

Access Level Condition Expression Rewrite Guide

Background

Use of condition attributes on unsupported permissions in your IAM policies will begin blocking access. To avoid any surprising effects from this change, the currently configured access level condition expressions may need to be rewritten.

This document provides rewriting guidance on access level condition expressions with representative examples and suggested actions.

Access Level Usage Instructions

For Identity Access Proxy usage, the following are the two supported permissions. No other permissions, including other Identity Access Proxy permissions, support access level conditions.

iap.webServiceVersions.accessViaIAP permission
iap.tunnelInstances.accessViaIAP permission

Please refer to the [Attribute reference for IAM Conditions/Access level attribute](#) documentation for detailed instructions on using access level condition with supported permissions only.

Valid Access Level Strings

To check if the string used with request.auth.access_levels attribute in condition expression is valid, see [List Access Levels](#) for how to list all the configured access levels for your organization. Any string value outside the returned list of strings is invalid, including strings that are not correctly formatted, not defined for the organization, or using an incorrect access policy identifier.

Example Cases

Case 1

Example Expression	<code>!(<string literal> in request.auth.access_levels)</code> where <string literal> is an invalid access level string .
Description	Note the negation. Because it is an invalid access level string, this is equivalent to unconditional binding and always grants access
Suggested actions	Two options: 1. If this is a typo, fix the string to be a valid access level string, proceed to case #3 below. 2. Remove this sub-expression from the condition, or remove

	condition from the binding, if it's the only expression.
--	--

Case 2

Example Expression	<code><string literal> in request.auth.access_levels</code> where <string literal> is an invalid access level string .
Description	This expression will never grant access
Suggested actions	<i>Note: Change of Cloud IAM condition evaluation behavior does not impact the scope of this binding. However, it's a good practice to clean up.</i> Two options: 1. If this is a typo, fix the string to be a valid access level string, proceed to case #4 below. 2. Rewrite this portion as if it's a constant <code>FALSE</code>, or remove the binding, if it's the only expression.

Case 3

Example Expression	<code>!(<string literal> in request.auth.access_levels)</code> where <string literal> is a valid access level string .
Description	For service permissions supporting the attribute, this expression will grant access if request does not qualify for <string literal> access level. For service permissions NOT supporting the attribute, the behavior will change. From potentially granting access to not granting access when Cloud IAM changes the condition evaluation behavior.
Suggested actions	To keep the intended grant scope, use <code>resource.service</code> and/or <code>resource.type</code> to limit the scope of access level restriction to the intended usage. For instance, the following limits the scope to Identity Access Proxy tunnel instances only. The rest of the scope is not restricted by the access level condition. <pre>(! <string literal> in request.auth.access_levels) resource.type != "iap.googleapis.com/TunnelInstance"</pre>

Case 4

Condition	<code><string literal> in request.auth.access_levels</code> where <code><string literal></code> is a valid access level string .
Description	For service permissions in the corresponding role supporting the attribute, this expression will grant access with intended functionality For service permissions in the corresponding role NOT supporting the attribute, this expression will not grant access, which is the intended functionality. <i>Note: As access level conditions are supported by more services in the future, the same role with permissions previously not supported will become supported. In this case, your conditional binding with intended restriction scope may change.</i>
Suggested actions	The following are the suggested actions to take that explicitly scope the access level condition restriction to the desirable scope according to the current supported service permissions. This way, future changes to the scope of supported services for access level condition will have little or no impact on your existing conditional bindings. For instance, the following limits the scope to Identity Access Proxy tunnel instances only. The rest of the scope is not restricted by the access level condition. <pre>(! <string literal> in request.auth.access_levels) resource.type != "iap.googleapis.com/TunnelInstance"</pre>

Case 5

Example Expression	<pre><string literal1> in request.auth.access_levels <string literal2> in request.auth.access_levels !<string literal3> in request.auth.access_levels</pre>
Description	Condition consists of a number of <code><string literal></code> in <code>request.auth.access_levels</code>, and logically combined together This is an extended case from simpler cases 1-4. See above for detailed description
Suggested actions	For each access level condition sub-expression, take suggested actions to clean up or rewrite the expression, as discussed for case 1 - 4. If appropriate, you could choose to group them together when adding resource service or type scoping.

	For instance, the following limits the scope to Identity Access Proxy tunnel instances only. The rest of the scope is not restricted by the access level condition. <pre>(<string literal1> in request.auth.access_levels <string literal2> in request.auth.access_levels !<string literal3> in request.auth.access_levels) resource.type != "iap.googleapis.com/TunnelInstance"</pre>
--	--

Case 6

Example Expression	<code><string literal> in request.auth.access_levels</code>
Example Role	<code>roles/iap.admin</code>
Description	Access level condition set on the role that does not contain any supported permissions. This is an effective FALSE condition.
Suggested actions	1) Remove the conditional binding or 2) Remove the condition, to make it unconditional. The choice between #1 and #2 depends on the intended usage of this binding.

Case 7

Example Expression	<code><string literal> in request.auth.access_levels</code>
Example Role	<code>roles/iam.securityAdmin</code>
Description	Access level condition set on a role with mixed permissions, but cannot be type guarded via resource service or resource type. For instance, iam.securityAdmin role contains iam.roles.get list and iam.serviceAccounts.get permissions, which are all in scope. The same role also contains *.setIamPolicy permissions, in which some are supported, but most are not supported.
Suggested actions	1) Remove the conditional binding or 2) Remove the condition, to make it unconditional, or 3) Add type guard to limit it to only to specific supported service or resource type.

	The choice between #1, #2, or #3 depends on the intended usage of this binding. #3 rewrite example: <pre><string literal> in request.auth.access_levels resource.service != 'iam.googleapis.com'</pre> The updated expression limit the Access Level restriction on the following permissions: <pre>'iam.serviceAccounts.get' 'iam.roles.get' 'iam.roles.list' 'iam.*.setIamPolicy' 'iam.*.getIamPolicy'</pre>
--	--

Case 8

Example Expression	<code><string literal> in request.auth.access_levels</code>
Example Role	<code>roles/iap.tunnelAccessor</code>
Description	Access level condition set on a role with all permissions supporting access level condition.
Suggested actions	1) No action, or 2) Add type guard to restrict to the related resource types or services. The option #2 may be triggered, if the involved role is a custom role, the included permissions may be subject to change. To avoid any surprising effect on the policy, it is best to restrict the scope of access level conditions explicitly.