package com.twitter.tweetypie.client\_id

import com.twitter.finagle.mtls.authentication.EmptyServiceIdentifier

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

import com.twitter.finagle.mtls.transport.S2STransport

import com.twitter.finagle.thrift.ClientId

import com.twitter.servo.util.Gate

import com.twitter.strato.access.Access

import com.twitter.strato.access.Access.ForwardedServiceIdentifier

object ClientIdHelper {

val UnknownClientId = "unknown"

def default: ClientIdHelper = new ClientIdHelper(UseTransportServiceIdentifier)

/\*\*

\* Trims off the last .element, which is usually .prod or .staging

\*/

def getClientIdRoot(clientId: String): String =

clientId.lastIndexOf('.') match {

case -1 => clientId

case idx => clientId.substring(0, idx)

}

/\*\*

\* Returns the last .element without the '.'

\*/

def getClientIdEnv(clientId: String): String =

clientId.lastIndexOf('.') match {

case -1 => clientId

case idx => clientId.substring(idx + 1)

}

private[client\_id] def asClientId(s: ServiceIdentifier): String = s"${s.service}.${s.environment}"

}

class ClientIdHelper(serviceIdentifierStrategy: ServiceIdentifierStrategy) {

private[client\_id] val ProcessPathPrefix = "/p/"

/\*\*

\* The effective client id is used for request authorization and metrics

\* attribution. For calls to Tweetypie's thrift API, the thrift ClientId

\* is used and is expected in the form of "service-name.env". Federated

\* Strato clients don't support configured ClientIds and instead provide

\* a "process path" containing instance-specific information. So for

\* calls to the federated API, we compute an effective client id from

\* the ServiceIdentifier, if present, in Strato's Access principles. The

\* implementation avoids computing this identifier unless necessary,

\* since this method is invoked on every request.

\*/

def effectiveClientId: Option[String] = {

val clientId: Option[String] = ClientId.current.map(\_.name)

clientId

// Exclude process paths because they are instance-specific and aren't

// supported by tweetypie for authorization or metrics purposes.

.filterNot(\_.startsWith(ProcessPathPrefix))

// Try computing a value from the ServiceId if the thrift

// ClientId is undefined or unsupported.

.orElse(serviceIdentifierStrategy.serviceIdentifier.map(ClientIdHelper.asClientId))

// Ultimately fall back to the ClientId value, even when given an

// unsupported format, so that error text and debug logs include

// the value passed by the caller.

.orElse(clientId)

}

def effectiveClientIdRoot: Option[String] = effectiveClientId.map(ClientIdHelper.getClientIdRoot)

def effectiveServiceIdentifier: Option[ServiceIdentifier] =

serviceIdentifierStrategy.serviceIdentifier

}

/\*\* Logic how to find a [[ServiceIdentifier]] for the purpose of crafting a client ID. \*/

trait ServiceIdentifierStrategy {

def serviceIdentifier: Option[ServiceIdentifier]

/\*\*

\* Returns the only element of given [[Set]] or [[None]].

\*

\* This utility is used defensively against a set of principals collected

\* from [[Access.getPrincipals]]. While the contract is that there should be at most one

\* instance of each principal kind present in that set, in practice that has not been the case

\* always. The safest strategy to in that case is to abandon a set completely if more than

\* one principals are competing.

\*/

final protected def onlyElement[T](set: Set[T]): Option[T] =

if (set.size <= 1) {

set.headOption

} else {

None

}

}

/\*\*

\* Picks [[ServiceIdentifier]] from Finagle SSL Transport, if one exists.

\*

\* This works for both Thrift API calls as well as StratoFed API calls. Strato's

\* [[Access#getPrincipals]] collection, which would typically be consulted by StratoFed

\* column logic, contains the same [[ServiceIdentifier]] derived from the Finagle SSL

\* transport, so there's no need to have separate strategies for Thrift vs StratoFed

\* calls.

\*

\* This is the default behavior of using [[ServiceIdentifier]] for computing client ID.

\*/

private[client\_id] class UseTransportServiceIdentifier(

// overridable for testing

getPeerServiceIdentifier: => ServiceIdentifier,

) extends ServiceIdentifierStrategy {

override def serviceIdentifier: Option[ServiceIdentifier] =

getPeerServiceIdentifier match {

case EmptyServiceIdentifier => None

case si => Some(si)

}

}

object UseTransportServiceIdentifier

extends UseTransportServiceIdentifier(S2STransport.peerServiceIdentifier)

/\*\*

\* Picks [[ForwardedServiceIdentifier]] from Strato principals for client ID

\* if [[ServiceIdentifier]] points at call coming from Strato.

\* If not present, falls back to [[UseTransportServiceIdentifier]] behavior.

\*

\* Tweetypie utilizes the strategy to pick [[ServiceIdentifier]] for the purpose

\* of generating a client ID when the client ID is absent or unknown.

\* [[PreferForwardedServiceIdentifierForStrato]] looks for the [[ForwardedServiceIdentifier]]

\* values set by stratoserver request.

\* The reason is, stratoserver is effectively a conduit, forwarding the [[ServiceIdentifier]]

\* of the \_actual client\_ that is calling stratoserver.

\* Any direct callers not going through stratoserver will default to [[ServiceIdentfier]].

\*/

private[client\_id] class PreferForwardedServiceIdentifierForStrato(

// overridable for testing

getPeerServiceIdentifier: => ServiceIdentifier,

) extends ServiceIdentifierStrategy {

val useTransportServiceIdentifier =

new UseTransportServiceIdentifier(getPeerServiceIdentifier)

override def serviceIdentifier: Option[ServiceIdentifier] =

useTransportServiceIdentifier.serviceIdentifier match {

case Some(serviceIdentifier) if isStrato(serviceIdentifier) =>

onlyElement(

Access.getPrincipals

.collect {

case forwarded: ForwardedServiceIdentifier =>

forwarded.serviceIdentifier.serviceIdentifier

}

).orElse(useTransportServiceIdentifier.serviceIdentifier)

case other => other

}

/\*\*

\* Strato uses various service names like "stratoserver" and "stratoserver-patient".

\* They all do start with "stratoserver" though, so at the point of implementing,

\* the safest bet to recognize strato is to look for this prefix.

\*

\* This also works for staged strato instances (which it should), despite allowing

\* for technically any caller to force this strategy, by creating service certificate

\* with this service name.

\*/

private def isStrato(serviceIdentifier: ServiceIdentifier): Boolean =

serviceIdentifier.service.startsWith("stratoserver")

}

object PreferForwardedServiceIdentifierForStrato

extends PreferForwardedServiceIdentifierForStrato(S2STransport.peerServiceIdentifier)

/\*\*

\* [[ServiceIdentifierStrategy]] which dispatches between two delegates based on the value

\* of a unitary decider every time [[serviceIdentifier]] is called.

\*/

class ConditionalServiceIdentifierStrategy(

private val condition: Gate[Unit],

private val ifTrue: ServiceIdentifierStrategy,

private val ifFalse: ServiceIdentifierStrategy)

extends ServiceIdentifierStrategy {

override def serviceIdentifier: Option[ServiceIdentifier] =

if (condition()) {

ifTrue.serviceIdentifier

} else {

ifFalse.serviceIdentifier

}

}