This document is published in the IAM User Guide: <https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html>

# Policy Evaluation Logic

When a principal tries to use the AWS Management Console, the AWS API, or the AWS CLI, that principal sends a request to AWS. When an AWS service receives the request, AWS completes several steps to determine whether to allow or deny the request.

1. **Authentication** – AWS first authenticates the principal that makes the request, if necessary. This step is not necessary for a few services, such as Amazon S3, that allow some requests from anonymous users.
2. [**Processing the Request Context**](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html#policy-eval-reqcontext) – AWS processes the information gathered in the request to determine which policies apply to the request.
3. [**Evaluating Policies**](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html#policy-eval-basics) – AWS evaluates all of the policy types, which affect the order in which the policies are evaluated.
4. [**Determining Whether a Request Is Allowed or Denied**](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html#policy-eval-denyallow) – AWS then processes the policies against the request context to determine whether the request is allowed or denied.

## Processing the Request Context

AWS processes the request to gather the following information into a request context:

* **Actions (or Operations)** – The actions or operations that the principal wants to perform.
* **Resources** – The AWS resource object upon which the actions or operations are performed.
* **Principal** – The user, role, federated user, or application that sent the request. Information about the principal includes the policies that are associated with that principal.
* **Environment data** – Information about the IP address, user agent, SSL enabled status, or the time of day.
* **Resource data** – Data related to the resource that is being requested. This can include information such as a DynamoDB table name or a tag on an Amazon EC2 instance.

AWS then uses this information to find policies that apply to the request context.

## Evaluating Policies

AWS evaluates the policies that apply to a request context based on the policy type and category. AWS supports identity-based policies, resource-based policies, AWS Organizations SCPs, and access control lists (ACLs). Those [policies types](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#access_policy-types) can be categorized as permissions policies or permissions boundaries. Permissions policies control the permissions for the object to which they’re attached. These include identity-based policies (most common), resource-based policies, and ACLs. A permissions boundary is an advanced feature that allows you to use policies to limit the maximum permissions that a principal can have. These boundaries can be applied to AWS Organizations organizations or to IAM users or roles. You can further reduce the permissions for a role session by passing a policy [during AWS STS role assumption](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-api.html).

If no permissions boundary or STS assume role policy exists, then the permissions policy alone controls access. If more than one of these types exists, then the access is determined by the intersection of all of the applicable policy types. For example, consider a role with a permissions boundary that allows all Amazon EC2 and CloudWatch actions. That role also has a permissions policy that allows only the ec2:StartInstances, ec2:StopInstances, and s3:ListBucketactions. During AWS STS role assumption, an administrator passes a permissions policy that allows starting and stopping only the MyCompanyInstance Amazon EC2 instance and listing the MyCompanyBucket Amazon S3 bucket. When AWS evaluates the three policy types, the resulting access is the intersection of the three policy types. In this case, anyone assuming the role can only start and stop the MyCompanyInstance Amazon EC2 instance. They cannot perform the actions that appear in only one or two policy types.

AWS evaluates the policies used as permissions boundaries or passed during STS role assumption, if they apply, before evaluating the permissions policies.



When AWS evaluates the permissions policies and permissions boundary for a user, the resulting permissions are the intersection of the two categories. That means that when you add a permissions boundary to a user with existing permissions policies, you might reduce the actions that the user can perform. Alternatively, when you remove a permissions boundary from a user, you might increase the actions they can perform.



## Determining Whether a Request Is Allowed or Denied

When a principal sends a request to AWS, the AWS enforcement code decides whether a given request should be allowed or denied. The following is a high-level summary of the AWS evaluation logic:

* By default, all requests are denied. (In general, requests made using the AWS account root user credentials for resources in the account are always allowed.)
* An explicit allow in a permissions policy overrides this default.
* A permissions boundary (an AWS Organizations SCP or a user or role boundary) or a policy used during AWS STS role assumption overrides the allow. If one or more of these items exists, they must all allow the request. Otherwise, it is implicitly denied.
* An explicit deny in any policy overrides any allows.

The following flow chart provides details about how the decision is made.



1. **Deny evaluation.** The decision starts with the position that all requests are denied by default. This is called an implicit deny. For more information, see [The Difference Between Explicit and Implicit Denies](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html#AccessPolicyLanguage_Interplay).
2. The enforcement code evaluates all policies within the account that apply to the request. These include permissions policies (identity-based, resource-based, and ACLs) and permissions boundaries (AWS Organizations SCPs, user or role boundaries, and STS assume role boundaries). All policy elements are considered. The order in which the enforcement code evaluates the policies for an explicit deny is not important.
3. In all those policies, the enforcement code looks for a Deny statement that applies to the request. If the code finds even one explicit deny that applies, the code returns a decision of **Deny** and the process is finished. This is called an explicit deny. For more information, see [The Difference Between Explicit and Implicit Denies](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html#AccessPolicyLanguage_Interplay).
4. **Organizations and boundaries.** If no explicit deny is found, the code begins evaluating the AWS Organizations permissions boundaries defined by the service control policies (SCPs). SCPs apply if the principal making the request is a member of an account that is a member of that organization.
5. If the enforcement code does not find any Allow statements in the applicable SCPs, then the request is implicitly denied. The code returns a decision of **Deny** and the process is finished.
6. **User or role permissions boundaries.** If the code finds even one Allow statement in the applicable SCPs, or if there are no applicable SCPs, then the enforcement code continues. It then checks permissions boundaries for the applicable user or role.
7. If the enforcement code does not find any Allow statements in the applicable policies, then the request is implicitly denied. The code returns a decision of **Deny** and the process is finished.
8. **STS assume role policies.** If the code finds even one Allow statement in the user or role boundaries, or if there is no applicable user or role boundary, then the enforcement code continues. It then checks for a role that is assumed using AWS STS.
9. If the enforcement code does not find any Allow statements in the applicable policy, then the request is implicitly denied. The code returns a decision of **Deny** and the process is finished.
10. **Permissions.** If the code finds even one Allow statement in the policy passed during role assumption, or if there is no applicable policy, then the enforcement code continues. It then evaluates all the permissions policies (identity-based, resource-based, and ACLs). Applicable permissions policies are checked together and order does not matter.
11. If the code finds even one Allow statement in the applicable permissions policies, then the enforcement code returns a decision of **Allow** and the process is finished. If there are no Allow statements, then the request is implicitly denied. The code returns a decision of **Deny**, and the process is finished.

If the enforcement code encounters an error at any point during the evaluation, then it generates an exception and closes.

## Example Evaluation

The most common types of policies are identity-based policies and resource-based policies.

Assume that Carlos has the user name carlossalazar and he tries to save a file to the carlossalazar-logs Amazon S3 bucket.

Also assume that the following policy is attached to the carlossalazar IAM user.

{

 "Version": "2012-10-17",

 "Statement": [

 {

 "Sid": "AllowS3ListRead",

 "Effect": "Allow",

 "Action": [

 "s3:ListAllMyBuckets",

 "s3:HeadBucket"

 ],

 "Resource": "\*"

 },

 {

 "Sid": "AllowS3Self",

 "Effect": "Allow",

 "Action": "s3:\*",

 "Resource": [

 "arn:aws:s3:::carlossalazar/\*",

 "arn:aws:s3:::carlossalazar"

 ]

 },

 {

 "Sid": "DenyS3Logs",

 "Effect": "Deny",

 "Action": "s3:\*",

 "Resource": [

 "arn:aws:s3:::\*log\*",

 "arn:aws:s3:::\*log\*/\*"

 ]

 }

 ]

}

The AllowS3ListRead statement in this policy allows Carlos to view a list of all of the buckets in the account. The AllowS3Self statement allows Carlos full access to the bucket with the same name as his user name. The DenyS3Logsstatement denies Carlos access to any S3 bucket with log in its name.

Additionally, the following resource-based policy (called a bucket policy) is attached to the carlossalazar bucket.

{

 "Version": "2012-10-17",

 "Statement": [

 {

 "Effect": "Allow",

 "Action": "s3:\*",

 "Principal": { "AWS": "arn:aws:iam::111122223333:user/carlossalazar" },

 "Resource": "\*"

 }

 ]

}

This policy specifies that only the carlossalazar user can access the carlossalazar bucket.

When Carlos makes his request to save a file to the carlossalazar-logs bucket, AWS determines what policies apply to the request. In this case, only the identity-based policy and the resource-based policy apply. These are both permissions policies. Because no permissions boundaries apply, the evaluation logic is reduced to the following logic.



AWS first checks for a Deny statement that applies to the context of the request. It finds one, because the identity-based policy explicitly denies Carlos access to any S3 buckets used for logging. Carlos is denied access.

Assume that he then realizes his mistake and tries to save the file to the carlossalazar bucket. AWS checks for a Denystatement and does not find one. It then checks the permissions policies. The identity-based policy allows the request. Therefore, AWS allows the request. If either of them explicitly denied the statement, the request would have been denied.

## The Difference Between Explicit and Implicit Denies

A request results in an explicit deny if an applicable policy includes a Deny statement. If policies that apply to a request include an Allow statement and a Deny statement, the Deny statement trumps the Allow statement. The request is explicitly denied.

An implicit denial occurs when there is no applicable Deny statement but also no applicable Allow statement. Because an IAM user, role, or federated user is denied access by default, they must be explicitly allowed to perform an action. Otherwise, they are implicitly denied access.

When you design your authorization strategy, you must create policies with Allow statements to allow your principals to successfully make requests. However, you can choose any combination of explicit and implicit denies. For example, you can create the following policy to allow an administrator user full access to all resources in AWS, but explicitly deny access to billing. If someone adds another policy to this administrator user granting them access to billing, it is still denied because of this explicit deny.

{

 "Version": "2012-10-17",

 "Statement": [

 {

 "Effect": "Allow",

 "Action": "\*",

 "Resource": "\*"

 },

 {

 "Effect": "Deny",

 "Action": "aws-portal:\*",

 "Resource": "\*"

 }

 ]

}

Alternatively, you can create the following policy to allow a user to manage users, but not groups or any other resources in IAM. Those actions are implicitly denied, as are actions in other services. However, if someone adds a policy to the user that allows them to perform these other actions, then they are allowed.

{

 "Version": "2012-10-17",

 "Statement": {

 "Effect": "Allow",

 "Action": [

 "iam:AttachUserPolicy",

 "iam:CreateUser",

 "iam:DeleteUser",

 "iam:DeleteUserPolicy",

 "iam:DetachUserPolicy",

 "iam:GetUser",

 "iam:GetUserPolicy",

 "iam:ListAttachedUserPolicies",

 "iam:ListUserPolicies",

 "iam:ListUsers",

 "iam:PutUserPolicy",

 "iam:UpdateUser"

 ],

 "Resource": "\*"

 }

}