KIP-84: Support SASL SCRAM mechanisms

Status

Current state: *"Accepted"*

Discussion thread: *here*

JIRA: *KAFKA-3751*

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

Motivation

Kafka currently supports two SASL mechanisms out-of-the-box. SASL/GSSAPI enables authentication using Kerberos and SASL/PLAIN enables simple username-password authentication. Support for more mechanisms will provide Kafka users more choice and the option to use the same security infrastructure for different services. Salted Challenge Response Authentication Mechanism (SCRAM) is a family of SASL mechanisms that addresses the security concerns with traditional mechanisms like PLAIN and DIGEST-MD5. The mechanism is defined in RFC 5802 (https://tools.ietf.org/html/rfc5802).

This KIP proposes to add support for SCRAM SASL mechanisms to Kafka clients and brokers:

- `SCRAM-SHA-256`
- `SCRAM-SHA-512`

Public Interfaces

No public interface changes or new configuration options are required for this KIP.

Two new mechanisms may be provided for the client configuration `sasl.mechanism` and the broker configurations `sasl.enabled.mechanisms` and `sasl.mechanism.inter.broker.protocol`. The new mechanism names are `SCRAM-SHA-256`, and `SCRAM-SHA-512`. Any combination of SCRAM mechanisms can be enabled in the broker along with existing mechanisms if required.

Since support for SASL/SCRAM servers and clients is not available in Java, a new login module class will be added that loads and installs the SASL server and client implementations for SCRAM as Java security providers (similar to the existing SASL/PLAIN server support in Kafka). SCRAM is enabled by specifying one of the SCRAM mechanisms as the SASL mechanism (eg. `sasl.mechanism=SCRAM-SHA-256`) along with the new login module in the JAAS configuration. The login module and the underlying implementations can be overridden if required, for example, to integrate with existing authentication servers.

The implementation included in Kafka will store user credentials in Zookeeper as dynamically configurable properties. The credentials include a randomly generated salt, salted hash of the password (StoredKey and ServerKey), and the iteration count for each SCRAM mechanism that is enabled. These are stored as properties for each user under `/config/users/<user>`. These credentials are not sufficient to impersonate a client, but in installations where Zookeeper is not secure, an alternative secure SASL server implementation may be used to protect against a brute-force attack that may recover the password if a strong cryptographic hash function and high iteration count are not used. Zookeeper is a suitable store for short-lived credentials like delegation tokens.

Proposed Changes

ScramLoginModule

The static initializer of the SCRAM login module installs the SASL/SCRAM server and client implementations as security providers for the supported SASL /SCRAM mechanisms. The module obtains username and password for client connections from the JAAS configuration options “`username`” and “`passwor`” and these are set as the public and private credentials of the `Subject` respectively.

ScramSaslClientProvider/ScramSaslClient

`ScramSaslClient` implements the client-side SCRAM algorithm defined in RFC 5802.

Username and password are obtained from the `Subject's` public and private credentials using existing callback handlers. No other shared secrets are required.

ScramSaslServerProvider/ScramSaslServer

`ScramSaslServer` implements the server-side SCRAM algorithm defined in RFC 5802.

The implementation included in Kafka will obtain user credentials from Zookeeper. Dynamic config update handlers will be used to maintain a cache of valid credentials in the broker.

For production use, the login modules and server/client implementations can be replaced if required with an alternative implementation that stores credentials more securely.

JAAS configuration

The login context `KafkaClient` is used by clients and the context `KafkaServer` is used by brokers. Username/password specified in `KafkaClient` is used for client connections and the username/password in `KafkaServer` is used for inter-broker connections. Credentials supplied by the client are validated by the SASL server in the broker against the salted, hashed passwords stored in Zookeeper using the SCRAM algorithm.

JAAS configuration for Clients

```
KafkaClient {
        org.apache.kafka.common.security.scram.ScramLoginModule required
        username="alice"
        password="alice-secret";
};

KafkaServer {
        org.apache.kafka.common.security.scram.ScramLoginModule required
        username="admin"
        password="admin-secret";
}
```

Credential configuration in Zookeeper

User credentials are stored in Zookeeper as dynamically configurable properties in the path /config/users/<encoded-user>. User names will be URL-encoded using the same encoding scheme as KIP-55.

Sample configuration for user credentials

```
// SCRAM credentials for user alice: Zookeeper persistence path /config/users/alice
{
        "version":1,
        "config": {
          "SCRAM-SHA-512" : "salt=djR5dXdtZGNqamVpeml6NGhiZmMwY3hrbg==,
stored_key=sb5jkqStV9RwPVTGxG1ZJHxF89bqjsD1jT4S...==,server_key=...,iterations=4096",
          "SCRAM-SHA-256" : "salt=10ibs0z7xzlu6w5ns0n188sis5,stored_key=+Acl
/wi1vLZ95Uqj8rRHVcSp6qrdfQIwZbaZBwM0yvo=,server_key=nN+fZauE6vG0hmFAEj/49+2yk0803y67WSXMYkgh77k=,
iterations=4096"
        }
};
```

For each supported mechanism, a new property is added with the mechanism name. The value of the property is a comma-separated list of key-value pairs similar to SCRAM messages and has the following elements:

- salt=<Salt>

- stored_key=<StoredKey>
- server_key=<ServerKey>
- iterations=<Iterations>

Tools

`kafka-configs.sh` will be extended to support management of credentials in Zookeeper as dynamic properties of users. Two new properties will be supported for entity type `users`, one for each mechanism. For ease of use, the tool will take a password and an optional iteration count and generate a random salt, ServerKey and StoredKey as specified in in RFC 5802. For example:

```
bin/kafka-configs.sh --zookeeper localhost:2181 --alter --add-config 'SCRAM-SHA-256=[iterations=4096,
password=alice-secret],SCRAM-SHA-512=[password=alice-secret]' --entity-type users --entity-name alice
```

When the above config command is run, the tool generates a random salt for each requested SCRAM mechanism (SCRAM-SHA-256 and SCRAM-SHA-512 in the example). The tool then generates stored key and server key as described in SCRAM Algorithm Overview using the SCRAM message formatter implementation that is used to salt/hash during SCRAM exchanges.

```
SaltedPassword  := Hi(Normalize(password), salt, i)
ClientKey       := HMAC(SaltedPassword, "Client Key")
StoredKey       := H(ClientKey)
ServerKey       := HMAC(SaltedPassword, "Server Key")
```

Default iteration count will be 4096. The actual password "`alice-secret`" is not stored in Zookeeper and is not known to Zookeeper or Kafka brokers. The hashed properties stored in Zookeeper can be retrieved using the `--describe` option of kafka-configs.sh. The random salt, stored key, server key and iteration count are persisted in Zookeeper using the format described in Credential configuration in Zookeeper.

For example:

```
bin/kafka-configs.sh --zookeeper localhost:2181 --describe --entity-type users --entity-name alice
```

> *Configs for user-principal 'alice' are SCRAM-SHA-512=[salt=djR5dXdtZGNqamVpeml6NGhiZmMwY3hrbg==, stored_key=sb5jkqStV9RwPVTGxG1ZJHxF89bqjsD1jT4SFDK4An2goSnWpbNdY0nkq0fNV8xFcZqb7MVMJ1tyEgif5OXKDQ==, server_key=3EfuHB4LPOcjDH0O5AysSSPiLskQfM5K9+mOzGmkixasmWEGJWZv7svtgkP+acO2Q9ms9WQQ9EndAJCvKHmjjg==, iterations=4096],SCRAM-SHA-256=[salt=10ibs0z7xzlu6w5ns0n188sis5,stored_key=+Acl /wi1vLZ95Uqj8rRHVcSp6qrdfQIwZbaZBwM0yvo=,server_key=nN+fZauE6vG0hmFAEj/49+2yk0803y67WSXMYkgh77k=, iterations=4096]*

Credentials can be deleted using the --delete option. For example:

```
bin/kafka-configs.sh --zookeeper localhost:2181 --alter --delete-config 'SCRAM-SHA-256,SCRAM-SHA-512' --entity-
type users --entity-name alice
```

Extensions to support Delegation tokens

KIP-48 addresses support for delegation tokens in Kafka. SCRAM is a suitable mechanism for authentication using delegation tokens. KIP-48 proposes to persist credentials for delegation tokens in Zookeeper which includes the Kafka principal as the token owner. Clients authenticate using SCRAM-SHA-256, providing the delegation token HMAC as password.

SCRAM messages have an optional extensions field which is a comma-separated list of `key=value` pairs. When KIP-48 is implemented, an extension will be added to the first client SCRAM message to indicate that authentication is being requested for a delegation token. This will enable Kafka broker to obtain credentials and principal using a different code path for delegation tokens.

Security Considerations

As described in http://www.isode.com/whitepapers/scram.html, strong passwords and high iteration counts are important to guarantee security using SCRAM. Zookeeper is expected to be used as the credential store for SCRAM only in installations where Zookeeper is secure. A strong hash function like SHA-256, high iteration counts and strong passwords must be used for protection against brute force attacks if Zookeeper security is compromised. Even though SCRAM is safe against replay attacks, SCRAM is expected to be used with TLS-encryption to prevent interception of SCRAM exchanges. This protects against dictionary or brute force attacks and against impersonation if Zookeeper is compromised. For more details, refer to *Security Considerations* in RFC-5802.

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 mechanisms will continue to be supported. The new mechanisms can be enabled in the broker along with SASL/GSSAPI and SASL/PLAIN. Existing upgrade procedures for new SASL mechanisms (as currently described in the documentation) can be used to switch to SCRAM.

Test Plan

One integration test and a system test will be added to test the good path for SASL/SCRAM. A system test will also be added for the upgrade scenario to test rolling upgrade and multiple broker mechanisms that include SCRAM. Unit tests will be added for failure scenarios and to test all supported SCRAM mechanisms.

Rejected Alternatives

Bump up the version of *SaslHandshakeRequest* to indicate support for new mechanisms

It was suggested during KIP-43/KIP-35 discussions that SaslHandshakeRequest version could be updated when new mechanisms are added to enable client implementors to choose a newer mechanism when connecting to new versions of brokers. After discussions, it was decided that the increase in handshake request version doesn't add value since clients have to use handshake requests to determine if a mechanism has been enabled in the broker and handle the case where a mechanism has been disabled. On failure, clients can indicate which mechanisms are enabled in the broker. At the moment, there are no client implementations or requirements to handle fallback mechanisms and clients fail with an error when the client mechanism is not enabled in the broker.

Specify username, password as Kafka client properties instead of the JAAS configuration

JAAS configuration is the standard Java way of specifying security properties and since Kafka already relies on JAAS configuration for SASL, it makes sense to store the options in jaas.conf. This is also consistent with SASL/PLAIN implementation in Kafka and similar mechanisms in Zookeeper. However, JAAS configuration is not particularly flexible and hence providing credentials as properties may provide an interface that is simpler to use. But this should be looked at in the context of all SASL mechanisms rather than just SCRAM.

Make the credential provider in ScramSaslServer pluggable

Some Kafka users may want to replace Zookeeper-based credential store with an external secure store. It may be useful to make the credential provider in ScramSaslServer pluggable to enable this easily. Since it is possible to plug in new login modules and SaslServer implementations using standard Java security extension mechanisms, this KIP does not propose to make the credential provider a plugabble public interface. A generic solution to configure callback handlers for any mechanism is being addressed in KIP-86.

Support more SCRAM mechanisms

All the hash functions defined in http://www.iana.org/assignments/hash-function-text-names/hash-function-text-names.xhtml are available in Java and we could support SASL/SCRAM for all of these in Java. But some of the mechanisms like SHA-1 are known to be insecure. To start with, it was decided to support only SHA-256 and SHA-512 to reduce the effort for clients in other languages. Support for other mechanisms like SHA-224 and SHA-384 can be added easily in future if there is a requirement.