

KIP-43: Kafka SASL enhancements


- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
 - [Configuration options](#)
 - [Protocol changes](#)
- [Proposed Changes](#)
 - [SASL configuration](#)
 - [SASL mechanism](#)
 - [Support for multiple mechanisms in a broker](#)
 - [SASL/PLAIN implementation](#)
 - [Testing](#)
- [Compatibility, Deprecation, and Migration Plan](#)
 - [Impact on existing clients](#)
 - [Rolling upgrade from 0.9.0.x](#)
 - [Rolling upgrade with change in SASL mechanism](#)
- [Rejected Alternatives](#)
 - [Enable a small set of SASL mechanisms with a default implementation in Kafka](#)
 - [Make Authenticator configurable](#)
 - [Support multiple SASL mechanisms within a Kafka broker with mechanism negotiation](#)
 - [Support multiple SASL mechanisms within a Kafka broker on different ports](#)

Status

Current state: *Accepted*

Discussion thread: [kafka-dev](#)

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

Kafka 0.9.0.0 added SASL-based authentication for clients and inter-broker communication. SASL is a framework that enables authentication and data security via replaceable mechanisms. But at the moment, SASL implementation in Kafka supports only SASL/GSSAPI using Kerberos and does not allow other SASL mechanisms to be plugged in. Enabling other SASL mechanisms will allow better integration with existing non-Kerberos authentication servers. In this KIP, we discuss a proposal for enabling other SASL mechanisms.

This KIP addresses the following extensions to the existing implementation:

1. Configurable SASL mechanism to enable integration with existing authentication servers
2. Support for additional standard SASL mechanisms which do not require custom callbacks or login interfaces to support token refresh
3. SASL/PLAIN implementation to enable simple username/password authentication without complex infrastructure
4. Support for multiple SASL mechanisms within the same server. This will be useful, for example, in organizations where internal and external users require different authentication mechanisms.

This KIP does not address the following extensions which be considered in a follow-on KIP:

1. Support for custom mechanisms
2. Configurable callback handlers to provide mechanism-specific input
3. Configurable login interface that manages the login process and the lifecycle of login context related resources to support custom mechanisms that require periodic ticket refresh

Public Interfaces

Configuration options

The following options will be added to `SaslConfigs.java` and can be configured as properties for Kafka clients and server:

1. `sasl.mechanism` (`String`) SASL mechanism used for client connections. This may be any mechanism for which a security provider is available in the JVM. Default value is GSSAPI.

2. `sasl.enabled.mechanisms` (`List<String>`) The list of SASL mechanisms enabled in the Kafka server. This may include any mechanism for which a security provider is available in the JVM. Default value is GSSAPI.
3. `sasl.mechanism.inter.broker.protocol` (`String`) SASL mechanism used for inter-broker connections. Default value is GSSAPI.

Protocol changes

A new Kafka `SaslHandshakeRequest` and response type will be defined to add a handshake request flow to enable clients to communicate their chosen SASL mechanism to the broker. The formats of the request and response are shown below:

SaslHandshakeRequest

```
SaslHandshakeRequest => Mechanism
Mechanism => string
```

Field	Description
Mechanism	The SASL mechanism chosen by the client for SASL authentication

SaslHandshakeResponse

```
SaslHandshakeResponse => ErrorCode EnabledMechanisms
ErrorCode => int16
EnabledMechanisms => [string]
```

Field	Description
ErrorCode	Error code set to NONE(0) if the request succeeds. UNSUPPORTED_SASL_MECHANISM is returned if the mechanism specified in the client request is not enabled in the broker. ILLEGAL_SASL_STATE indicates that the request was unexpected. Kafka Sasl handshake requests may be sent only once, prior to the actual SASL authentication flow.
EnabledMechanisms	Array of mechanisms enabled in the broker

Successful Kafka `SaslHandshakeRequest/Response` flow should be immediately followed by the actual SASL authentication packets using the selected SASL mechanism. SASL authentication exchange consists of opaque client and server tokens as defined by the SASL mechanism and are typically obtained using a standard SASL library. These packets are not prefixed with Kafka request/response headers. No further Kafka requests may be sent until SASL authentication exchange is completed. For interoperability with 0.9.0.x, the Kafka handshake request flow is omitted for inter-broker communication using SASL for connections to older brokers.

Proposed Changes

The changes will be implemented under `KAFKA-3149`. The implementation will include changes to support additional mechanisms and enable multiple mechanisms in the broker. A simple default implementation for SASL/PLAIN in Kafka that is currently in `KAFKA-2658` will also be included. This section gives a summary of the changes and the rationale behind them.

SASL configuration

SASL in Kafka is configured using the standard JAAS configuration. SASL configuration consists of:

1. `LoginContext` that specifies the login module class and properties for the login module, specified using JAAS configuration
2. SASL mechanism and other properties specific to the mechanism configured as Kafka client or server properties. Additional properties and selection policies handled by the SASL implementation may also be specified when the `SaslClient` or `SaslServer` is constructed.
3. Additional input required by the SASL implementation obtained using `CallbackHandlers`
4. `SaslServer` or `SaslClient` implementation for the configured mechanism. These are installed as security providers in the JVM. For custom mechanisms, providers can be initialized in the static initializer of the login module to ensure that the providers are installed before Kafka creates the `SaslServer/SaslClient`.

All the four types of configuration above need to be configured consistently for SASL authentication to operate correctly. 1) and 4) are JVM configuration options. 2) and 3) are currently hard-coded in Kafka. The proposed changes enable configuration for 2) to enable more SASL mechanisms to be supported in Kafka clients and servers. 3) will be addressed in a follow-on KIP to support any mechanism including custom mechanisms.

SASL mechanism

Client connections will use the property `sasl.mechanism` to specify the mechanism to be used for SASL authentication. Kafka servers may also specify the configuration option `sasl.enabled.mechanisms` to provide the list of enabled mechanisms when multiple mechanisms are enabled in the server. In

order to plug in any SASL mechanism including custom mechanisms, mechanism will be specified as `String` rather than an `enum` with a restricted set of values. GSSAPI will be used as the default mechanism for interoperability with 0.9.0.x. If `sasl.enabled.mechanisms` is not specified, only GSSAPI will be enabled in the server. If a list of values is specified, GSSAPI will be enabled only if included in the list, allowing servers to be run with SASL without complex Kerberos setup if required.

Clients may enable only one mechanism and the mechanism name is sent to the server before any SASL authentication packets are sent, if the mechanism is not GSSAPI. Server fails the authentication if the client mechanism is not enabled in the broker. For inter-broker communication, `sasl.mechanism.inter.broker.protocol` configuration on the broker is used by the client-mode connection to choose the SASL mechanism.

Support for multiple mechanisms in a broker

Some organizations may have a requirement to use different authentication mechanisms within the same broker. For instance, this may be useful if internal and external clients connecting to the same broker use different authentication servers. Negotiation of SASL mechanisms is not included in the SASL protocol. Typically, application protocols which support multiple mechanisms include a mechanism negotiation phase where the server advertises the list of enabled mechanisms and the client selects one of the available mechanisms and sends the selected mechanism to the server. The current Kafka authentication protocol for SASL does not perform any negotiation of mechanisms since only GSSAPI is supported.

To keep the Kafka protocol simple, only a single SASL mechanism is allowed for Kafka clients. Clients send the mechanism name to the server in a handshake request before any SASL authentication packets are sent. Kafka server checks the first packet and if it is one of the mechanism names that it recognizes including custom mechanism names, it switches to that mechanism and starts SASL authentication using subsequent packets. If not, the server defaults to GSSAPI mechanism, treating the first packet as the first GSSAPI authentication packet from the client. The Kafka Java client implementation skips handshake request flow for GSSAPI for inter-broker connections if `inter.broker.protocol.version` is 0.9.0.x, enabling rolling upgrade of 0.9.0.x clusters which use SASL for replication. Handshake requests are sent by other clients even for GSSAPI from 0.10.0.0 onwards.

With GSSAPI, the first context establishment packet starts with byte 0x60 (APPLICATION-0 tag) followed by a variable-length encoded size. The first Kafka handshake request packet containing client mechanism starts with a two-byte API key of 0x0011, making it easy to distinguish from a GSSAPI packet.

Client flow:

1. If `sasl.mechanism` is not GSSAPI, send a Kafka handshake request packet with the mechanism name to the server. Otherwise go to Step 3.
 - Request Format: | Kafka RequestHeader | Kafka SaslHandshakeRequest |
2. Wait for response from the server. If the error code in the response is non-zero, indicating failure, report the error and fail authentication.
3. Perform SASL authentication with the configured client mechanism. SASL authentication packets do not contain a Kafka RequestHeader.
 - Client token Format: | Size (int32) | SASL client authentication token |

Server flow:

1. Wait for first authentication packet from client
2. If this packet is a not valid Kafka handshake request, go to Step 4 and process this packet as the first GSSAPI client token
3. If the client mechanism in the Kafka handshake request received in Step 2 is enabled in the broker, send a response with error code zero and start authentication using the specified mechanism. Otherwise, send an error response including the list of enabled mechanisms and fail authentication.
 - Response Format: | Kafka ResponseHeader | Kafka SaslHandshakeResponse |
4. Perform SASL authentication with the selected mechanism. If mechanism exchange was skipped, process the initial packet that was received from the client first. SASL authentication packets are expected without a Kafka RequestHeader until SASL authentication exchange completes. SASL server authentication packets are sent back without a Kafka response header.
 - Server token Format: | Size (int32) | SASL server authentication token |

Step 3 in the client flow and Step 4 in the server flow correspond to the start of the actual SASL authentication exchange when authentication tokens are exchanged for the selected SASL mechanism. These are treated as opaque tokens and are processed by the SASL provider of the mechanism. No further Kafka requests may be sent until the authentication exchange completes.

SASL/PLAIN implementation

SASL/PLAIN is a simple username/password authentication mechanism that is typically used with TLS for encryption to implement secure authentication. Unlike Kerberos, PLAIN does not require complex authentication infrastructure. Adding a default implementation for PLAIN in Kafka enables a simpler authentication mechanism for organizations which do not already use Kerberos. SASL/PLAIN protocol and its uses are described in <https://tools.ietf.org/html/rfc4616>.

The PR in `KAFKA-2658` will be rebased on the extensible interface from this KIP for this implementation. For the default SASL/PLAIN implementation included in Kafka, the username specified as authentication ID will be used as the authorization ID and principal.

Testing

- Due to the complexity of setting up Kerberos, limited unit testing has been implemented for SASL in the *clients* project. Along with the implementation for SASL/PLAIN, comprehensive unit tests will be added for the existing SASL implementation as well as the new interfaces.
- End-to-end tests will be added in the *core* project along with the existing SASL/Kerberos tests for SASL/PLAIN and multi-mechanism configuration
- System tests will be added for SASL/PLAIN and for multi-mechanism support.

Compatibility, Deprecation, and Migration Plan

Impact on existing clients

Existing clients will continue to use GSSAPI as the SASL mechanism and will not be impacted by the changes. Since default callback handlers can be used for SASL mechanisms that are implemented in Kafka, no configuration changes are required.

Rolling upgrade from 0.9.0.x

Rolling upgrade with GSSAPI as the SASL mechanism can be performed using a standard rolling restart with `inter.broker.protocol.version` set to 0.9.0.x in the first sequence of the upgrade. By default, if `sasl.mechanism` property is not specified, GSSAPI will be used. Handshake requests are not sent for GSSAPI when `inter.broker.protocol.version` is 0.9.0.x. Once the cluster is upgraded, `inter.broker.protocol.version` can be set to 0.10.0, enabling handshake requests for all SASL connections. If the mechanism is to be changed, this rolling restart can be followed by the addition of the new mechanism as described below.

Rolling upgrade with change in SASL mechanism

SASL mechanism can be modified with rolling restart using the following sequence:

1. Add the new mechanism to the list of mechanisms enabled in the broker and add the configuration properties required for the new mechanism. Perform rolling restart of the brokers.
2. Restart all clients with new SASL mechanism
3. Disable the old mechanism in the broker if necessary

Rejected Alternatives

Enable a small set of SASL mechanisms with a default implementation in Kafka

Since the security requirements and infrastructure used in different organizations vary, default implementation of login modules and security providers are unlikely to be sufficient for all users. Unlike Kerberos, where most users are likely to use the Kerberos module provided in the JDK, other mechanisms are likely to be customized by users to enable integration with existing authentication providers. The proposed implementation along with a follow-on KIP to configure callbacks will remove the restriction that a SASL mechanism without a default implementation in Kafka cannot be used at all.

Make Authenticator configurable

This would provide additional flexibility, but would require users to implement more code. This would be more suitable if there is a requirement to implement authentication using protocols other than SASL.

Support multiple SASL mechanisms within a Kafka broker with mechanism negotiation

Typical application protocols which support multiple SASL mechanisms perform negotiation of mechanisms before commencing authentication exchange using the negotiated mechanism. This involves exchange of mechanisms followed by the chosen mechanism being sent from the client to the server. For simplicity, Kafka clients will be allowed to enable only one mechanism in the client configuration. Kafka broker chooses the mechanism based on the first packet from the client, keeping the wire protocol simple.

Support multiple SASL mechanisms within a Kafka broker on different ports

To avoid mechanism exchange altogether, each SASL mechanism could be defined on a different port. With the current endpoint definitions, this would require each combination of transport layer and SASL mechanism to be defined as a new security protocol. This makes it harder to introduce new mechanisms without changing Kafka code. To define custom SASL protocols, the current security protocol enumeration needs to be made extensible as well.