Introduction to Apache Kafka Security 🔒

In this blog, I will try my best to explain Kafka Security in terms everyone can understand. We will go over SSL, SASL and ACL.

. . .

Apache Kafka and the need for security

Apache Kafka is an internal middle layer enabling your back-end systems to share real-time data feeds with each other through Kafka topics. **With a standard Kafka setup, any user or application can write any messages to any topic, as well as read data from any topics**. As your company moves towards a shared tenancy model where multiple teams and applications use the same Kafka Cluster, or your Kafka Cluster starts on boarding some critical and confidential information, you need to implement security.

Problems Security is solving

Kafka Security has three components:

- **Encryption of data in-flight using SSL / TLS:** This allows your data to be encrypted between your producers and Kafka and your consumers and Kafka. This is a very common pattern everyone has used when going on the web. That's the "S" of HTTPS (that beautiful green lock you see everywhere on the web).

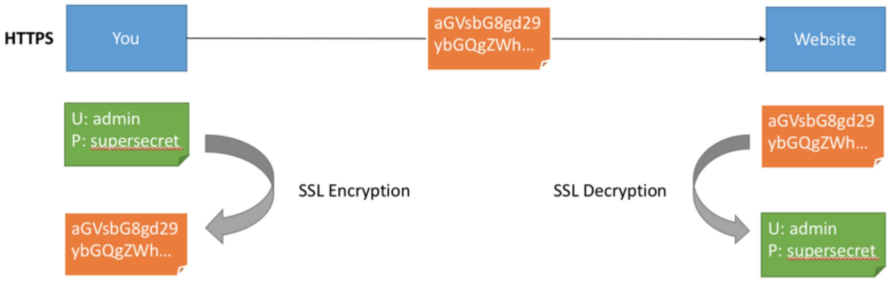
- **Authentication using SSL or SASL:** This allows your producers and your consumers to authenticate to your Kafka cluster, which verifies their identity. It's also a secure way to enable your clients to endorse an identity. Why would you want that? Well, for authorization!
- **Authorization using ACLs:** Once your clients are authenticated, your Kafka brokers can run them against access control lists (ACL) to determine whether or not a particular client would be authorised to write or read to some topic.

All these concepts are carefully taught and practiced in my Udemy Course on Kafka Security, but in this blog we'll get a good introduction to how security work.

. . .

Encryption (SSL)

Encryption solves the problem of the man in the middle (MITM) attack. That's because your packets, while being routed to your Kafka cluster, travel your network and hop from machines to machines. If your data is PLAINTEXT (by default in Kafka), any of these routers could read the content of the data you're sending:



SSL encryption for dummies

Now with Encryption enabled and carefully setup SSL certificates, your data is now encrypted and securely transmitted over the network. With SSL, only the first and the final machine possess the ability to decrypt the packet being sent.

This encryption comes at a cost: CPU is now leveraged for both the Kafka Clients and the Kafka Brokers in order to encrypt and decrypt packets. SSL Security comes at the cost of performance, but it’s low to negligible. Using Java 9 vs Java 8 with Kafka 1.0 or greater, the performance cost is decreased further by a substantial amount!

kafka-producer-perf-test on laptop with TLS achieves 2.5x throughput with Java 9 (AES_256_GCM_SHA384) vs Java 8 (AES_128_CBC_SHA256) (1/2)

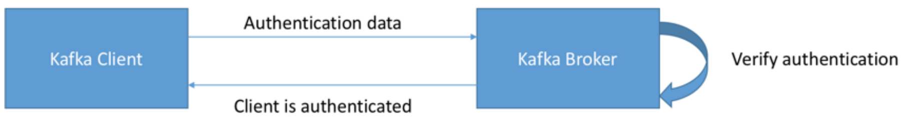
—@ijuma

Performance Improvements when using SSL + Java 9

Please note the encryption is only in-flight and the data still sits un-encrypted on your broker's disk.

In my Kafka Security Course, we go over an SSL Setup to demonstrate how encryption works

Authentication (SSL & SASL)



Authentication for Dummies

There are two ways to authenticate your Kafka clients to your brokers: SSL and SASL. Let's go over both

SSL Authentication

SSL Auth is basically leveraging a capability from SSL called two ways authentication. The idea is to also issue certificates to your clients, signed by a certificate authority, which will allow your Kafka brokers to verify the identity of the clients.

This is the most common setup I've seen when you are leveraging a managed Kafka clusters from a provider such as Heroku, Confluent Cloud or CloudKarafka.

SASL Authentication

SASL stands for Simple Authorization Service Layer and trust me, the name is deceptive, things are not simple. Basically, the idea is that the authentication mechanism is separated from the Kafka protocol (which is a nice idea). It's very popular with Big Data systems and most likely your Hadoop setup already leverages that.

SASL takes many shapes and forms and the following are supported by Kafka:

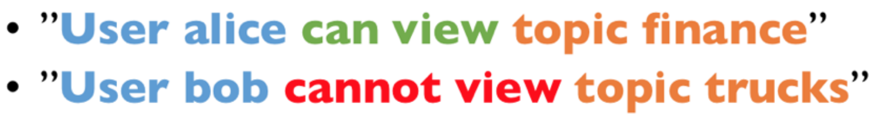
- **SASL PLAINTEXT:** This is a classic username/password combination. These usernames and passwords have to be stored on the Kafka brokers in advance and each change needs to trigger a rolling restart. It's very annoying and not the recommended kind of security. If you use SASL/PLAINTEXT, make sure to enable SSL encryption so that credentials aren't sent as PLAINTEXT on the network
- **SASL SCRAM:** This is a username/password combination alongside a challenge (salt), which makes it more secure. On top of this, username and password hashes are stored in Zookeeper, which allows you to scale

security without rebooting brokers. If you use SASL/SCRAM, make sure to enable SSL encryption so that credentials aren't sent as PLAINTEXT on the network

- **SASL GSSAPI (Kerberos):** This is based on Kerberos ticket mechanism, a very secure way of providing authentication. Microsoft Active Directory is the most common implementation of Kerberos. SASL/GSSAPI is a great choice for big enterprises as it allows the companies to manage security from within their Kerberos Server. Additionally, communications are encrypted to SSL encryption is optional with SASL/GSSAPI. Needless to say, setting up Kafka with Kerberos is the most difficult option, but worth it in the end.
- **(WIP) SASL Extension** (KIP-86 in progress)**:** To make it easier to configure new or custom SASL mechanisms that are not implemented in Kafka (have a read through the KIP)
- **(WIP) SASL OAUTHBEARER** (KIP-255 in progress): This will allow you to leverage OAUTH2 token for authentication (have a read through the KIP)

To make things short and simple, I would encourage SASL/SCRAM or SASL/GSSAPI (Kerberos) to be used today for your authentication layer.

Authorization (ACL)



- ”User alice can view topic finance”
- ”User bob cannot view topic trucks”

ACL for Dummies

Once your Kafka clients are authenticated, Kafka needs to be able to decide what they can and cannot do. This is where Authorization comes in, controlled by Access Control Lists (ACL). **ACL are what you expect them to be: User A can(‘t) do Operation B on Resource C from Host D**. Please note that currently with the packaged `SimpleAclAuthorizer` coming with Kafka, ACL are not implemented to have Groups rules or Regex-based rules. Therefore, each security rule has to be written in full (with the exception of the * wildcard).

ACL are great because they can help you prevent disasters. For example, you may have a topic that needs to be writeable from only a subset of clients or hosts. You want to prevent your average user from writing anything to these topics, hence preventing any data corruption or deserialization errors. ACLs are also great if you have some sensitive data and you need to prove to regulators that only certain applications or users can access that data.

To add ACLs, you can use the `kafka-acls` command (documentation here). It also even has some facilities and shortcuts to add producers or consumers.

```
kafka-acl --topic test --producer --authorizer-properties

zookeeper.connect=localhost:2181 --add --allow-principal
```

```
User:alice
```

The result being:

```
Adding ACLs for resource `Topic:test`:
  User:alice has Allow permission for operations: Describe from
hosts: *
 User:alice has Allow permission for operations: Write from hosts:
*

Adding ACLs for resource `Cluster:kafka-cluster`:
  User:alice has Allow permission for operations: Create from
hosts: *
```

Please note that using the default provided `SimpleAclAuthorizer` , your ACL are stored in Zookeeper. Therefore, it is important to secure Zookeeper and make sure only your Kafka brokers are allowed to write to Zookeeper (`zookeeper.set.acl=true`). Otherwise any user could come in and edit ACLs, hence defeating the point of security.

Finally, you may find the `kafka-acls` command hard to use in the long run. **For this, I have created a small utility called the Kafka Security Manager**: https://github.com/simplesteph/kafka-security-manager . This long running application (Docker image provided) allows you to source your ACL from an external source of truth and synchronize them continuously with Zookeeper, hence keeping your Kafka even more secure and making your audit team happy.

. . .

Next Steps

Now that you're interested in learning about security, or even setting it up for your cluster, you're going to have to go hands deep in it. This is going to be a fun and frustrating experience. To help, couple of resources:

- Kafka Documentation on Security: it is comprehensive but will require many reads and tries to make your setup work. Nonetheless, try to read it
- Confluent Documentation on Security: it is complete and comes with some hands-on to secure most Kafka components (including Kafka Connect, Kafka Schema Registry, etc…)
- The Kafka Security course: we created this course so that anyone can learn and get started with Security in a fully explained and hands-on way. It goes through the setup of SSL encryption, SSL authentication, SASL/Kerberos authentication, and Kafka ACL. We truly hope this will be a great launching pad for you.

Happy learning!♡

Liked it? Clap 👏, Share, Comment!