package com.twitter.product\_mixer.core.service.debug\_query

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

import com.twitter.inject.annotations.Flag

import com.twitter.product\_mixer.core.functional\_component.common.access\_policy.AccessPolicy

import com.twitter.product\_mixer.core.functional\_component.common.access\_policy.AccessPolicyEvaluator

import com.twitter.product\_mixer.core.model.common.identifier.ComponentIdentifierStack

import com.twitter.product\_mixer.core.module.product\_mixer\_flags.ProductMixerFlagModule.ServiceLocal

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.Authentication

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.BadRequest

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.PipelineFailure

import com.twitter.turntable.{thriftscala => t}

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Basic class that provides a verification method for checking if a call to our debugging

\* features is allowed/authorized to make said call.

\* @param isServiceLocal Whether the service is being run locally.

\*/

@Singleton

class AuthorizationService @Inject() (@Flag(ServiceLocal) isServiceLocal: Boolean) {

import AuthorizationService.\_

/\*\*

\* Check whether a call to a given product is authorized. Throws an [[UnauthorizedServiceCallException]]

\* if not.

\* @param requestingServiceIdentifier The Service Identifier of the calling service

\* @param productAccessPolicies The access policies of the product being called.

\* @param requestContext The request context of the caller.

\*/

def verifyRequestAuthorization(

componentIdentifierStack: ComponentIdentifierStack,

requestingServiceIdentifier: ServiceIdentifier,

productAccessPolicies: Set[AccessPolicy],

requestContext: t.TurntableRequestContext

): Unit = {

val isServiceCallAuthorized =

requestingServiceIdentifier.role == AllowedServiceIdentifierRole && requestingServiceIdentifier.service == AllowedServiceIdentifierName

val userLdapGroups = requestContext.ldapGroups.map(\_.toSet)

val accessPolicyAllowed = AccessPolicyEvaluator.evaluate(

productAccessPolicies = productAccessPolicies,

userLdapGroups = userLdapGroups.getOrElse(Set.empty)

)

if (!isServiceLocal && !isServiceCallAuthorized) {

throw new UnauthorizedServiceCallException(

requestingServiceIdentifier,

componentIdentifierStack)

}

if (!isServiceLocal && !accessPolicyAllowed) {

throw new InsufficientAccessException(

userLdapGroups,

productAccessPolicies,

componentIdentifierStack)

}

}

}

object AuthorizationService {

final val AllowedServiceIdentifierRole = "turntable"

final val AllowedServiceIdentifierName = "turntable"

}

class UnauthorizedServiceCallException(

serviceIdentifier: ServiceIdentifier,

componentIdentifierStack: ComponentIdentifierStack)

extends PipelineFailure(

BadRequest,

s"Unexpected Service tried to call Turntable Debug endpoint: ${ServiceIdentifier.asString(serviceIdentifier)}",

componentStack = Some(componentIdentifierStack))

class InsufficientAccessException(

ldapGroups: Option[Set[String]],

desiredAccessPolicies: Set[AccessPolicy],

componentIdentifierStack: ComponentIdentifierStack)

extends PipelineFailure(

Authentication,

s"Request did not satisfy access policies: $desiredAccessPolicies with ldapGroups = $ldapGroups",

componentStack = Some(componentIdentifierStack))