN) only by implementing and using a custom ZooKeeper authentication provider that overrides the method `protected String getClientId (X509Certificate clientCert)`. A client that accesses an ACL-protected Znode is authorized if it has at least 1 of the identities present in any authorizing ACL.

ZooKeeper also supports TLS connectivity between ZK nodes for Quorum-related communication. This is configured independent of Kafka within ZooKeeper.

It is possible to enable TLS connectivity to Zookeeper from Apache Kafka 2.4 -- the problem is that configuration information has to be passed via system properties as -D command line options on the Java invocation of the broker or CLI tool (e.g. ZooKeeper Security Migration), and such -D command line options are not secure because anyone with access to the box can see the command line used to start the process; the configuration includes sensitive keystore/truststore password information, so we need a secure mechanism for passing the configuration values. The motivation for this KIP is to harden /secure the configuration mechanism for Zookeeper TLS connectivity.

With this KIP we aim to introduce the necessary changes to enable the use of secure configuration values when defining TLS encrypted channels for communications with Zookeeper. These changes will enable the secure use of TLS from brokers as well as any CLI tools that require it in the next AK (AK 2.5) release.

Brokers

Brokers talk to ZooKeeper, of course. In addition, the class `kafka.security.authorizer.AclAuthorizer` talks directly to ZooKeeper and supports being pointed to a separate ZooKeeper quorum, so it must be possible to configure that ZooKeeper connection for TLS as well. Note that `kafka.security.auth.SimpleAclAuthorizer` was deprecated in AK 2.4 (in favor of `AclAuthorizer`) and will not support TLS connectivity to ZooKeeper.

CLI Tools

The list of CLI tools that used non-deprecated direct ZooKeeper access in the *previous* AK (AK 2.4) release was as follows:

1. `zookeeper-security-migration.sh` (`kafka.admin.ZSecurityMigrator`)
2. `kafka-reassign-partitions.{bat,sh}` (`kafka.admin.ReassignPartitionsCommand`)
3. `kafka-configs.{bat,sh}` (`kafka.admin.ConfigCommand`)
4. `zookeeper-shell.{bat,sh}`

It doesn't make sense to address direct ZK access in #1 since connecting to ZooKeeper and applying/removing ZooKeeper ACLs is the whole point of the tool. (In theory we could replace its direct ZK access in favor of a Kafka API, but that seems silly.)

Direct ZK access in #2 above is being addressed via the already-accepted KIP-455: Create an Administrative API for Replica Reassignment, and the direct access flag will be deprecated via KIP-555: Deprecate Direct Zookeeper access in Kafka Administrative Tools. Therefore no changes are required here.

Direct ZK access in #3 has already been replaced via a `--bootstrap-server` flag and will be deprecated in the next release via KIP-555 as well. A comment in the code at https://github.com/apache/kafka/blob/trunk/core/src/main/scala/kafka/admin/ConfigCommand.scala#L65) indicates that connecting directly to ZooKeeper with this CLI tool is still a supported use case when configuration information needs to be bootstrapped into a ZooKeeper quorum prior to starting Kafka. This is a very special use case for sure, but it does mean that accessing ZooKeeper directly via this CLI tool will be required, and passing TLS configuration to it in a secured way will be necessary.

Direct ZooKeeper access is the whole point of #4, of course, but the ZooKeeper project does not yet provide functionality to pass secured TLS configs to the underlying class that we invoke from this shell script – org.apache.zookeper.ZooKeeperMain – as described in

⚠ Unable to render Jira issues macro, execution error.

. We will need to add it.

`kafka-acls.{bat,sh}` (`kafka.admin.AclCommand`) is an additional CLI tool not in the above list that supports bootstrapping information into ZooKeeper. Direct ZooKeeper access has been deprecated in the ACLs CLI tool for some time, but it is still required for this special use case, and passing TLS configuration in a secured way will be necessary.

Goals

- Harden/secure the configuration mechanism for Zookeeper TLS connectivity from:
  - Kafka Brokers (including from `kafka.security.authorizer.AclAuthorizer` if/when configured)
  - `zookeeper-security-migration.sh`
  - `kafka-configs.{bat,sh}` and `kafka-acls.{bat,sh}`
  - `zookeeper-shell.{bat,sh}`
- Support client certificate authentication to ZooKeeper both with and without SASL authentication in ZK Security Migrator and the broker (when `zookeeper.set.acl` is true).
- Add system tests to confirm the hardened/secured configuration for TLS connectivity to ZooKeeper
- Add explicit Kafka documentation on how to configure TLS connectivity to ZooKeeper – both mutual TLS and encryption-only
- Add a reference in the Kafka documentation to the ZooKeeper Quorum TLS configuration (https://zookeeper.apache.org/doc/r3.5.7/zookeeperAdmin.html#Communication+using+the+Netty+framework)

Out of Scope

- Zookeeper-to-Zookeeper Quorum TLS system tests and in-depth documentation (the ZooKeeper project already has such tests and documentation)
- Dynamic reconfiguration of ZooKeeper TLS configs

Public Interfaces

New Broker and AclAuthorizer Configurations

The below table contains the complete list of added configs. All configs being added are optional Strings with no default value unless otherwise noted. These values are potentially required to access ZooKeeper in the first place, so they are not dynamically reconfigurable (dynamic reconfiguration values are currently stored in ZooKeeper). Sensitive values (e.g. those of type **`Password`**) can be encrypted as described in KIP-421: Automatically resolve external configurations.

As an example, these are some of the configs that will be introduced:

```
zookeeper.ssl.client.enable=true
zookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty
zookeeper.ssl.keystore.location=kafka.jks
zookeeper.ssl.keystore.password=test1234
zookeeper.ssl.truststore.location=truststore.jks
zookeeper.ssl.truststore.password=test1234
```

Every config can be prefixed with "`authorizer.`" for the case when `kafka.security.authorizer.AclAuthorizer` requires a different TLS configuration when connecting to ZooKeeper. Each can config can be individually prefixed at will, and any prefixed config will override the corresponding Kafka config value.

Config Key	Documentation
`zookeeper.ssl.client.enable` **`Optional Boolean, default=false`**	Set client to use TLS when connecting to ZooKeeper. An explicit value overrides any value set via the <code>zookeeper.client.secure</code> system property (note the different name). Defaults to false if neither is set; when true, <code>zookeeper.clientCnxnSocket</code> must be set (typically to <code>org.apache.zookeeper.ClientCnxnSocketNetty</code>); other values to set may include <include list of all other properties below>
`zookeeper.clientCnxnSocket`	Typically set to <code>org.apache.zookeeper.ClientCnxnSocketNetty</code> when using TLS connectivity to ZooKeeper. Overrides any explicit value set via the same-named <code>zookeeper.clientCnxnSocket</code> system property.
`zookeeper.ssl.keystore.location`	Keystore location when using a client-side certificate with TLS connectivity to ZooKeeper. Overrides any explicit value set via the <code>zookeeper.ssl.keyStore.location</code> system property (note the camelCase).
`zookeeper.ssl.keystore.password` **`Optional Password`**	Keystore password when using a client-side certificate with TLS connectivity to ZooKeeper. Overrides any explicit value set via the <code>zookeeper.ssl.keyStore.password</code> system property (note the camelCase). Note that ZooKeeper does not support a <code>key</code> password different from the <code>keystore</code> password, so be sure to set the key password in the keystore to be identical to the keystore password; otherwise the connection attempt to Zookeeper will fail.
`zookeeper.ssl.keystore.type`	Keystore type when using a client-side certificate with TLS connectivity to ZooKeeper. Overrides any explicit value set via the <code>zookeeper.ssl.keyStore.type</code> system property (note the camelCase). The default value of <code>null</code> means the type will be auto-detected based on the filename extension of the keystore.
`zookeeper.ssl.truststore.location`	Truststore location when using TLS connectivity to ZooKeeper. Overrides any explicit value set via the <code>zookeeper.ssl.trustStore.location</code> system property (note the camelCase).
`zookeeper.ssl.truststore.password` **`Optional Password`**	Truststore password when using TLS connectivity to ZooKeeper. Overrides any explicit value set via the <code>zookeeper.ssl.trustStore.password</code> system property (note the camelCase).

`zookeeper.ssl.truststore.type`	Truststore type when using TLS connectivity to ZooKeeper. Overrides any explicit value set via the <code>zookeeper.ssl.trustStore.type</code> system property (note the camelCase). The default value of <code>null</code> means the type will be auto-detected based on the filename extension of the truststore.
`zookeeper.ssl.protocol` **Default=TLSv1.2**	Specifies the protocol to be used in ZooKeeper TLS negotiation. An explicit value overrides any value set via the same-named <code>zookeeper.ssl.protocol</code> system property.
`zookeeper.ssl.enabled.protocols`	Specifies the enabled protocol(s) in ZooKeeper TLS negotiation (csv). Overrides any explicit value set via the <code>zookeeper.ssl.enabledProtocols</code> system property (note the camelCase). The default value of <code>null</code> means the enabled protocol will be the value of the <code>zookeeper.ssl.protocol</code> configuration property.
`zookeeper.ssl.cipher.suites`	Specifies the enabled cipher suites to be used in ZooKeeper TLS negotiation (csv). Overrides any explicit value set via the <code>zookeeper.ssl.ciphersuites</code> system property (note the single word "ciphersuites"). The default value of <code>null</code> means the list of enabled cipher suites is determined by the Java runtime being used.
`zookeeper.ssl.endpoint.identification.algorithm` **Default=https**	Specifies whether to enable hostname verification in the ZooKeeper TLS negotiation process, with (case-insensitively) "https" meaning ZooKeeper hostname verification is enabled and an explicit blank value meaning it is disabled (disabling it is only recommended for testing purposes). An explicit value overrides any "true" or "false" value set via the <code>zookeeper.ssl.hostnameVerification</code> system property (note the different name and values; true implies https and false implies blank).
`zookeeper.ssl.crl.enable` **Optional Boolean, default=false**	Specifies whether to enable Certificate Revocation List in the ZooKeeper TLS protocols. Overrides any explicit value set via the <code>zookeeper.ssl.crl</code> system property (note the shorter name).
`zookeeper.ssl.ocsp.enable` **Optional Boolean, default=false**	Specifies whether to enable Online Certificate Status Protocol in the ZooKeeper TLS protocols. Overrides any explicit value set via the <code>zookeeper.ssl.ocsp</code> system property (note the shorter name).

As noted above, ZooKeeper does not support setting a key password within the keystore that differs from the keystore password itself: trying to do so will cause the ZooKeeper connection attempt to fail. Therefore there is no configuration value for the key password, and the key password must be identical to the keystore password.

ZooKeeper Security Migration CLI

The existing configuration option *java.security.auth.login.config*, used to set provide the JAAS configuration content will be kept as it is.

A new parameter will be added:

- *--zk-tls-config-file: <String: Zookeeper TLS configuration file path>*

This config file may contain any of the above properties to configure the TLS connection to ZooKeeper; any config(s) not mentioned in the above list will be ignored (a convenience to make it possible to pass a Broker properties file)

Config Command CLI

The same *--zk-tls-config-file* parameter will be added to support the bootstrap use case.

ACL Command CLI

The same *--zk-tls-config-file* parameter will be added to support the bootstrap use case.

ZooKeeper Shell CLI

A *-zk-tls-config-file* parameter will be added. Note the use of single-dash as opposed to double-dash here since all of the tool's parameters follow the ZooKeeper project's style and are specified via a single dash. The "usage" message will be updated accordingly.

Proposed Changes

The proposed changes include the public interface changes:

- New Kafka configurations, both non-prefixed as well as prefixed with "`authorizer.`"
- A new *--zk-tls-config-file* parameter for:
    - ZooKeeper Security Migration Tool
    - Config Command CLI (for the special use case of bootstrapping TLS-enabled ZooKeeper)
    - ACL Command CLIs (for the special use case of bootstrapping TLS-enabled ZooKeeper)
- A new *-zk-tls-config-file* parameter in the ZooKeeper Shell (again, note the single dash as opposed to the double-dash used above)

The proposed changes also include the addition of:

- System tests to confirm the hardened/secured configuration for TLS connectivity to ZooKeeper
- The use of ZooKeeper Security Migrator and Kafka Brokers with client certificate authentication both with and without SASL
- Explicit Kafka documentation on how to configure TLS connectivity to ZooKeeper – both mutual TLS and encryption-only

Compatibility, Deprecation, and Migration Plan

The changes are additions only, and there is no compatibility issue in the broker because the default for the broker config `zookeeper.client.secure` is `false`; TLS to ZooKeeper is an opt-in.

Test Plan

System tests will cover the following:

- Migrating Zookeeper/Kafka clusters from non-TLS-enabled ZooKeeper to TLS-enabled ZooKeeper
- Invoking the Zookeeper Security Migration tool against TLS-enabled ZooKeeper both with and without ZK SASL authentication enabled
- TLS encryption-only (i.e. no client certificate) connectivity to ZooKeeper.

Compatibility testing is unnecessary because Zookeeper TLS is not available in prior versions.

The connection between Kafka and Zookeeper is not on a critical path related to performance – brokers don't repeatedly communicate with Zookeeper as they process messages, for example – so introducing TLS encryption here does not require explicit performance testing.

Rejected Alternatives

ZooKeeper-Style Configuration Names

ZooKeeper uses camelCase configs that are inconsistent with Kafka broker configs:

ZooKeeper Config	Kafka Broker Config
zookeeper.ssl.keyStore.location	ssl.keystore.location
zookeeper.ssl.keyStore.password	ssl.keystore.password
zookeeper.ssl.keyStore.type	ssl.keystore.type
zookeeper.ssl.trustStore.location	ssl.truststore.location
zookeeper.ssl.trustStore.password	ssl.truststore.password
zookeeper.ssl.trustStore.type	ssl.truststore.type
zookeeper.ssl.ciphersuites	ssl.cipher.suites
zookeeper.ssl.enabledProtocols	ssl.enabled.protocols

It would be confusing and prone to mistake to have such a mismatch – especially for people who tend to know very little about ZooKeeper compared to Kafka.

The ZooKeeper config "`zookeeper.client.secure`" is also confusing – it only refers to whether the client should be configured to communicate with a TLS-encrypted ZK socket and has nothing to do with any other security-related client configuration that might be implied by the term "secure" (e.g. SASL, ACLs). We therefore use the broker-side config "`zookeeper.ssl.client.enable`" instead.

Inheriting Broker Configuration Values

As mentioned above, certain broker configuration related to TLS cannot be inherited because keystore and trustore information is dynamically reconfigurable and may end up being stored in ZooKeeper. There are other TLS configuration values that are not dynamically reconfigurable in the broker (protocols and cipher suites, for example), but selectively inheriting these configs provides little value and could simply introduce confusion as people might assume – incorrectly – that keystore and truststore information could also be inherited. We therefore inherit nothing from the broker related to TLS configuration.

Requiring All Authorizer Prefixed TLS Configs

We considered requiring all authorizer ZooKeeper TLS configuration values (prefixed with "`authorizer.`") to be specified if any of them were specified with a prefix. This seemed onerous and unnecessary, and it was inconsistent with the way other ZooKeeper configs for the authorizer worked: they are overlays/overrides, the Kafka config is inherited if they aren't specified via the prefix, and any/all can be prefixed or left out. So we treat the ZooKeeper TLS configs the same way.

zookeeper.ssl.hostname.verifiction.enable

We could opt to use a true/false config to enable/disable ZooKeeper hostame verification. The ZooKeeper system property `zookeeper.ssl.hostnameVerification` works that way (and cannot be changed). However, Kafka uses a different convention: it clears the endpoint identification algorithm from its default value of `https` to disable hostname verification. Since we are explicitly deviating from the ZooKeeper system properties everywhere else, and since this config is rarely used, we will stay consistent with the Kafka config here as well.

zookeeper.ssl.context.supplier.class

The "zookeeper.ssl.context.supplier.class" configuration doesn't actually exist in ZooKeeper 3.5.7. The ZooKeeper admin guide documents it as being there, but it doesn't appear in the code. This means we can't support it in this KIP. It has been added in the latest ZooKeeper 3.6 SNAPSHOT, so this conf ig could probably be added to Kafka via a new, small KIP if/when we upgrade to ZooKeeper 3.6 (which looks to be in release-candidate stage as of this writing). Unable to render Jira issues macro, execution error. ("Add zookeeper.ssl.context.supplier.class config if/when adopting ZooKeeper 3.6") exists for this issue.