KIP-507: Securing Internal Connect REST Endpoints

- Status
- Motivation
- Proposed Changes
  - Additional Connect subprotocol
  - Key rotation, request signing, request verification
  - Requests with expired keys
  - New JMX worker metric
    - Backwards compatibility
    - Enabling request signature verification on a cluster
      - Upgrade steps
      - Possible failure scenarios
        - Failure to relay task configurations to leader due to incorrect configuration
        - Accidental disabling of internal request verification
    - Reverting an upgrade
      - Via connect.protocol config
      - Via worker version downgrade
    - Migrating to a new request signature algorithm
- Rejected Alternatives
  - Configurable inter-worker headers
  - Replace endpoint with Kafka topic
  - Distribute session key via Connect protocol

Status

Current state: *Accepted*

Discussion thread: here

Vote thread: here

JIRA: Unable to render Jira issues macro, execution error.

PR: https://github.com/apache/kafka/pull/7310

Released: 2.4.0

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

Motivation

The Connect framework uses an internal REST endpoint to relay task configurations from follower worker nodes to the leader. This endpoint is unique in that it is only meant to be invoked by Connect workers; every other endpoint is documented as part of the public REST API. This in turn leads to a problem when the Connect REST API is secured and authentication is required; every other kind of request will read user-supplied credentials for authentication and use those credentials when forwarding requests to the leader. However, since workers query this internal endpoint entirely on their own, they can't rely on authentication credentials that would be supplied in a user's REST request.

Successful requests this endpoint allow for arbitrary rewrite of task configurations, which is a significant security vulnerability that could lead to leaking of topic data, writing arbitrary data to topics, and other serious problems. As a result, it is imperative that access to this endpoint be restricted in order for any Kafka Connect cluster to be considered truly secure.

An ideal restriction of this endpoint would guarantee that requests made to it come exclusively from another work in the same cluster. Although mutual authentication via TLS, for example, may seem like a viable approach, this only accomplishes *authentication* and not *authorization*; that is, it verifies that the request comes from a trusted party with a given identity, but it does not make distinctions about whether party should be allowed to perform actions on the cluster. If mutual authentication is used for the Connect REST API, then this endpoint is still effectively unsecured since any user of the public REST API may also access this endpoint.

It should be noted that the goal here is not to completely secure any Kafka Connect cluster, but rather to patch an existing security hole for clusters that are already intended to be secure. A few examples of steps that should be taken in order to secure a Kafka Connect cluster include securing the public REST API (which can be done using a Connect REST extension), securing the worker group (which can be done with the use of ACLs on the Kafka broker), and securing the internal topic used by Connect to store configurations, statuses, and offsets for connectors (which can also be done with the use of ACLs on the Kafka broker). If any of these steps are not taken, the cluster is insecure anyways; therefore, relying on these precautions being in place in order to implement a fix for the problem posed by the internal REST endpoint used by Connect is acceptable.

However, it is not a goal of this KIP to limit the rollout of the new features based on whether the Connect cluster is already secure. With the pluggable nature of Kafka and Kafka Connect authorization, it would be difficult to know if all of the important Connect resources (worker group, internal topics, and Kafka Connect REST API to name a few) are actually secured. Additionally, the presence of other attack vectors shouldn't be justification for opening up a new one; it seems particularly dangerous in the event that someone accidentally misconfigures their Connect cluster and this endpoint ends up being exposed even though the user believes it to be protected.

Public Interfaces

There will be five new configurations added for distributed workers:

- `inter.worker.key.generation.algorithm`
  - **Purpose:** the algorithm used to generate session keys
  - **Type**: string
  - **Default:** `"HmacSHA256"`
  - **Importance:** low
- `inter.worker.key.size`
  - **Purpose:** the size of generated session keys, in bits; if null, the default key size for the generation algorithm will be used (see the KeyGenerator Javadocs; specifically: "In case the client does not explicitly initialize the KeyGenerator (via a call to an `init` method), each provider must supply (and document) a default initialization.")
  - **Type:** int
  - **Default:** `null`
  - **Importance:** low
- `inter.worker.key.ttl.ms`
  - **Purpose:** how often to force a rotation of the internal key used for request validation, or `0` if forced rotation should never occur
  - **Type:** long
  - **Default:** `3600000` (one hour)
  - **Importance:** low
- `inter.worker.signature.algorithm`
  - **Purpose:** the algorithm to use to sign internal requests when sent from a follower worker to the leader
  - **Type:** string
  - **Default:** `"HmacSHA256"`
  - **Importance:** low
- `inter.worker.verification.algorithms`
  - **Purpose:** a list of supported algorithms for verifying internal requests that are received by the leader from a follower. This list must include the value provided for the `inter.worker.signature.algorithm` property
  - **Type:** list
  - **Default:** `"HmacSHA256"`
  - **Importance:** low

Barring any need for heightened levels of security compliance, none of these configurations should need to be altered and the defaults should suffice.

The default value for the `connect.protocol` configuration will also be altered from `compatible` to `sessioned`, so that the new request signing behavior proposed by this KIP will be enabled by default (once all workers in a cluster support it).

Additionally, although not part of the public API, the `POST /connectors/<name>/tasks` endpoint will be effectively disabled for public use. This endpoint should never be called by users, but since until now there hasn't been anything to prevent them from doing so, it should still be noted that anything that relies that endpoint will no longer work after these changes are made. The expected impact of this is low, however; the Connect framework (and the connectors it runs) handle the generation and storage of task configurations and there's no discernible reason for using that endpoint directly instead of going through the public Connect REST API.

Proposed Changes

Additional Connect subprotocol

A new Connect subprotocol, `sessioned`, will be implemented that will be identical to the cooperative incremental protocol but a higher protocol version number (2, instead of the current version for cooperative incremental rebalancing, which is 1). One downside of this approach is that the use of cooperative incremental assignments will be required in order to enable this new security behavior; however, given the lack of any serious complaints about the new rebalancing protocol thus far, this seems preferable to trying to enable this behavior across both assignment styles. If the `connect.protocol` property is set to `sessioned`, the worker will advertise this new `sessioned` protocol to the Kafka group coordinator as a supported (and, currently, most preferable) protocol.

If the `sessioned` protocol is then agreed on by the cluster during group coordination, a session key will be randomly generated by the leader and distributed to the cluster via the config topic. This key will be used by followers to sign requests to the internal endpoint, and verified by the leader to ensure that the request came from a current group member. It is imperative that inter-worker communication have some kind of transport layer security; otherwise, this session key will be leaked during rebalance to anyone who can eavesdrop on request traffic.

Key rotation, request signing, request verification

Periodically (with frequency dictated by the `inter.worker.key.ttl.ms` property), the leader will compute a new session key and distribute it to the cluster.

The default algorithm used to sign requests will be `HmacSHA256`; this algorithm is guaranteed to be supported on all implementations of the Java Platform (source). However, users will be able to configure their cluster to use other algorithms with the `inter.worker.signature.algorithm` property if, for example, the default is not suitable for compliance with an existing security standard.

Similarly, the default algorithm used to generate request keys will also be `HmacSHA256`; again, this algorithm is guaranteed to be supported on all implementations of the Java Platform (source). And again, users will be able to configure their cluster to use other algorithms or keys of a different size with the `inter.worker.key.generation.algorithm` and `inter.worker.key.size` properties, respectively.

Each signed request will include two headers:

- `X-Connect-Authorization`: the signature of the request body (base 64 encoded)
- `X-Connect-Request-Signature-Algorithm`: the key algorithm used to sign the request

When a request is received by the leader, the request signature algorithm described by the `X-Connect-Request-Signature-Algorithm` header will be used to sign the request body and the resulting signature will be checked against the contents of the `X-Connect-Authorization` header. If the contents do not match, or the request signature algorithm is not in the list of permitted algorithms controlled by the `inter.worker.verification.algorithms` property, the request will be rejected.

Valid requests will be met with an HTTP 200 response; invalid requests will be met with either HTTP 400 (bad request) should they lack the required signature and signature algorithm headers, specify an invalid (non-base-64-decodable) signature, or specify a signature algorithm that isn't permitted by the leader, and HTTP 403 (forbidden) if they contain well-formed values for the signature and signature algorithm headers, but which fail request verification.

Requests with expired keys

The leader will only accept requests signed with the most current key. However, Connect follower workers may routinely experience small delays when reading the new key. Rather than always logging such task configuration failure and retry attempts as errors (the current behavior), Connect's distributed herder will be modified slightly to handle such HTTP 403 responses for this task configuration request by quietly retrying them with the latest key for up to 1 minute. If failures persist for more than 1 minute, they will be logged as errors.

New JMX worker metric

Finally, a new worker JMX metric will be exposed that can be used to determine whether the new behavior proposed by this KIP is enabled:

- **MBean**: `kafka.connect:type=connect-worker-rebalance-metrics`
- **Metric name**: `connect-protocol`
- **Description**: `The Connect protocol used by this cluster`
- **Value**: The Connect subprotocol in use based on the latest join group response for this worker joining the Connect cluster.

Compatibility, Deprecation, and Migration Plan

Backwards compatibility

All of the proposed configurations here have default values, making them backwards compatible.

Enabling request signature verification on a cluster

Upgrade steps

These steps should be completed on each worker in the cluster as part of a rolling upgrade:

1. Shut down the worker.
2. If necessary, either remove the `connect.protocol` property from the worker config file, or set it to `sessioned`.
3. Upgrade the Connect JARs for that worker to the new Kafka Connect release that has support for internal request signing.
4. Start the worker.

Once every worker has been restarted, internal request verification should become automatically enabled. While there are any workers left in the cluster that are either not-yet upgraded, or that use a different `connect.protocol` (i.e., `eager` or `compatible`), internal request verification will remain disabled; no session keys will be distributed by the leader, requests will not be signed by followers, and the leader will not protect its `POST /connectors/<connector>/tasks` endpoint.

An info-level log message will be emitted by any worker that has newly joined a cluster with internal request validation enabled so that Connect cluster administrators can be assured that their clusters are secure.

Possible failure scenarios

Failure to relay task configurations to leader due to incorrect configuration

The only seriously bad scenario is if a follower worker is configured to use a request signing algorithm that isn't allowed by the leader. In this case, any failure will only occur if/when that follower starts up a connector and then has to forward tasks for that connector to the leader, which may not happen immediately. Once that failure occurs, an endless failure loop will occur wherein the follower endlessly retries to send those task configurations to the leader and pauses by the backoff interval in between each failed request.

There will be two symptoms that could indicate to the user that this has occurred:

1. Failure of connectors hosted by the follower worker to spawn tasks
2. Error-level log messages emitted by the follower worker

There are two ways to rectify this situation; either shut down the follower and restart it after editing its configuration to use a request signing algorithm permitted by the leader, or shut down all other workers in the cluster that do not permit the request signing algorithm used by the follower, reconfigure them to permit it, and then restart them.

This scenario is unlikely to occur with any normal usage of Connect, but the circumstances leading to it will need to be called out very carefully (in the upgrade notes for any release that begins to support this new behavior) in order to avoid putting the cluster into a bad state and flooding logs with scary-looking error messages. Additionally, it will be vital to design appropriate error messages for this scenario so that users can dig themselves out of that hole on their own.

Differing values for any of the other configurations in this KIP shouldn't actually be too problematic, given that the remaining ones all deal with key generation/rotation, which is only handled by one worker at a time (the leader).

Accidental disabling of internal request verification

There are two scenarios that may indicate that the user has *accidentally* disabled internal request verification; these should result in warn-level log messages that carefully explain that request verification is disabled, what has caused request verification to become disabled, and how the user can fix this if desired.

1. A worker is started up with `connect.protocol` set to something other than `sessioned` (this may occur if the value was accidentally left in the worker file during an upgrade)
2. A worker with `connect.protocol` set to `sessioned` (or, in the future, any protocol that enables internal request signing) joins the cluster group, and it is observed that the subprotocol used by the cluster does not indicate support for internal request singing (this may occur in the same scenario as above, but would be caught from the perspective of a correctly-configured worker as opposed to an incorrectly-configured worker and should be logged as well)

Neither of these scenarios warrant error-level logging messages as they could theoretically be brought about by an intentional downgrade.

The newly-proposed `connect-protocol` JMX metric can be used to monitor whether internal request verification is enabled for a cluster; if its value is `sessioned` (or, presumably, a later protocol), then request verification should be enabled.

Reverting an upgrade

Via `connect.protocol` config

The group coordination protocol will be used to ensure that all workers in a cluster support verification of internal requests before this behavior is enabled; therefore, a rolling upgrade of the cluster will be possible. In line with the regression plan for KIP-415: Incremental Cooperative Rebalancing in Kafka Connect, if it is desirable to disable this behavior for some reason, the `connect.protocol` configuration can be set to `compatible` or `default` for one (or more) workers, and it will automatically be disabled.

Via worker version downgrade

It should also be noted that the above will occur automatically if a worker is downgraded to a prior release of Kafka Connect that does not support the `sessioned` protocol. If this occurs, the worker will begin emitting error-level log messages when it reads session keys from the config topic. However, the worker will be otherwise unaffected and will continue to function properly (but without the security benefit of internal request verification).

Migrating to a new request signature algorithm

If a new signature algorithm should be used, a rolling upgrade will be possible with the following steps (assuming a new algorithm of `HmacSHA489`):

1. Add `HmacSHA489` to the `internal.key.verification.algorithms` list for each worker, and restart them one-by-one
2. Change the `internal.key.signature.algorithm` property for each worker to `HmacSHA489`, and restart them one-by-one
3. (Optional) Remove the old algorithm from the `internal.key.verification.algorithms` list for each worker, and restart them one-by-one

Rejected Alternatives

Configurable inter-worker headers

Summary: A new worker configuration would be added that would control auth headers used by workers when making requests to the internal endpoint.

Rejected because: The additional complexity of another required configuration would be negative for users; security already isn't simple to implement with Kafka Connect, and requiring just one more thing for them to add should be avoided if possible. Also, the use of static headers isn't guaranteed to cover all potential auth mechanisms, and would require manual rotation by reconfiguring the worker.

Replace endpoint with Kafka topic

Summary: The REST endpoint could be removed entirely and replaced with a Kafka topic. Either an existing internal Connect topic (such as the configs topic) could be used, or a new topic could be added to handle all non-forwarded follower-to-leader communication.

Rejected because: Achieving consensus in a Connect cluster about whether to begin engaging in this new topic-based protocol would require either reworking the Connect group coordination protocol or installing several new configurations and a multi-stage rolling upgrade in order to enable it. Requiring new configurations and a multi-stage rolling upgrade for the default use case of a simple version bump for a cluster would be a much worse user experience, and if the group coordination protocol is going to be reworked, we might as well just use the group coordination protocol to distribute keys instead. Additionally, the added complexity of switch from a synchronous to an asynchronous means of communication for relaying task configurations to the leader would complicate the implementation enough that reworking the group coordination protocol might even be a simpler approach with smaller changes required.

Distribute session key via Connect protocol

Summary: Instead of distributing a session key via the config topic, include the session key as part of the worker assignment handed out during rebalance via the Connect protocol. Periodically force a rebalance in order to rotate session keys.

Rejected because: The implementation complexity of adding a session key to the rebalance protocol would be quite high, and the additional API would complicate the code base significantly. Additionally, there are few, if any advantages, compared to distributing the keys via the config topic.