KIP-111: Kafka should preserve the Principal generated by the PrincipalBuilder while processing the request received on socket channel, on the broker.

- Status
- Motivation
- New or Changed Public Interfaces
- Proposed Changes
- Compatibility, Deprecation, and Migration Plan
- Test Plan
- Rejected Alternatives

Status

Current state: Closed (Covered by KIP-189)

Discussion thread: *(Original Archive) (Markmail)*

JIRA: *KAFKA-4454*

Released: <Kafka Version>

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

Motivation

Kafka allows users to plugin a custom `PrincipalBuilder` and a custom `Authorizer` by specifying the classpath of the corresponding classes in the config.

When a Kafka broker receives request bytes from clients over the socket channel, it reconstructs the request which is then handed over to the request handler threads (a.k.a API threads) to process the request. In the process of doing this, the Kafka broker constructs a `Session` object that holds a `KafkaPrincipal` and client's socket IP address. The `KafkaPrincipal` includes the type of the client principal (**"User"** as of now) and the name of the `Principal`, generated by the `PrincipalBuilder`. The `Authorizer` interface includes an *authorize(...)* method, that is invoked on every request that is received by the Kafka broker. If the broker has a custom `Authorizer` configured, it will delegate the *authorize(...)* call to the custom implementation. The *authorize(...)* method takes in the `Session` object, `Operation` requested and the `Resource` on which the operation is requested, as method parameters and returns true or false depending on the configured ACLs as follows :

```
def authorize(session: Session, operation: Operation, resource: Resource): Boolean
```

However, the principals generated by the plugged in `PrincipalBuilder` may contain additional custom fields, and the user's `Authorizer` implementation may need to access those fields in order to enforce ACLs correctly for those principals. Unfortunately, Kafka currently only extracts the name of the `Principal` when constructing the `Session` object as shown below, and loses the additional information at runtime:

```
val session = RequestChannel.Session(new KafkaPrincipal(KafkaPrincipal.USER_TYPE, channel.principal.getName),
channel.socketAddress) // (custom fields in principal if any are not passed through)
```

It is important to note that Java's `Principal` API is opaque and different Kafka service providers can have custom implementations of the `Principal` interface with additional features as per their requirements. Since Kafka allows users to plug in a custom `PrincipalBuilder` and a custom `Authorizer`, it does not make sense to extract only the name of the `Principal` and ignore the other fields in the generated `Principal` (which may be required by the custom `Authorizer`).

This issue can be addressed if Kafka preserves the original `Principal` object when it processes the incoming request, before handing it over to the API threads. The `Authorizer` will then be able to access this `Principal` object and use it to verify the ACLs.

New or Changed Public Interfaces

This KIP introduces a change to `Session` class to accept a parameter of `Java Principal` type instead of `KafkaPrincipal` type.

This change will not affect the default ACL `Authorizer` (`SimpleAclAuthorizer`) as we would generate a `KafkaPrincipal` from the `Java Principal` in the default `Authorizer`.

Proposed Changes

- Change the `Session` class to accept a parameter of type `Java Principal` instead of `KafkaPrincipal`.

```
case class Session(principal: Principal, clientAddress: InetAddress)
```

- The `Authorizer` can access this principal object as follows :

```
public boolean authorize(RequestChannel.Session session, Operation operation, Resource resource) {
...
 Principal principal = session.principal();
 User_Defined_Principal principal = (User_Defined_Principal) principal;
...
}
```

- `User_Defined_Principal` is the `Principal` generated by the `PrincipalBuilder` and it implements `Java Principal.`

Compatibility, Deprecation, and Migration Plan

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

There is no compatibility impact as there is no change in behavior.

Test Plan

- Unit tests to validate that new changes work as expected without affecting the existing behavior.

Rejected Alternatives

Alternative 1 :

- Kafka-acls.sh will allow to specify a custom `PrincipalBuilder` using a new command line parameter "-- principalBuilder" and `PrincipalBuilder` configs using a new command line parameter "--principalBuilder-properties". Users can use these to build their custom `Principal` (that implements Java `Principal`). Add a new API to `PrincipalBuilder` Interface :

```
public interface PrincipalBuilder extends Configurable {
...
 /**
 * Build a Principal using name.
 *
 * @param name Principal name
 * @return Principal
 */
 Principal buildPrincipal(String name);

...
}
```

- This `PrincipalBuilder` API will then be used to generate a `Principal` using the names specified in --allow-principal and --deny-principal parameters. This `Principal` can be included in `KafkaPrincipal` using the new constructor specified above.
- This alternative was rejected due to following reasons :
    1. Since the `Principal` is built using the "--principalBuilder-properties", users can only specify a particular type of `Principal`(s) (using --allow-principal / --deny-principal) at a time.
    2. If users want to specify multiple types of Principals, they will have to run the kafka-acls.sh multiple times with different "--principalBuilder-properties", even if the Principals might have the same name. For example, we can have a service `Principal` with name "XYZ" and a user `Principal` with name "XYZ".
- Due to above reasons, it is quite clear that it is less user friendly and not intuitive.

Alternative 2 :

- **Changes to kafka-acls.sh**
    - Kafka-acls.sh will allow to specify a custom `PrincipalBuilder` class using a new command line parameter *"-- principalBuilder"* and `PrincipalBuilder` configs using a new command line parameter *"--principalBuilder-properties"*.

- The *"--allow-principal"* will take list of properties as follows :

```
bin/kafka-acls.sh ...... --principalBuilder <PrincipalBuilder-class> --principalBuilder-properties
<PrincipalBuilder-properties> --add --allow-principal <principal-properties> --allow-principal
<principal-properties> ...... --operations Read,Write --topic Test-topic
```

- Add a new API to `PrincipalBuilder` :

```
public interface PrincipalBuilder extends Configurable {
...

  /**
   * Build a Principal using the provided configs.
   *
   * @param  principalConfigs  configs used to create the Principal
   * @return Principal
   */
  Principal buildPrincipal(Map<String, ?> principalConfigs);

...
}
```

- The specified `PrincipalBuilder` class will be responsible for building the `Principal` using the <principal-properties>.
- The `Principal` generated by this `PrincipalBuilder` can then be included in `KafkaPrincipal` using the new constructor specified above.
- The "--principalBuilder" and *"--principalBuilder-properties"* parameters are optional. If its not specified, the Kafka-acls.sh would still work as it does today.

- This was rejected as per discussions on the email thread as this is a nice to have feature but there is no urgent need for this.