Mutual TLS (mTLS) Authentication

Use TLS certificates to authenticate users

# Objectives

We have learned to use API Keys and OIDC to authenticate users. A third way, mutual TLS (mTLS), uses TLS certificates. We will learn how to create a simple certificate authority (CA) for self-signing, upload this CA to your Confluent Organisation, configure identity pools, and create certificates for your clients.

# Labs

## Creating the Certificate Authority (CA)

In a real-world scenario, the Certificate Authority is provided by the It or security administrators. For testing and learning purposes, we will create our own certificate authority to sign and verify the client certificates.

* Use openssl to create a certificate authority and key

| openssl req -new -x509 -keyout ca-key.pem -out ca-cert.pem -days 1825 -subj '/CN=ccloud.bootcamp.confluent.io/OU=BOOTCAMP/O=CONFLUENT/L=PaloAlto/ST=Ca/C=US' -passin pass:confluent -passout pass:confluent |
| --- |

This will create two files: the key and the certificate.

You can adjust the subject to your liking but keep the format.

## Uploading the Certificate Authority (CA)

There are multiple ways to upload the CA. You can use the **confluent** CLIapplication to do this, but we will use the Confluent Cloud console.

* Log into your Confluent Cloud account and navigate to “Accounts & Access” via the Hamburger menu.
* Choose “Workload Identities” and add a new identity provider.
* Choose Certificate Authority
* Give the provider a new name, such as “Bootcamp CA”, and upload your new ca-cert.pem file.
* Press next and then validate and save

You now have a new Identity Provider in place.

## Creating the Identity Pool

Next, you need to create a few identity pools to link a certificate to a set of permissions (RBAC).

* Click on your Identity Provider
* Click on Add Pool
* Provide your pool with a name, for example, “Producer”
* Fill in the description if desired
* Leave the identifier at CN
* Set the filter to   
  CN == “producer”
* Save
* On the access page, click on your dedicated cluster and navigate to Topics
* Add a role assignment
  + Choose prefix rule
  + Create the topic name “test”
  + Choose DeveloperWrite
* Save

Repeat the exercise by adding a new identity pool called “Consumer” with the filter CN==”consumer”. On the access page, create a role assignment for the same prefix rule for test\*, but this time for DeveloperRead.

This time, we will add additional permission for all Consumer Groups. Set the role to DeveloperRead.

Finally, we are adding a third pool, this time for all clients. Call the pool “Clients”. Change the identifier to DN(Up to 255 char), and set the filter  
 DN.contains("OU=CLIENT")  
  
Set the access permissions to ResourceOwner for all topics starting with “client”.

## Creating and signing the client certificates

Now, we need to create certificate requests and sign these with our CA.

| openssl req -new -newkey rsa:2048 -keyout producer-key.pem -out producer-cert.csr -subj '/CN=producer/OU=CLIENT/OU=BOOTCAMP/O=CONFLUENT/L=PaloAlto/ST=Ca/C=US' -passin pass:changeme -passout pass:changeme  openssl req -new -newkey rsa:2048 -keyout consumer-key.pem -out consumer-cert.csr -subj '/CN=consumer/OU=CLIENT/OU=BOOTCAMP/O=CONFLUENT/L=PaloAlto/ST=Ca/C=US' -passin pass:changeme -passout pass:changeme |
| --- |

Then, we need to sign these new certificates with our CA:

| openssl x509 -req -days 365 -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -CAserial serial -in producer-cert.csr -out producer-cert.pem -passin pass:confluent  openssl x509 -req -days 365 -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -CAserial serial -in consumer-cert.csr -out consumer-cert.pem -passin pass:confluent |
| --- |

You can see that this is a signed certificate with the command

| openssl x509 -in consumer-cert.pem -noout -text |
| --- |

You can verify the certificate chain is intact with this command

| openssl verify -CAfile ca-cert.pem producer-cert.pem |
| --- |

## Convert the client certificates to JKS format

It is easiest to convert the PEM files to the Java JKS format for Java clients. You must first convert the certificate and private key into a PKCS#12 archive and then the JKS archive. Here are the steps:

| openssl pkcs12 -export -in producer-cert.pem -inkey producer-key.pem -out producer.p12 -name "Producer Certificate" |
| --- |

| keytool -importkeystore -srckeystore producer.p12 -srcstoretype pkcs12 -destkeystore producer.jks |
| --- |

You will be asked for the key password and the store password (twice). Use the password “changeme” for simplicity in this lab.

* Do this for your consumer certificate as well.

## Using the client certificates for authentication

We can finally try out our certificates. Create a configuration file like this, maybe named “producer.properties”:

| bootstrap.servers=<your Confluent Cloud cluster>:9092  security.protocol=SSL  ssl.keystore.location=producer.jks  ssl.keystore.password=changeme  ssl.key.password=changeme |
| --- |

Remember that you cannot access your cluster from your local machine. You must upload your JKS certificates to your jumphost using your preferred method (for example, scp or sftp).

* Continue your tests from your jumphost

The easiest command to run to see if you have access is probably “kafka-topics”.

| kafka-topics --list --bootstrap-server <your Confluent Cloud cluster>:9092 --command-config producer.properties |
| --- |

Since we have not created any topics yet, this will return nothing, but it should not flag any errors.

* Create a topic with this command

| kafka-topics --create --topic client-topic --partitions 6 --bootstrap-server <your Confluent Cloud cluster>:9092 --command-config producer.properties |
| --- |

This command should succeed since we gave all users with OU=CLIENT in the certificate name the role ResourceOwner for all topics starting with the word “client”.

We also need a “test-topic,” for which we need to use the **confluent** CLI or the Web console after enabling the Proxy. By now, you should know how to create a topic using these methods.

* Create a topic “test-topic”

Now that we have two topics, we can produce and consume messages.

| kafka-console-producer --topic test-topic --bootstrap-server <your Confluent Cloud cluster>:9092 --producer.config producer.properties |
| --- |

This command should return with a prompt (>) awaiting your input. Type in a few strings as your message, each followed by a RETURN. Type in CONTROL-D and return to finish.

To consume these messages, use the console consumer with our consumer user:

| kafka-console-consumer --topic test-topic --bootstrap-server <your Confluent Cloud cluster>:9092 --consumer.config consumer.properties --from-beginning |
| --- |

Note that this command does not return, so you need to cancel it with CONTROL-C.

Experiment with these commands, for example, by attempting to write with a consumer and note the errors you receive.

# References

<https://docs.confluent.io/cloud/current/security/authenticate/workload-identities/identity-providers/mtls/configure.html>

<https://docs.confluent.io/cloud/current/security/authenticate/workload-identities/identity-providers/mtls/cel-filters.html>

# Expected Outcomes

Set up mTLS for your dedicated clusters as an alternative to API Keys and OIDC.

Configure multiple users via identity pools.

Successfully produce and consume messages.

# Check your understanding

This colour marks advanced questions.

* How does a certificate differ from an API Key?
  + Are there any similarities?
* Which advantages do certificates offer over API Keys?
* Which disadvantages might you have?

# 