# KIP-1006: Remove SecurityManager Support



## Status

**Current state**: Draft

**Discussion thread**: https://lists.apache.org/thread/hyxb9gwpsjdbxjs9rw8qsg6ws0nvlx0t

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

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

## Motivation

With the adoption of JEP 411: Deprecate the SecurityManager for Removal in Java 17, the SecurityManager is now deprecated, and will be removed in an unknown future version.

As Kafka supports multiple versions of Java (8, 11, 17, 21 for 3.x, 11, 17, 21 for 4.x), we should decide how to handle this deprecation and removal, and what replacements or mitigations we can provide for users of Kafka.

Kafka has a compile-time dependency on the `AccessController` class in `clients` , `core` , and `connect:runtime` , and the removal of SecurityManager and it's accompanying classes would make these modules un-buildable with their current implementation.

## Public Interfaces

The Kafka project should to define a major release in which SecurityManager support will be removed. It is not possible to remove support for this feature in a minor release, as it would constitute a breaking change.

There are two strategies for approaching the removal that we should pursue concurrently:

### Remove SecurityManager support as soon as possible

As SecurityManager has not yet been removed from Java, we have the opportunity to remove it ourselves first.

For example, if SecurityManager is removed from the Kafka codebase in 4.0, and SecurityManager is removed from Java 25, users of Kafka 4.x can immediately upgrade to Java 25 without special considerations.

However, there are some constraints on how early we can remove support:

1. Java 11 still has the non-deprecated SecurityManager, and Kafka 4.0 must support Java 11. If we were to remove SecurityManager support from Kafka 4.0, users on Java 11 would experience a removal without a prior deprecation.
2. Java 17 does not have the replacement Subject API https://bugs.openjdk.org/browse/JDK-8267108 , and Kafka 4.0 must support Java 17. We would need dual support for the old and new APIs.

Therefore, we will plan to remove SecurityManager support in the Kafka major version which drops support for JDK 17.

We can notify users of this removal with a log message, and with a notice in the Kafka documentation.

### Remove compile-time usages & detect SecurityManager removal at runtime

The longer it takes for Kafka to remove SecurityManager support, the more likely that the Java removal will take place before Kafka's removal is complete. To be prepared, we should anticipate that the project will need to support both versions with and without SecurityManager, using the same codebase & artifacts.

In an upcoming minor release after this KIP, the deprecated calls will be changed to use reflection. If the legacy APIs are available, they will be used. If not, the modern APIs will be called reflectively.

In the major release which removes SecurityManager support, the reflection will be replaced with direct calls to the modern APIs.

# Proposed Changes

A static utility in `clients` will reflectively examine the presence or absence of the legacy and modern implementations.

If it finds that the legacy implementation is available, it will use it. If not, the modern implementation is substituted.

| Class | Legacy implementation | Modern implementation |
|---|---|---|
| SaslClientCallbackHandler<br>OAuthBearerSaslClientCallbackHandler | Subject.getSubject | Subject.current |
| SaslClientAuthenticator<br>SaslServerAuthenticator | Subject#doAs | Subject#callAs |
| ClassLoaderFactory<br>PluginScanner<br>RemoteLogManager<br>SynchronizationTest | AccessController#doPrivileged | Pass-through/call runnable directly |

# Compatibility, Deprecation, and Migration Plan

Users upgrading to a version of Java or Kafka without SecurityManager support will need to disable their security manager implementations/configurations.

Users upgrading to Java 17+ (in which SecurityManager was deprecated) already receive a warning message on startup.

# Test Plan

There are currently no tests exercising the SecurityManager interface in Kafka, and no new tests for the end-to-end behavior will be added for this project.

Further tests will compare the reflective implementations to the old and new APIs. This is to ensure that changing the legacy APIs to reflection, and that later when the reflection is replaced with the new APIs, the behavior will stay the same throughout. These tests can mock the AccessController and Subject classes to make it appear that the removal has taken place.

# Rejected Alternatives

### Wait until SecurityManager is removed to change the implementations (do nothing)

When the first Java version without SecurityManager is released, there will have already been a number of Kafka minor and patch releases made. All of these will be unable to run with the new Java version, regardless of if the user was using the SecurityManager at all. This also includes users of the clients, who will be unable to run in the latest Java version until the clients have been released again.

Similarly, developers will be unable to build older branches of Kafka with the new Java version, and this will be generally inconvenient. When we front-load the effort in this migration, we can ensure that when the removal takes place, the most recent Kafka releases are already compatible.

Since removing SecurityManager is a breaking change, we would have to either delay support for the new Java version significantly, or perform a major release of Kafka when we might not be prepared for it. These negatives outweigh the maintenance burden of making these function calls reflective.

### Set a removal date for Java 11 / Java 17 support

This should be delegated to another KIP with a dedicated discussion, and the removal of SecurityManager support can be included in whatever version changes the minimum java version to 21.

### Implement a alternative to the SecurityManager for use with Kafka

The SecurityManager is a flawed design, and we should not replicate it or try to solve those problems in a similar way. Instead should encourage users to use the process boundary as a security boundary.