package com.twitter.visibility.rules

import com.twitter.guano.commons.thriftscala.PolicyInViolation

import com.twitter.spam.rtf.thriftscala.SafetyResultReason

import com.twitter.util.Memoize

import com.twitter.util.Time

import com.twitter.visibility.common.actions.ComplianceTweetNoticeEventType

import com.twitter.visibility.common.actions.LimitedEngagementReason

import com.twitter.visibility.configapi.params.RuleParam

import com.twitter.visibility.configapi.params.RuleParams.EnableSearchIpiSafeSearchWithoutUserInQueryDropRule

import com.twitter.visibility.features.Feature

import com.twitter.visibility.features.TweetSafetyLabels

import com.twitter.visibility.models.TweetSafetyLabel

import com.twitter.visibility.models.TweetSafetyLabelType

import com.twitter.visibility.rules.Condition.And

import com.twitter.visibility.rules.Condition.LoggedOutOrViewerOptInFiltering

import com.twitter.visibility.rules.Condition.Not

import com.twitter.visibility.rules.Condition.Or

import com.twitter.visibility.rules.Condition.SearchQueryHasUser

import com.twitter.visibility.rules.Condition.TweetComposedAfter

import com.twitter.visibility.rules.Condition.TweetHasLabel

import com.twitter.visibility.rules.Reason.\_

import com.twitter.visibility.rules.State.Evaluated

object PublicInterest {

object PolicyConfig {

val LowQualityProxyLabelStart: Time = Time.fromMilliseconds(1554076800000L)

val DefaultReason: (Reason, Option[LimitedEngagementReason]) =

(OneOff, Some(LimitedEngagementReason.NonCompliant))

val DefaultPolicyInViolation: PolicyInViolation = PolicyInViolation.OneOff

}

val policyInViolationToReason: Map[PolicyInViolation, Reason] = Map(

PolicyInViolation.AbusePolicyEpisodic -> AbuseEpisodic,

PolicyInViolation.AbusePolicyEpisodicEncourageSelfharm -> AbuseEpisodicEncourageSelfHarm,

PolicyInViolation.AbusePolicyEpisodicHatefulConduct -> AbuseEpisodicHatefulConduct,

PolicyInViolation.AbusePolicyGratuitousGore -> AbuseGratuitousGore,

PolicyInViolation.AbusePolicyGlorificationofViolence -> AbuseGlorificationOfViolence,

PolicyInViolation.AbusePolicyEncourageMobHarassment -> AbuseMobHarassment,

PolicyInViolation.AbusePolicyMomentofDeathDeceasedUser -> AbuseMomentOfDeathOrDeceasedUser,

PolicyInViolation.AbusePolicyPrivateInformation -> AbusePrivateInformation,

PolicyInViolation.AbusePolicyRighttoPrivacy -> AbuseRightToPrivacy,

PolicyInViolation.AbusePolicyThreattoExpose -> AbuseThreatToExpose,

PolicyInViolation.AbusePolicyViolentSexualConduct -> AbuseViolentSexualConduct,

PolicyInViolation.AbusePolicyViolentThreatsHatefulConduct -> AbuseViolentThreatHatefulConduct,

PolicyInViolation.AbusePolicyViolentThreatorBounty -> AbuseViolentThreatOrBounty,

PolicyInViolation.OneOff -> OneOff,

PolicyInViolation.AbusePolicyElectionInterference -> VotingMisinformation,

PolicyInViolation.MisinformationVoting -> VotingMisinformation,

PolicyInViolation.HackedMaterials -> HackedMaterials,

PolicyInViolation.Scam -> Scams,

PolicyInViolation.PlatformManipulation -> PlatformManipulation,

PolicyInViolation.MisinformationCivic -> MisinfoCivic,

PolicyInViolation.AbusePolicyUkraineCrisisMisinformation -> MisinfoCrisis,

PolicyInViolation.MisinformationGeneric -> MisinfoGeneric,

PolicyInViolation.MisinformationMedical -> MisinfoMedical,

)

val reasonToPolicyInViolation: Map[Reason, PolicyInViolation] = Map(

AbuseEpisodic -> PolicyInViolation.AbusePolicyEpisodic,

AbuseEpisodicEncourageSelfHarm -> PolicyInViolation.AbusePolicyEpisodicEncourageSelfharm,

AbuseEpisodicHatefulConduct -> PolicyInViolation.AbusePolicyEpisodicHatefulConduct,

AbuseGratuitousGore -> PolicyInViolation.AbusePolicyGratuitousGore,

AbuseGlorificationOfViolence -> PolicyInViolation.AbusePolicyGlorificationofViolence,

AbuseMobHarassment -> PolicyInViolation.AbusePolicyEncourageMobHarassment,

AbuseMomentOfDeathOrDeceasedUser -> PolicyInViolation.AbusePolicyMomentofDeathDeceasedUser,

AbusePrivateInformation -> PolicyInViolation.AbusePolicyPrivateInformation,

AbuseRightToPrivacy -> PolicyInViolation.AbusePolicyRighttoPrivacy,

AbuseThreatToExpose -> PolicyInViolation.AbusePolicyThreattoExpose,

AbuseViolentSexualConduct -> PolicyInViolation.AbusePolicyViolentSexualConduct,

AbuseViolentThreatHatefulConduct -> PolicyInViolation.AbusePolicyViolentThreatsHatefulConduct,

AbuseViolentThreatOrBounty -> PolicyInViolation.AbusePolicyViolentThreatorBounty,

OneOff -> PolicyInViolation.OneOff,

VotingMisinformation -> PolicyInViolation.MisinformationVoting,

HackedMaterials -> PolicyInViolation.HackedMaterials,

Scams -> PolicyInViolation.Scam,

PlatformManipulation -> PolicyInViolation.PlatformManipulation,

MisinfoCivic -> PolicyInViolation.MisinformationCivic,

MisinfoCrisis -> PolicyInViolation.AbusePolicyUkraineCrisisMisinformation,

MisinfoGeneric -> PolicyInViolation.MisinformationGeneric,

MisinfoMedical -> PolicyInViolation.MisinformationMedical,

)

val ReasonToSafetyResultReason: Map[Reason, SafetyResultReason] = Map(

AbuseEpisodic -> SafetyResultReason.Episodic,

AbuseEpisodicEncourageSelfHarm -> SafetyResultReason.AbuseEpisodicEncourageSelfHarm,

AbuseEpisodicHatefulConduct -> SafetyResultReason.AbuseEpisodicHatefulConduct,

AbuseGratuitousGore -> SafetyResultReason.AbuseGratuitousGore,

AbuseGlorificationOfViolence -> SafetyResultReason.AbuseGlorificationOfViolence,

AbuseMobHarassment -> SafetyResultReason.AbuseMobHarassment,

AbuseMomentOfDeathOrDeceasedUser -> SafetyResultReason.AbuseMomentOfDeathOrDeceasedUser,

AbusePrivateInformation -> SafetyResultReason.AbusePrivateInformation,

AbuseRightToPrivacy -> SafetyResultReason.AbuseRightToPrivacy,

AbuseThreatToExpose -> SafetyResultReason.AbuseThreatToExpose,

AbuseViolentSexualConduct -> SafetyResultReason.AbuseViolentSexualConduct,

AbuseViolentThreatHatefulConduct -> SafetyResultReason.AbuseViolentThreatHatefulConduct,

AbuseViolentThreatOrBounty -> SafetyResultReason.AbuseViolentThreatOrBounty,

OneOff -> SafetyResultReason.OneOff,

VotingMisinformation -> SafetyResultReason.VotingMisinformation,

HackedMaterials -> SafetyResultReason.HackedMaterials,

Scams -> SafetyResultReason.Scams,

PlatformManipulation -> SafetyResultReason.PlatformManipulation,

MisinfoCivic -> SafetyResultReason.MisinfoCivic,

MisinfoCrisis -> SafetyResultReason.MisinfoCrisis,

MisinfoGeneric -> SafetyResultReason.MisinfoGeneric,

MisinfoMedical -> SafetyResultReason.MisinfoMedical,

IpiDevelopmentOnly -> SafetyResultReason.DevelopmentOnlyPublicInterest

)

val Reasons: Set[Reason] = ReasonToSafetyResultReason.keySet

val SafetyResultReasons: Set[SafetyResultReason] = ReasonToSafetyResultReason.values.toSet

val SafetyResultReasonToReason: Map[SafetyResultReason, Reason] =

ReasonToSafetyResultReason.map(t => t.\_2 -> t.\_1)

val EligibleTweetSafetyLabelTypes: Seq[TweetSafetyLabelType] = Seq(

TweetSafetyLabelType.LowQuality,

TweetSafetyLabelType.MisinfoCivic,

TweetSafetyLabelType.MisinfoGeneric,

TweetSafetyLabelType.MisinfoMedical,

TweetSafetyLabelType.MisinfoCrisis,

TweetSafetyLabelType.IpiDevelopmentOnly

)

private val EligibleTweetSafetyLabelTypesSet = EligibleTweetSafetyLabelTypes.toSet

def extractTweetSafetyLabel(featureMap: Map[Feature[\_], \_]): Option[TweetSafetyLabel] = {

val tweetSafetyLabels = featureMap(TweetSafetyLabels)

.asInstanceOf[Seq[TweetSafetyLabel]]

.flatMap { tsl =>

if (PublicInterest.EligibleTweetSafetyLabelTypesSet.contains(tsl.labelType)) {

Some(tsl.labelType -> tsl)

} else {

None

}

}

.toMap

PublicInterest.EligibleTweetSafetyLabelTypes.flatMap(tweetSafetyLabels.get).headOption

}

def policyToReason(policy: PolicyInViolation): Reason =

policyInViolationToReason.get(policy).getOrElse(PolicyConfig.DefaultReason.\_1)

def reasonToPolicy(reason: Reason): PolicyInViolation =

reasonToPolicyInViolation.get(reason).getOrElse(PolicyConfig.DefaultPolicyInViolation)

}

class PublicInterestActionBuilder[T <: Action]() extends ActionBuilder[T] {

def actionType: Class[\_] = classOf[InterstitialLimitedEngagements]

override val actionSeverity = 11

def build(evaluationContext: EvaluationContext, featureMap: Map[Feature[\_], \_]): RuleResult = {

val (reason, limitedEngagementReason) =

PublicInterest.extractTweetSafetyLabel(featureMap).map { tweetSafetyLabel =>

(tweetSafetyLabel.labelType, tweetSafetyLabel.source)

} match {

case Some((TweetSafetyLabelType.LowQuality, source)) =>

source match {

case Some(source) =>

SafetyResultReason.valueOf(source.name) match {

case Some(matchedReason)

if PublicInterest.SafetyResultReasonToReason.contains(matchedReason) =>

(

PublicInterest.SafetyResultReasonToReason(matchedReason),

Some(LimitedEngagementReason.NonCompliant))

case \_ => PublicInterest.PolicyConfig.DefaultReason

}

case \_ => PublicInterest.PolicyConfig.DefaultReason

}

case Some((TweetSafetyLabelType.MisinfoCivic, source)) =>

(Reason.MisinfoCivic, LimitedEngagementReason.fromString(source.map(\_.name)))

case Some((TweetSafetyLabelType.MisinfoCrisis, source)) =>

(Reason.MisinfoCrisis, LimitedEngagementReason.fromString(source.map(\_.name)))

case Some((TweetSafetyLabelType.MisinfoGeneric, source)) =>

(Reason.MisinfoGeneric, LimitedEngagementReason.fromString(source.map(\_.name)))

case Some((TweetSafetyLabelType.MisinfoMedical, source)) =>

(Reason.MisinfoMedical, LimitedEngagementReason.fromString(source.map(\_.name)))

case Some((TweetSafetyLabelType.IpiDevelopmentOnly, \_)) =>

(Reason.IpiDevelopmentOnly, Some(LimitedEngagementReason.NonCompliant))

case \_ =>

PublicInterest.PolicyConfig.DefaultReason

}

RuleResult(InterstitialLimitedEngagements(reason, limitedEngagementReason), Evaluated)

}

}

class PublicInterestComplianceTweetNoticeActionBuilder

extends ActionBuilder[ComplianceTweetNoticePreEnrichment] {

override def actionType: Class[\_] = classOf[ComplianceTweetNoticePreEnrichment]

override val actionSeverity = 2

def build(evaluationContext: EvaluationContext, featureMap: Map[Feature[\_], \_]): RuleResult = {

val reason =

PublicInterest.extractTweetSafetyLabel(featureMap).map { tweetSafetyLabel =>

(tweetSafetyLabel.labelType, tweetSafetyLabel.source)

} match {

case Some((TweetSafetyLabelType.LowQuality, source)) =>

source match {

case Some(source) =>

SafetyResultReason.valueOf(source.name) match {

case Some(matchedReason)

if PublicInterest.SafetyResultReasonToReason.contains(matchedReason) =>

PublicInterest.SafetyResultReasonToReason(matchedReason)

case \_ => PublicInterest.PolicyConfig.DefaultReason.\_1

}

case \_ => PublicInterest.PolicyConfig.DefaultReason.\_1

}

case Some((TweetSafetyLabelType.MisinfoCivic, \_)) =>

Reason.MisinfoCivic

case Some((TweetSafetyLabelType.MisinfoCrisis, \_)) =>

Reason.MisinfoCrisis

case Some((TweetSafetyLabelType.MisinfoGeneric, \_)) =>

Reason.MisinfoGeneric

case Some((TweetSafetyLabelType.MisinfoMedical, \_)) =>

Reason.MisinfoMedical

case Some((TweetSafetyLabelType.IpiDevelopmentOnly, \_)) =>

Reason.IpiDevelopmentOnly

case \_ =>

PublicInterest.PolicyConfig.DefaultReason.\_1

}

RuleResult(

ComplianceTweetNoticePreEnrichment(reason, ComplianceTweetNoticeEventType.PublicInterest),

Evaluated)

}

}

class PublicInterestDropActionBuilder extends ActionBuilder[Drop] {

override def actionType: Class[\_] = classOf[Drop]

override val actionSeverity = 16

private def toRuleResult: Reason => RuleResult = Memoize { r => RuleResult(Drop(r), Evaluated) }

def build(evaluationContext: EvaluationContext, featureMap: Map[Feature[\_], \_]): RuleResult = {

val reason = PublicInterest.extractTweetSafetyLabel(featureMap).map(\_.labelType) match {

case Some(TweetSafetyLabelType.LowQuality) =>

Reason.OneOff

case Some(TweetSafetyLabelType.MisinfoCivic) =>

Reason.MisinfoCivic

case Some(TweetSafetyLabelType.MisinfoCrisis) =>

Reason.MisinfoCrisis

case Some(TweetSafetyLabelType.MisinfoGeneric) =>

Reason.MisinfoGeneric

case Some(TweetSafetyLabelType.MisinfoMedical) =>

Reason.MisinfoMedical

case \_ =>

Reason.OneOff

}

toRuleResult(reason)

}

}

object PublicInterestRules {

object AbusePolicyEpisodicTweetLabelInterstitialRule

extends Rule(

new PublicInterestActionBuilder(),

And(

TweetComposedAfter(PublicInterest.PolicyConfig.LowQualityProxyLabelStart),

Or(

PublicInterest.EligibleTweetSafetyLabelTypes.map(TweetHasLabel(\_)): \_\*

)

)

)

object AbusePolicyEpisodicTweetLabelComplianceTweetNoticeRule

extends Rule(

new PublicInterestComplianceTweetNoticeActionBuilder(),

And(

TweetComposedAfter(PublicInterest.PolicyConfig.LowQualityProxyLabelStart),

Or(

PublicInterest.EligibleTweetSafetyLabelTypes.map(TweetHasLabel(\_)): \_\*

)

)

)

object AbusePolicyEpisodicTweetLabelDropRule

extends Rule(

new PublicInterestDropActionBuilder(),

And(

TweetComposedAfter(PublicInterest.PolicyConfig.LowQualityProxyLabelStart),

Or(

PublicInterest.EligibleTweetSafetyLabelTypes.map(TweetHasLabel(\_)): \_\*

)

)

)

object SearchIpiSafeSearchWithoutUserInQueryDropRule

extends Rule(

new PublicInterestDropActionBuilder(),

And(

TweetComposedAfter(PublicInterest.PolicyConfig.LowQualityProxyLabelStart),

Or(

PublicInterest.EligibleTweetSafetyLabelTypes.map(TweetHasLabel(\_)): \_\*

),

LoggedOutOrViewerOptInFiltering,

Not(SearchQueryHasUser)

)

) {

override def enabled: Seq[RuleParam[Boolean]] = Seq(

EnableSearchIpiSafeSearchWithoutUserInQueryDropRule)

}

}