KIP-498: Add client-side configuration for maximum response size to protect against OOM

Status

Current state: *Under discussion*

Discussion thread: Originally: msg-55658. Follow-up: here.

JIRA: KAFKA-4090 - Getting issue details... STATUS

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

Motivation

This KIP addresses a minor addition to the set of configuration properties of Kafka producers in order to make the client validates the size of a message received from brokers (as denoted by the first four bytes of the response). The motivation is to prevent out-of-memory (OOM) errors when the size to be allocated is abnormally high.

A typical scenario where this vulnerability to OOM is exhibited consists in a client sending a plaintext request to an SSL endpoint on a broker. When this happens, the broker expects a Client Hello as part of a new TLS handshake. It detects the request is invalid [*] and sends back an Alert to the client. The client reads the response starting with the first four bytes to infer the size of the reply. With TLS v2.2, this results in 352,518,912 bytes trying to be allocated.

As reported in KAFKA-4090, this can happen when the producer property `security.protocol` is not defined, because the client then defaults to the plaintext transport layer implementation.

Public Interfaces

As part of the client-side configuration for producers and consumers, the property **`max.response.size`** will be added.

Note that this property has a different purpose from **`fetch.max.bytes`** defined for consumers. The latter is used to control the size of batches to be received by a consumer, while the former applies to any message read from the socket. Also note **`fetch.max.bytes`** will not put a limit the size of the first batch received by a consumer.

Proposed Changes

This new element of configuration was originally suggested on the discussion thread and mirrors what already exists server-side, i.e. the property `socket.request.max.bytes` - which was actually added in brokers for the very same reason.

The class `org.apache.kafka.common.network.NetworkReceive` which is used both client- and server-side already takes a maximum receive size and throws a `org.apache.kafka.common.network.InvalidReceiveException` when a message indicates a higher value. Therefore the change will mostly consists in propagating the value assigned to the property `socket.response.max.bytes` (or a sensible default) to the `NetworkReceive` (and the classes in-between). More precisely, in the method `org.apache.kafka.clients.producer.KafkaProducer#newSender`, the value can be provided to the constructor of the `org.apache.kafka.common.network.Selector` used later on by the client.

This proposed change was influenced by the following considerations:

- It does not modify the message decoding logic on the client. That is, the client is not trying to identify a failed SSL handshake by forward-reading a message. Indeed, doing so would affect the code path for *all* messages, thus would be likely introducing a better-avoided overhead;
- It introduces minimal change in the codebase. Because this functionality is already implemented server-side and the client uses the same network-level classes as the server, propagating the value of this new property is a minimal-impact, low-risk change. It is also fully transparent from a server perspective.
- It is intended to exhibit no change in behaviour for the existing clients, as long as the default value for the maximum allowed size is above that of any valid message (note: need to check this).

This proposal appears to come with multiple drawbacks, some of which may be considered as red flags.

Incorrect failure mode

As was experimented and as can be seen as part of the integration tests, when an invalid size is detected and the exception `InvalidReceiveException` is thrown, consumers and producers keeps retrying until the poll timeout expires or the maximum number of retries is reached. This is incorrect if we consider the use case of protocol mismatch which motivated this change. Indeed, producers and consumers would need to fail fast as retries will only prolong the time to remediation from users/administrators.

Limited scope

The proposed change cannot provide an definite guarantee against OOM in all scenarios - for instance, it will still manifest if the maximum size is set to 100 MB and the JVM is under memory pressure and have less than 100 MB of allocatable memory.

Illegitimate message rejection

Even worse: what if the property is incorrectly configured and legitimate messages not reaching the client?

Potential confusing

- The name **`max.response.size`** intends to mirror the existing **`max.request.size`** from the producer's configuration properties. However, **`max.request.size`** intends to check the size of producer records as provided by a client; while **`max.response.size`** is to check the size directly decoded from the network according to Kafka's binary protocol.
- On the broker, the property **`socket.request.max.bytes`** is used to validate the size of messages received by the server. The new property serves the same purpose, which introduces duplicated semantic, even if one property is characterised with the keyword "request" and the other with "response", in both cases reflecting the perspective adopted from either a client or a server.

Compatibility, Deprecation, and Migration Plan

This feature preserves backward compatibility. A default maximal size should be chosen tactically in order to avoid any change in behaviour when consuming large messages (*i.e.* legitimate responses should not be rejected). The same default value as `socket.request.max.bytes`, or 104,857,600 bytes, could be used ?.

Rejected Alternatives

None.

[*] Notes

The server-side SSL engine detects the message is invalid as confirmed in server logs:

```
kafka-network-thread-0-ListenerName(SSL)-SSL-4, fatal error: 80: problem unwrapping net record
javax.net.ssl.SSLException: Unrecognized SSL message, plaintext connection?
kafka-network-thread-0-ListenerName(SSL)-SSL-4, SEND TLSv1.2 ALERT:  fatal, description = internal_error
kafka-network-thread-0-ListenerName(SSL)-SSL-4, WRITE: TLSv1.2 Alert, length = 2
```

And sends back to the client the following sequence of bytes `15 03 03 00 02 02 50`:

Message bytes	0x15	0x0303	0x0002	0x02	0x50
Field	Record type	Version	Length	Handshake Type	Alert desc.
Field value	ALERT	3.3 (TLS v1.2)	2 bytes	Server Hello	internal_error(80)

The client tries to allocate 0x15030300 = 352,518,912 bytes which can result in OOM depending on the available heap size.

Reference

RFC 5246 - TLS v1.2