This is a guest blog post by Stephane Maarek. Stephane is an IT consultant at Simple Machines (http://simplemachines.com.au/). You can check his GitHub (https://github.com/simplesteph) or connect with him on LinkedIn (https://www.linkedin.com/in/stephanemaarek).

If you use Apache Kafka to integrate and decouple different data systems across multiple teams and departments, you probably have experienced the need to standardize data formats in order to guarantee data quality.

Confluent Open Source (https://www.confluent.io/download/) includes a schema registry (http://docs.confluent.io/current/schema-registry/docs/) as a way to manage Avro schemas (https://avro.apache.org/) within Kafka topics. The Confluent Schema Registry (https://www.confluent.io/confluent-schema-registry/) helps enforce schema compatibility, which allows you to evolve your schema over time without breaking your downstream consumers.

In this blog post, we will explain how to configure the Confluent Schema Registry to work with a secured Kafka cluster.

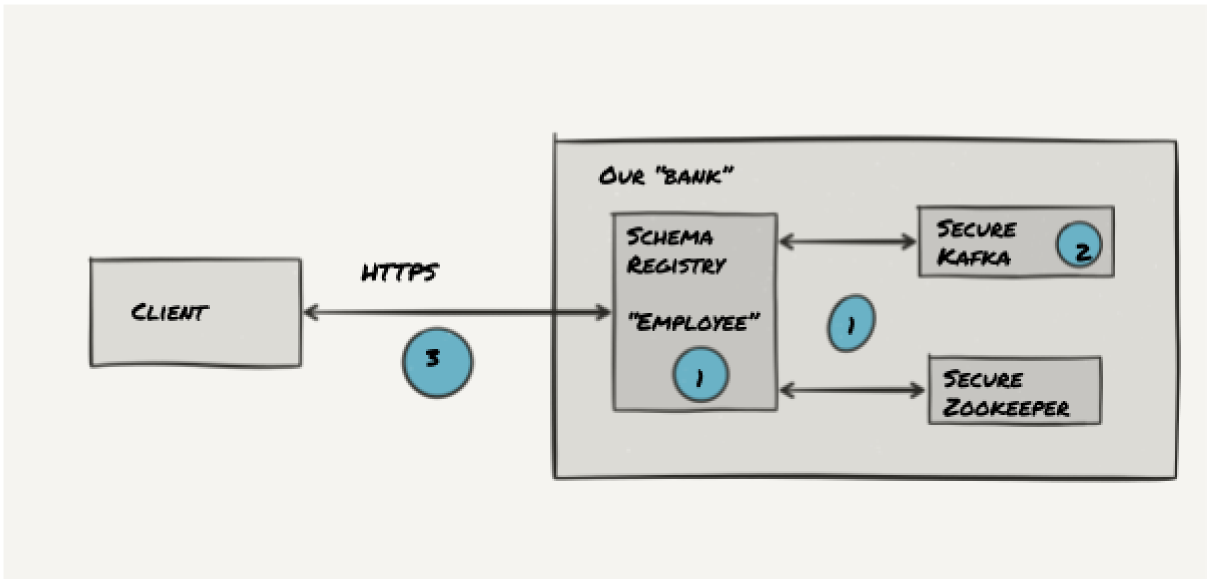
The main challenge for connecting the schema registry to a secured Kafka cluster is that highly secured Kafka clusters often only allow authenticated clients to connect to Kafka. This is done by closing the non-authenticated ports and it is especially important during audits – when you need to prove that only authorized users can access sensitive information.

In order to work in high-security Kafka environments, we need to configure the schema registry so it can communicate with Kafka using the authenticated SASL mechanism. We also need to make sure the schema registry has the privileges it needs to register schemas with Kafka and we need to secure the communication between the schema registry and its clients.

To help see how this works, let's use a bank as an analogy. Your bank needs to be secured, so that:

1. All your employees must have a badge to be identified before entering the building (SASL)
2. Once at their desk, all of your employees only have the exact set of permissions to the systems they need to perform their jobs (ACLs)
3. When your clients discuss important matters with your employees, they need to be certain that their discussions are private (HTTPS / SSL)

In our case the bank is Kafka, our employee is the schema registry, and your clients are… your Kafka clients!



This rest of this blog post describes the comprehensive setup of SASL/SSL, ACLs and HTTPS for the Confluent Schema Registry.

Pre-requisites

This blog post assumes you have already configured Apache Kafka security using SASL and SSL. If you are looking for details on exactly how to configure a secured Kafka cluster, you can read the documentation (https://kafka.apache.org/documentation/#security) and a tutorial blog post (https://www.confluent.io/blog/apache-kafka-security-authorization-authentication-encryption). In this post, we will focus on SASL implemented with Kerberos (GSSAPI), although the steps don't change much based on your authentication mechanism (Plaintext Module, or SCRAM Module).

We will also assume that your ZooKeeper client principal is the same for all your brokers and that the principal is myzkclient@EXAMPLE.COM.

3 Steps to Securing the Schema Registry

Now that you have a secured Kafka cluster *(our bank)*, you need to secure the schema registry (http://docs.confluent.io/current/schema-registry/docs/index.html (http://docs.confluent.io/current/schema-registry/docs/index.html)). Let’s analyze what the schema registry *(our bank employee)* needs to do:

- *It connects and gets securely identified to Kafka and Zookeeper (our employee gets identified by the bank guards)*
- It is only allowed to read and write to a topic named _schemas (our employee only has the set of permissions it needs to perform its job)
- It provides an HTTPS REST API to external clients *(our employee has secured and private discussions with their clients)*.

Connecting securely to Kafka and Zookeeper

In this first part, we make sure that the schema registry gets securely authenticated to Kafka and Zookeeper using SASL. The schema registry principal is schemaregistry@EXAMPLE.COM

As a reminder, the schema registry needs to connect to:

- Kafka in order to read and write to the topic _schemas
- Zookeeper in order to manipulate some Zookeeper nodes

Therefore, your JAAS file should look like this:

```
// Kafka Client authentication
KafkaClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/etc/kafka/keytabs/schemaregistry.keytab"
    principal="schemaregistry@EXAMPLE.COM";
};

// Zookeeper client authentication
Client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/etc/kafka/keytabs/myzkclient.keytab"
    principal="myzkclient@EXAMPLE.COM";
};
```

m/simplesteph/d744fba56091e08920a8da0f6f1470c6/raw/3094241d52a350e2599d05e2f404763c9906cf56/kafka_server_jaas.conf)
kafka_server_jaas.conf (https://gist.github.com/simplesteph/d744fba56091e08920a8da0f6f1470c6#file-kafka_server_jaas-conf)
hosted with ❤ by GitHub (https://github.com)

And your configuration should be like this:

```
# java properties file:
# SASL related properties
kafkastore.bootstrap.servers=SASL_SSL://kafka-1:9095/
kafkastore.sasl.kerberos.service.name=kafka
zookeeper.set.acl=true
# SSL related properties
kafkastore.ssl.truststore.location=/etc/kafka/secrets/kafkatruststore.jks
kafkastore.ssl.truststore.password=kafkatruststorepassword

# environment variable:
KAFKA_OPTS="-Djava.security.auth.login.config=/etc/kafka/kafka_client_jaas.conf"
```

raw
s://gist.github.com/simplesteph/fa1dc9ad532e198e2eef4307c2168147/raw/9a127d78d49db254ebaca7b32f26d928ae52ab3c/kafka-
r.properties)
kafka-server.properties (https://gist.github.com/simplesteph/fa1dc9ad532e198e2eef4307c2168147#file-kafka-server-properties)
hosted with ❤ by GitHub (https://github.com)

Important: Zookeeper.set.acl If that setting is set to true in Kafka, you have to set it to true in the schema registry. Otherwise, your schema registry won't be able to manipulate the ZooKeeper nodes. If zookeeper.set.acl is set to true, then you also need to make sure that your ZooKeeper Client principal is the same as your Kafka cluster (myzkclient@EXAMPLE.COM).

If your Kafka cluster has zookeeper.set.acl=false (which we don't recommend because then your cluster isn't really secured), then you're able to use zookeeper.set.acl=false in your schema registry and connect to ZooKeeper using any or no principal.

Setting the Right Kafka ACLs

As stated before, your schema registry will need to read and write to a Kafka topic named _schemas.

As your Kafka cluster is secured, by default no one will be able to read and write to such a topic. If your topic hasn't been created yet, you will need to create it manually:

```
kafka-topics --create --topic _schemas --zookeeper zoo1:2181 \
                 --config cleanup.policy=compact --partitions 1 --replication-factor 3
```

w
/gist.github.com/simplesteph/5c338be9f7d5a9dd7de7f39cf9a03142/raw/eec79c3b13563bbb98f4fb63269ed881ae77b30e/schema-
ate-topic.sh)
schema-reg-create-topic.sh (https://gist.github.com/simplesteph/5c338be9f7d5a9dd7de7f39cf9a03142#file-schema-reg-create-topic-sh) hosted with ❤ by GitHub (https://github.com)

Then you need to authorize the Schema Registry for a write:

```
kafka-acls --add --allow-principal User:schemaregistry \
               --producer --topic _schemas
```

'
gist.github.com/simplesteph/3f8651da47724d449b6e60205279c983/raw/9a57f2153ff02e48fe2074f8b01236237b2f9c6c/authorize-
producer.sh)
authorize-sch-reg-producer.sh (https://gist.github.com/simplesteph/3f8651da47724d449b6e60205279c983#file-authorize-sch-reg-producer-sh) hosted with ❤ by GitHub (https://github.com)

You also need to ensure that your schema registry user can read from that topic. The tricky part is that to read a topic, you need to be authorized alongside a group.id. On Confluent <=3.2.0, you have no control over the group id that the schema registry will pick, which will be built the following way:

- schema-registry-<hostname>-<firstportinlisteners>

In our blog example, the group.id would be:

- Hostname=kafka-schema-registry-1.kafka-schema-registry.example.com
- Listeners=https://0.0.0.0:443/
- group.id = schema-registry-kafka-schema-registry-1.kafka-schema-registry.example.com-443

Therefore, you need to make sure your hostnames and ports will remain constant over time, as having a stable group.id is key for Kafka security stability.

```
kafka-acls --add --allow-principal User:schemaregistry \
               --consumer --topic _schemas \
               --consumer-group schema-registry-kafka-schema-registry-1.kafka-schema-registry.example.com-
```

st.github.com/simplesteph/6f33de6cd2719c6bd3d20192570fcea4/raw/8ee6269bd624129a83be8c496fbf66cb0db766b2/authorize-
nsumer.sh)
authorize-sch-reg-consumer.sh (https://gist.github.com/simplesteph/6f33de6cd2719c6bd3d20192570fcea4#file-authorize-sch-reg-consumer-sh) hosted with ❤ by GitHub (https://github.com)

On Confluent Platform, on versions newer than 3.2.0 you will be able to overwrite the used group.id using the following config: kafkastore.group.id, thanks to this change (https://github.com/confluentinc/schema-registry/pull/491). This will give you full control over the group.id for the schema registry consumer. Please note that the same group.id can be shared by all of your schema registry instances as your consumer will never commit offsets. Therefore, your kafka-acls command will become:

```
kafka-acls --add --allow-principal User:schemaregistry \
           --consumer --topic _schemas --consumer-group schema-registry
```

st.github.com/simplesteph/2cd96b5c5833f95cdaaddff59c42fce9/raw/dc953ad0912ab04799bd3068503d862f7b63eb65/authorize-
›nsumer-simple.sh)
authorize-sch-reg-consumer-simple.sh (https://gist.github.com/simplesteph/2cd96b5c5833f95cdaaddff59c42fce9#file-authorize-sch-reg-consumer-simple-sh) hosted with ❤ by **GitHub (https://github.com)**

Enabling HTTPS for the REST API (SSL encryption):

Let's assume your schema registry is at *kafka-schema-registry-1.kafka-schema-registry.example.com*

You may want to get a certificate for that hostname, or better if you plan on having multiple kafka schema registry instances, get a wildcard certificate **.kafka-schema-registry.example.com*. Note that if your domain is private the certificate has to be self signed.

Generate a keystore and a truststore the same way you have done it when you configured SSL in kafka (https://kafka.apache.org/documentation/#security_ssl (https://kafka.apache.org/documentation/#security_ssl)).

Then enable the following properties in the schema registry configuration:

The following are optional only if you'd like to authenticate HTTPS clients

```
listeners=https://0.0.0.0:443/
ssl.keystore.location=/etc/kafka/secrets/yourkeystore.jks
ssl.keystore.password=yourkeystorepassword
ssl.key.password=yourkeypassword
# The following are optional only if you'd like to authenticate HTTPS clients
ssl.truststore.location=/etc/kafka/secrets/yourtruststore.jks
ssl.truststore.password=yourtruststorepassword
```

raw
://gist.github.com/simplesteph/dc03580342b1e1295cecbbcc55f8818a/raw/5f55db11a9821f10e6b740e6f18eef7b56dbc89a/schema-ry.properties)
schema-registry.properties (https://gist.github.com/simplesteph/dc03580342b1e1295cecbbcc55f8818a#file-schema-registry-properties) hosted with ❤ by GitHub (https://github.com)

Nonetheless, you should keep the CAroot public certificate so that the applications that talk to your REST API are able to trust your SSL certificate.

Run the following code to make sure everything is working as expected:

```
# double check that the certificates are loaded
openssl s_client -debug -connect kafka-schema-registry-1.kafka-schema-registry.example.com:443 -tls1
# to verify if the schema registry is responding to REST requests
curl -k https://kafka-schema-registry-1.kafka-schema-registry.example.com:443/subjects
# to verify if your public certificate matches the SSL certificate from the schema registry
curl --cacert cacert.pem https://kafka-schema-registry-1.kafka-schema-registry.example.com:443/subject
```

st.github.com/simplesteph/850df24dc6c4b845674ede51bddbd9d5/raw/48a373b5d173d49e566ecbd4ea1a08768376bb45/schema-st-ssl.sh)
schema-registry-test-ssl.sh (https://gist.github.com/simplesteph/850df24dc6c4b845674ede51bddbd9d5#file-schema-registry-test-ssl-sh) hosted with ❤ by GitHub (https://github.com)

At this point all the REST API calls will be encrypted.

Conclusion

In this post, we've illustrated how to configure the schema registry when your Kafka cluster is fully secured using the SASL protocol, as well as enabling HTTPS so that communications over the web are fully encrypted. This should also help you integrate security with any application that needs to talk to ZooKeeper and/or Kafka, as the configurations are similar. Kafka Security allows tight control over what the connected users can and cannot do, and in this case the assurance that your schema registry will be able to read and write to its topic _schemas. Your new employee (Confluent Schema Registry) is now fully secured!

You can install the Confluent Open Source (https://www.confluent.io/download/) packages and experiment with Kafka, schema registry and additional useful (and secure!) clients – all on your own development machine.