# KIP-7 - Security - IP Filtering

## Status

**Current state**: *Not Accepted <Closed>*

**Discussion thread**: *here*

**JIRA**: *KAFKA-1810*

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

## Motivation

Currently anyone with sufficient knowledge can write their own Kafka clients to read/write messages from/to Kafka. There are scenarios where having this type of open access is not desirable, and the ability to restrict incoming connections would be beneficial. Some examples are

- Financial Services - Systems that contain PCI / PII such as addresses, credit cards numbers etc
- Healthcare - Patient information, HIPAA
- Any other sensitive information
- Protection against logical data corruption (eg. Dev writing to Prod or vice versa)

This type of functionality could also be enforced via network administration, (Network firewalls, IPTables etc) but often times there is a disconnect between the operators of the system and those network teams in large organizations. Giving (software/systems) administrators the ability to control their own environment independent of external groups would be beneficial.

The broader security initiative will add more robust controls for these types of environments, and this proposal could be integrated with that work at the appropriate time. This is also the specific request of a large financial services company.

## Public Interfaces

The changes proposed here do not affect public interfaces or otherwise interfere with backward compatibility

Two additional (optional) parameters would be added to the Kafka broker configuration.

security.ip.filter.rule.type
security.ip.filter.list

## Proposed Changes

Administrators would list the type of filtering that they are implementing, either Whitelists - using the value "allow" or Blacklists - using the value "deny" and a list of IP Ranges represented in CIDR notation: http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing.

The rule type is mutually exclusive, either you are specifying a group of IPs to accept OR to reject, depending on the rule. Using CIDR notation is more compact than passing ranges and easy to calculate with a number of online tools. So an example configuration might be:

```
security.ip.filter.rule.type=allow
security.ip.filter.list=192.168.2.1/32,192.168.10.2/24,10.1.1.1/28
```

**A single IP address is represented with the prefix size /32 for IPv4 and /128 for IPv6*

Any addresses that did not fall within the specified range(s) would be rejected by the socket acceptor thread and a WARN message output in the logs. The rejection happens by throwing an IpFilterException and closing socket for that connection. Conversely, if in the same scenario "deny" was specified for security.ip.filter.rule.type then only connections that do fall in the range listed would be "rejected" and a WARN message output in the logs.

The solution proposed calls for an additional class in the network package and an additional check in the acceptor thread, as mentioned above. Translation of the CIDR ranges into upper and lower boundaries happens on server startup so the overhead of checking this in the acceptor thread should be O(N) where N is the number of ranges specified in the configuration. The actual comparison is done by converting the connection IP address into a BigInteger representation and comparing it with the upper and lower boundaries of each CIDR range. First match exits the lookup. The solution also supports IPv6 addresses (hence the need for BigInt vs Int).

## Compatibility, Deprecation, and Migration Plan

- N/A

## Rejected Alternatives

KAFKA-1688

Discusses the implementation for a more comprehensive authorization policy. Completion on that and the larger umbrella JIRA is TBD. This KIP would provide another way for administrators to control access to their cluster.