Table of Contents

[1 Overview 2](#_Toc511904404)

[2 Eligibility Policy Violation 2](#_Toc511904405)

[3 Multiple Accounts Policy Type Options 2](#_Toc511904406)

[3.1 Required Custom Artifact 2](#_Toc511904407)

[3.2 Optional Custom Artifact 3](#_Toc511904408)

[4 Soft Stop / Allow Access Request 4](#_Toc511904409)

[4.1 Application 4](#_Toc511904410)

[4.2 Multiple Applications 4](#_Toc511904411)

[5 Hard Stop / Deny Access Request 4](#_Toc511904412)

[5.1 Application 4](#_Toc511904413)

[5.2 Multiple Applications 4](#_Toc511904414)

[6 Enable Preventive Checks 5](#_Toc511904415)

[7 SOD Matrix Report 5](#_Toc511904416)

[8 Data Flow 5](#_Toc511904417)

.

# Overview

The purpose of this document is to explain the Policy Violation Feature that is bundled with the Accelerator Pack. This feature is focused only on “**Advanced Policy**”. This feature supports **multiple accounts** on an application for both prevention and detection. Out of the box policy violations are configured as usual. However, these OOTB configured policy violations can be used for hard/soft access requests. There is a “SOD Matrix Report” packaged with this feature that shows ONLY an “Advanced Policy” SOD matrix. These advanced policies must be configured as instructed below.

# Eligibility Policy Violation

This option is configured via the Self Service Onboarding Wizard or IdentityIQ Administrative Wizard. The name of the attribute that stores this configuration on an application object is “**eligibilityExpression**”. This is usually used for applications that have request-based eligibility criteria based on identity profile attributes. The supported format for the Self Service Onboarding Wizard is and IdentityIQ Population aka **GroupDefinition** **name**. The supported format for the IdentityIQ Administrative Wizard is String Format with Java Regular Expression.

Example of String Format with Java Regular Expression “**[Cube Attribute Name]#IIQPolicyEligibility#[Java Regular Expression]**”

|  |  |
| --- | --- |
| String Tokens and Separators | Description |
| [Cube Attribute Name] | Application Schema Attribute Name |
| #IIQPolicyEligibility# (Token Separator) | Token to Separate Identity Cube Attribute Name and Java Regular Expression |
| [Java Regular Expression] | This is executed against [Cube Attribute Name] value. |

A new policy and advanced rule needs to be configured to turn on the eligibility policy for an application. A Java method “**isEligibleViolation**” needs to be called from the rule to run the execute eligibility criteria. Here is an example of a code-snippet that needs to be defined in the advanced policy rule per application.

**Line 1:** import sailpoint.rapidapponboarding.rule.PolicyViolationsRuleLibrary;

**Line 2:** return PolicyViolationsRuleLibrary.isEligibleViolation(context, identity, "[Name of Application]",[Policy Violation Name], [Violation Description]);

# Multiple Accounts Policy Type Options

There are two types of menu options for an application for Advanced SOD on multiple accounts

* Single Entitlement: This means only one entitlement is allowed for an account on an application
  + Also, this policy type supports exceptions, and that means some combination of entitlements can be allowed for the access request
* Multiple Entitlement with Toxic Combinations: This means some combination of entitlements are considered to be toxic for an account

The name of the attribute that stores this configuration on an application object is “**policyType**”.

## Required Custom Artifact

Multiple Entitlement with Toxic Combinations Type **requires** a custom artifact that will store a combination of entitlements. The custom artifact name must be defined on the application.

* Toxic Combinations Custom Name: The name of the custom artifact that will be used for the Single Entitlement Option
* A new custom artifact, policy and advanced rule need to be configured to turn on the Multiple Entitlement with Toxic Combinations option for an application. A Java method “**detectPolicyViolation**” needs to be called from the rule to run the business logic for the Multiple Entitlement with Toxic Combinations Option. Here is an example of code-snippet that needs to be defined in the advanced policy rule per application.

**Line 1:** import sailpoint.rapidapponboarding.rule.PolicyViolationsRuleLibrary;

**Line 2:** return PolicyViolationsRuleLibrary.detectPolicyViolation(context,"[Application Name]", identity);

* Required Custom Artifact Format. The name of the attribute that stores this custom artifact name on an application object is “**txCombinationName**”. Here is an example of a custom artifact format for an application

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE Custom PUBLIC "sailpoint.dtd" "sailpoint.dtd">

<Custom name="Custom-SOD-Incompatibilities-Application-A">

    <Attributes>

        <Map>

            <entry key="ViolationsMap">

                <value>

                    <Map>

                    <!-- Toxic Combinations Not Allowed -->

                        <entry key="1" value="ENT 1,ENT 2"/>

                        <entry key="2" value="ENT 2,ENT 3"/>

                    </Map>

                </value>

            </entry>

        </Map>

    </Attributes>

</Custom>

## Optional Custom Artifact

The Single Entitlement Option provides an option where some entitlements can be exempted from the policy. These entitlements are stored in a custom artifact. The custom artifact name must be defined on the application.

* Single Entitlement Exceptions Custom Name: The name of the custom artifact that will be used for Single Entitlement Option Exception. This artifact is used for any exceptions for single entitlement option.
* An optional custom artifact, new policy and advanced rule need to be configured to turn on above Single Entitlement option for an application. A “**detectSingleEntitlementPolicyViolation**” Java method needs to be called from the rule to run the business logic for the Single Entitlement Option. Here is an example of code-snippet that needs to be defined in the advanced policy rule per application.

**Line 1:** import sailpoint.rapidapponboarding.rule.PolicyViolationsRuleLibrary;

**Line 2:** returnPolicyViolationsRuleLibrary.detectSingleEntitlementPolicyViolation(context,"[Application Name]", identity);

* Optional Custom Artifact Format for Exceptions. The name of the attribute that stores this custom artifact name on an application object is “**policytypeExceptions**”. Here is an example of custom artifact format for an application

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE Custom PUBLIC "sailpoint.dtd" "sailpoint.dtd">

<Custom name="Custom-PolicyViolation-SingleEntitlement-Exceptions-Application-B">

<Attributes>

<Map>

<entry key="singleEntitlementMap" >

<value>

<Map>

                       <entry key="exceptions" value="ENT 9,ENT 8"/>

                   </Map>

</value>

</entry>

</Map>

</Attributes>

</Custom>

# Soft Stop / Allow Access Request

This is when requestor is allowed is to submit an access request with violations. The requestor is required to enter comments for the request submission on preventive policy violations.

## Application

This is configured on the application. Multiple policies are separated by comma. The name of the indexed extended attribute used for hard stop policy violation is “softPolicyviolations”.

## Multiple Applications

Soft Stop Policies that are shared between applications can be configured in the “Custom-PolicyViolation-AllowDenyExceptions” custom artifact. They are configured with the value “Allow Exception” in the custom artifact or via a QuickLink that is used to modify Accelerator Pack “Global Definition”. Here is an example.

<Attributes>

<Map>

<entry key="[Policy Name]" value="Allow Exception"/>

</Map>

</Attributes>

# Hard Stop / Deny Access Request

This is when the requestor is not allowed is to submit access requests with violations. Hard Stop policy always wins over Soft stop policy during access request. This means if a request has both hard and soft stop preventive policy violations, the requestor will not be allowed to submit the request.

## Application

This is configured on the application. Multiple policies are separated by comma. The name of the indexed extended attribute used for hard stop policy violation is “hardPolicyviolations”.

## Multiple Applications

Hard Stop Policies that are shared between applications can be configured in the “Custom-PolicyViolation-AllowDenyExceptions” custom artifact. They are configured with the value “Deny Exception” in the custom artifact or via a QuickLink that is used to modify Accelerator Pack “Global Definition”. Here is an example.

<Attributes>

<Map>

<entry key="[Policy Name]" value="Deny Exception"/>

</Map>

</Attributes>

# Enable Preventive Checks

There is one global setting that is used to turn on interactive preventive policy violation checks. This setting can be configured via a QuickLink that is used to modify Accelerator Pack   
“Global Definition” or can be configured in a “Custom-Framework-Common-Settings” custom artifact.

|  |  |
| --- | --- |
| Token | Description |
| policyScheme | This option is to turn on/off Preventive SOD checks. The value “none” is to turn off and the value “interactive” is to turn on. All policy violations defined on the **requested** applications for hard/soft stops and policies defined in the “**Custom-PolicyViolation-AllowDenyExceptions**” custom artifact will be interrogated for preventive checks. |

# SOD Matrix Report

This report is based only on policy type options that are offered Policy Violation Feature. This report is only for Advanced Policies that uses **detectSingleEntitlementPolicyViolation and detectPolicyViolation** method in their rules.

* Single Entitlement
* Multiple Entitlement with Toxic Combinations

# Data Flow

The preventive policy violations are generated either from a shopping cart request or batch request. Here is the data flow diagram.

