KIP-86: Configurable SASL callback handlers

Status

Current state: *"Accepted"*

Discussion thread: here

JIRA: KAFKA-4292

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

Motivation

Kafka currently supports SASL authentication using SASL/PLAIN mechanism and KIP-84 addresses the addition of SASL/SCRAM. Credential verification in SASL/PLAIN servers is currently based on hard-coded credentials in JAAS configuration similar to Digest-MD5 configuration in Zookeeper. This is useful as a sample, but not suitable for production use since clear passwords are stored on disk. KIP-84 added SCRAM mechanism with Zookeeper as the password store. In production installations where Zookeeper is not secure (e.g. Kafka on the cloud), an alternative password store may be required.

With the current Kafka implementation, the entire `SaslServer` implementation needs to be replaced to enable new credential providers. It will be useful to add an extension point for SASL that enables just the credential providers to be replaced with a custom implementation. This should be done in a consistent way for all SASL mechanisms.

This KIP proposes to enable customization of SASL server and clients using configurable callback handlers. Configurable callback handlers for SASL /PLAIN and SASL/SCRAM will enable credential providers to be replaced in a simple and consistent way. In addition to this, configurable callback handlers for both server and clients make it easier to configure new or custom SASL mechanisms that are not implemented in Kafka.

To enable custom SASL mechanisms to be plugged in easily, the login interface for clients and servers will also be made configurable. This will enable custom logic to be added for new SASL mechanisms. This is particularly useful for mechanisms like Kerberos that require periodic token refresh.

Public Interfaces

Configuration properties

Client callback handler class (for clients and brokers using SASL for inter-broker communication)

- Name: `sasl.client.callback.handler.class`
- Type: `CLASS`
- Doc: The fully qualified name of a Sasl client callback handler class that implements the `org.apache.kafka.common.security.auth.AuthenticateCallbackHandler` interface.
- Default: null (by default, the appropriate internal default callback handlers for the mechanism will be used)

Server callback handler classes (for brokers only)

- Name: `sasl.server.callback.handler.class`
- Type: `STRING`
- Doc: The fully qualified name of a SASL server callback handler class that implements the `AuthenticateCallbackHandler` interface. The config name must be prefixed by the listener prefix and mechanism name in lower case. For example, `listener.name.sasl_ssl.plain.sasl.server.callback.handler.class`=com.example.CustomPlainCallbackHandler.
- Default: null (by default, the appropriate internal default callback handlers for each mechanism will be used)

Login class (for clients and brokers)

- Name: `sasl.login.class`
- Type: `CLASS`
- Doc: A class that implements the `org.apache.kafka.common.security.auth.Login` interface. For brokers, the config name must be prefixed by the listener prefix and mechanism name in lower case. For example, `listener.name.sasl_ssl.plain.sasl.login.class`=com.example.PlainServerLogin for brokers and `sasl.login.class`=com.example.KerberosClientLogin for clients.
- Default: null (by default, the internal class `KerberosLogin` will be used if Kerberos is enabled on the listener and `DefaultLogin` otherwise)

Login callback handler class (for clients and brokers)

- Name: `sasl.login.callback.handler.class`
- Type: `CLASS`
- Doc: The fully qualified name of a Sasl login callback handler class that implements the `org.apache.kafka.common.security.auth.AuthenticateCallbackHandler` interface. For servers, the config name must be prefixed by the listener prefix and mechanism name in lower case. For example, `listener.name.sasl_ssl.plain.sasl.login.callback.handler.class`=com.example.PlainLoginCallbackHandler for brokers and `sasl.login.callback.handler.class`=com.example.PlainLoginCallbackHandler for clients.
- Default: null (by default, the internal class `AbstractLogin.DefaultLoginCallbackHandler` will be used).

Callback Handler

The callback handler interface `AuthenticateCallbackHandler` will extend the standard `javax.security.auth.callback.CallbackHandler` interface, enabling the handler to be passed directly to `SaslServer/SaslClient` implementations. The callback handler configured for a mechanism must include the callbacks as described below:

1. If using a `SaslServer/SaslClient` implementation from the JRE, the callbacks required for the mechanism are described in the Java SASL reference.
2. When using the `SaslServer/SaslClient` implementation included in Kafka (PLAIN or SCRAM), the callback defined below for the SASL mechanism must be handled.
3. Applications using custom implementations of `SaslServer/SaslClient` may define their own callbacks.

Callback handlers which require additional options at runtime (eg. URL of a credential server) may include arguments in the JAAS configuration using the config file or `sasl.jaas.config` property (KIP-85). This is similar to the way keytab location is configured for GSSAPI. Client callback handlers can retrieve `Subject` using `Subject.getSubject(AccessController.getContext())` to obtain credentials populated by the login module.

org.apache.kafka.common.security.auth.AuthenticateCallbackHandler

```
package org.apache.kafka.common.security.auth;

import java.util.List;
import java.util.Map;

import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.AppConfigurationEntry;

/*
 * Callback handler for SASL-based authentication
 */
public interface AuthenticateCallbackHandler extends CallbackHandler {

    /**
     * Configures this callback handler for the specified SASL mechanism.
     * @param configs Configuration options
     * @param saslMechanism Negotiated SASL mechanism
     * @param jaasConfigEntries JAAS configuration entries from the JAAS login context.
     *        This list contains a single entry for clients and may contain more than
     *        one entry for servers if multiple mechanisms are enabled on a listener.
     */
    void configure(Map<String, ?> configs, String saslMechanism, List<AppConfigurationEntry> jaasConfigEntries);

    /**
     * Closes this instance.
     */
    void close();
}
```

SASL/PLAIN Server Callbacks

SASL/PLAIN servers using the SaslServer implementation included in Kafka must handle `NameCallback` and `PlainAuthenticateCallback`. The username for authentication is provided in `NameCallback` similar to other mechanisms in the JRE (eg. Digest-MD5). The password provided by the client during SASL authentication is provided in `PlainAuthenticateCallback`. The callback handler sets authenticated flag in the callback after verifying username and password.

SASL/PLAIN server callback handler is requested to handle `NameCallback` and `PlainAuthenticateCallback` in that order. See the sample SASL/PLAIN callback handler for an example.

org.apache.kafka.common.security.plain.PlainCredentialCallback

```
package org.apache.kafka.common.security.plain;
import javax.security.auth.callback.Callback;

public class PlainAuthenticateCallback implements Callback {
    private final char[] password;
    private boolean authenticated;
    public PlainAuthenticateCallback(char[] password) {
        this.password = password;
    }
    public char[] password() {
        return password;
    }
    public boolean authenticated() {
        return this.authenticated;
    }
    public void authenticated(boolean authenticated) {
        this.authenticated = authenticated;
    }
}
```

SASL/SCRAM Server Callbacks

SASL/SCRAM servers using the `SaslServer` implementation included in Kafka must handle `NameCallback` and `ScramCredentialCallback`. The username for authentication is provided in `NameCallback` similar to other mechanisms in the JRE (eg. Digest-MD5). The callback handler must return SCRAM credential for the user if credentials are available for the username for the configured SCRAM mechanism.

SASL/SCRAM server callback handler is requested to handle `NameCallback` and `ScramCredentialCallback` in that order. See the sample SASL /SCRAM callback handler for an example.

org.apache.kafka.common.security.scram.ScramCredentialCallback

```
package org.apache.kafka.common.security.scram;
import javax.security.auth.callback.Callback;
public class ScramCredentialCallback implements Callback {
    private ScramCredential scramCredential;
    public ScramCredential scramCredential() {
        return scramCredential;
    }
    public void scramCredential(ScramCredential scramCredential) {
        this.scramCredential = scramCredential;
    }
}
```

org.apache.kafka.common.security.scram.ScramCredential

```
package org.apache.kafka.common.security.scram;
public class ScramCredential {
    private final byte[] salt;
    private final byte[] serverKey;
    private final byte[] storedKey;
    private final int iterations;
    public ScramCredential(byte[] salt, byte[] storedKey, byte[] serverKey, int iterations) {
        this.salt = salt;
        this.serverKey = serverKey;
        this.storedKey = storedKey;
        this.iterations = iterations;
    }
    public byte[] salt() {
        return salt;
    }
    public byte[] serverKey() {
        return serverKey;
    }
    public byte[] storedKey() {
        return storedKey;
    }
    public int iterations() {
        return iterations;
    }
}
```

SASL/PLAIN callbacks enable authentication of passwords using an external authentication server without requiring Kafka to have knowledge of actual passwords. The callback checks if the (user, password) combination provided during SASL/PLAIN exchange is valid. SASL/SCRAM callbacks do require the salted credentials to perform the SCRAM authentication and hence the credentials are requested in the callback.

Login Interface

The `LoginManager` implementation in Kafka uses implementations of the `Login` interface to create a login instance for authentication. This login class instantiates a `javax.security.auth.login.LoginContext` and invokes `LoginContext#login()` to authenticate a `Subject`. Two different login class implementations are used by Kafka, `KerberosLogin` if GSSAPI is enabled and `DefaultLogin` for all other mechanisms. This KIP proposes to make the Login class configurable so that new mechanisms that require custom logic similar to Kerberos may be added these easily.

org.apache.kafka.common.security.auth.Login

```
package org.apache.kafka.common.security.auth;

import java.util.Map;

import javax.security.auth.Subject;
import javax.security.auth.login.Configuration;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

/**
 * Login interface for authentication.
 */
public interface Login {

    /**
     * Configures this login instance.
     */
    void configure(Map<String, ?> configs, String contextName, Configuration configuration,
                   AuthenticateCallbackHandler loginCallbackHandler);

    /**
     * Performs login for each login module specified for the login context of this instance.
     */
    LoginContext login() throws LoginException;

    /**
     * Returns the authenticated subject of this login context.
     */
    Subject subject();

    /**
     * Returns the service name to be used for SASL.
     */
    String serviceName();

    /**
     * Closes this instance.
     */
    void close();
}
```

Proposed Changes

`ChannelBuilder` will create an instance of each configured callback handler using the default constructor. For mechanisms without a callback handler override, the existing default callback handlers (`SaslServerCallbackHandler/SaslClientCallbackHandler`) or mechanism-specific server handlers for PLAIN/SCRAM will be created. Callback handler instances will be created once for each enabled mechanism in `ChannelBuilder`, instead of per-connection. This enables callback handlers using external authentication servers to cache credentials or reuse connections if required. `SaslClientCallbackHandler` will be modified to obtain `Subject` using *`Subject.getSubject(AccessController.getContext())`* to avoid the current per-connection state.

`LoginManager` will be updated to use the configured login class if provided and use current classes `DefaultLogin` and `KerberosLogin` as default. LoginManager currently caches static JAAS configs using the login context name as key and dynamic JAAS configs using the full `sasl.jaas.config` value as key. The map will be updated to include the existing string (context name/`sasl.jaas.config`) combined with the login class, so that a `Login` instance is reused only if both the config as well as the `Login` class match.

Scenarios

Use an external authentication server for SASL/PLAIN authentication using the SaslServer implementation for PLAIN included in Kafka

Define a new class that implements `AuthenticateCallbackHandler` which handles `NameCallback` and `PlainAuthenticateCallback` and add the class to the broker's `sasl.server.callback.handler.class` property. A single instance of this callback handler will be created for the broker. The configured callback handler is responsible for validating the password provided by clients and this may use an external authentication server.

Use custom credential store instead of Zookeeper for storing SCRAM credentials

Set broker callback handler to a class that implements `AuthenticateCallbackHandler` which handles `NameCallback` and `ScramCredentialCallback`. SCRAM credentials from a custom store can be returned by the callback handler.

Use a custom SaslServer implementation for SCRAM

If a custom SaslServer implementation is used instead of the one included in Kafka, the custom implementation may require a different set of callbacks. A callback handler for these callbacks may be specified in `sasl.server.callback.handler.class`.

Configure a new mechanism not included in Kafka using custom SaslServer/SaslClient

A handler that handles any callbacks required for these server/client implementations may be specified in `sasl.server.callback.handler.class` and `sasl.client.callback.handler.class` for brokers and clients respectively.

Configure a new mechanism using the implementation provided by the JRE

Callbacks defined for the mechanism in the Java implementation must be handled by custom callback handlers if the behaviour differs from the default callbacks in Kafka.

Configure callbacks for different mechanisms on different listeners in the broker

KIP-103 introduced support for multiple listeners in the broker for the same security protocol. This allows brokers to configure different SASL mechanisms for internal and external traffic. The listener name prefix can be applied to `sasl.server.callback.handler.class` to define different callback handlers for each of the listeners.

Add a new SASL mechanism that requires periodic re-login (similar to Kerberos token refresh)

Configure a new `Login` class by setting the config `sasl.login.class`. The custom login class can periodically log out of the current `LoginContext`, instantiate a new `LoginContext` and login again using `LoginContext#login()`. An example of this is `KerberosLogin` in the current Kafka implementation.

Sample Callback Handler for SASL/PLAIN

Sample SASL/PLAIN Callback Handler

```
public class PlainServerCallbackHandler implements AuthenticateCallbackHandler {
    private List<AppConfigurationEntry> jaasConfigEntries;
    @Override
    public void configure(Map<String, ?> configs, String mechanism, List<AppConfigurationEntry>
jaasConfigEntries) {
        this.jaasConfigEntries = jaasConfigEntries;
    }
    @Override
    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
        String username = null;
        for (Callback callback: callbacks) {
            if (callback instanceof NameCallback)
                username = ((NameCallback) callback).getDefaultName();
            else if (callback instanceof PlainAuthenticateCallback) {
                PlainAuthenticateCallback plainCallback = (PlainAuthenticateCallback) callback;
                boolean authenticated = authenticate(username, plainCallback.password());
                plainCallback.authenticated(authenticated);
            } else
                throw new UnsupportedCallbackException(callback);
        }
    }
    protected boolean authenticate(String username, char[] password) throws IOException {
        if (username == null)
            return false;
        else {
            // Return true if password matches expected password
        }
    }
    @Override
    public void close() throws KafkaException {
    }
}
```

For custom SASL/PLAIN authentication, override `authenticate()` with custom implementation that verifies the given `password` for `username`.

Sample Callback Handler for SASL/SCRAM

Sample SASL/SCRAM Callback Handler

```
public class ScramServerCallbackHandler implements AuthenticateCallbackHandler {
    @Override
    public void configure(Map<String, ?> configs, String mechanism, List<AppConfigurationEntry>
jaasConfigEntries) {
    }
    @Override
    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
        String username = null;
        for (Callback callback : callbacks) {
            if (callback instanceof NameCallback)
                username = ((NameCallback) callback).getDefaultName();
            else if (callback instanceof ScramCredentialCallback)
                ((ScramCredentialCallback) callback).scramCredential(credential(username));
            else
                throw new UnsupportedCallbackException(callback);
        }
    }
    protected ScramCredential credential(String username) {
        // Return SCRAM credential from credential store
    }
    @Override
    public void close() {
    }
}
```

For custom credential store for SCRAM, override `credential()` with alternative method that obtains credential from the custom store.

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*

  None

- *If we are changing behavior how will we phase out the older behavior?*

  Existing behaviour will be retained as default

Test Plan

Existing integration and system tests will test the default behaviour. Additional unit and integration tests will be added to test the new configuration:

1. Test that PLAIN credential provider can be replaced
2. Test that SCRAM password store can be replaced
3. Test that new mechanisms not included in Kafka can be run with custom callback handlers and custom Login class.

Rejected Alternatives

Define a new credential provider interface instead of using CallbackHandler

The format of credentials required for each mechanism is different. For SASL/PLAIN, the proposed callback handler can be used with external authentication servers which validate passwords without allowing the broker to retrieve password for a user. For SASL/SCRAM, the hashed, salted credentials required for the mechanism are provided to the broker. Different credential providers can be defined to capture these differences, which may make the interface slightly simpler in these two cases compared to callback handlers. But the use of standard Java `CallbackHandler` interface is more flexible and future-proof since it is the interface used by `SaslServer/SaslClient` implementations to obtain application-specific data.