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 (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 Within a Single Account (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 Within an Account (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 Within a Single Account

How AWS evaluates policies depends on the types of policies that apply to the request context. The following policy types, listed in order of frequency, are available for use within a single AWS account. For more information about these policy types, see Policies and Permissions (access_policies.html) . To learn about cross-account authorization, see How IAM Roles Differ from Resource-based Policies (id_roles_compare-resource-policies.html) .

1. **Identity-based policies** – Identity-based policies are attached to an IAM identity (user, group of users, or role) and grant permissions to IAM entities (users and roles). If only identity-based policies apply to a request, then AWS checks all of those policies for at least one `Allow`.

2. **Resource-based policies** – Resource-based policies grant permissions to the principal entity (account, user, role, or federated user) specified as the principal. The permissions define what the principal can do with the resource to which the policy is attached. If resource-based policies and identity-based policies both apply to a request, then AWS checks all the policies for at least one `Allow`.

3. **IAM permissions boundaries** – Permissions boundaries are an advanced feature that sets the maximum permissions that an identity-based policy can grant to an IAM entity (user or role). When you set a permissions boundary for an entity, the entity can perform only the actions that are allowed by both its identity-based policies and its permissions boundaries. Permissions boundaries do not affect the permissions granted by a resource-based policy.

4. **AWS Organizations service control policies (SCPs)** – Organizations SCPs specify the maximum permissions for an organization or organizational unit (OU). The SCP maximum applies to entities in member accounts, including each AWS account root user. If an SCP is present, identity-based and resource-based policies grant permissions to entities only if those policies and the SCP allow the action. If both a permissions boundary and an SCP are present, then the boundary, the SCP, and the identity-based policy must all allow the action.

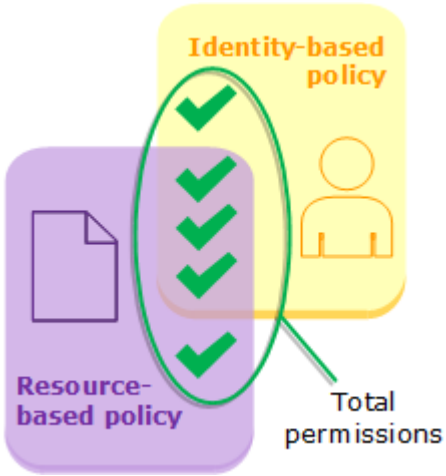
5. **Session policies** – Session policies are advanced policies that you pass as parameters when you programmatically create a temporary session for a role or federated user. To create a role session programmatically, use one of the `AssumeRole*` API operations. When you do this and pass session policies, the resulting session's permissions are the intersection of the IAM entity's identity-based policy and the session policies. To create a federated user session, you use an IAM user's access keys to programmatically call the `GetFederationToken` API operation. A resource-based policy has a different effect on the evaluation of session policy permissions. The difference depends on whether the user or role's ARN or the session's ARN is listed as

the principal in the resource-based policy. For more information, see Session Policies (access_policies.html#policies_session) .

Remember, an explicit deny in any of these policies overrides the allow.

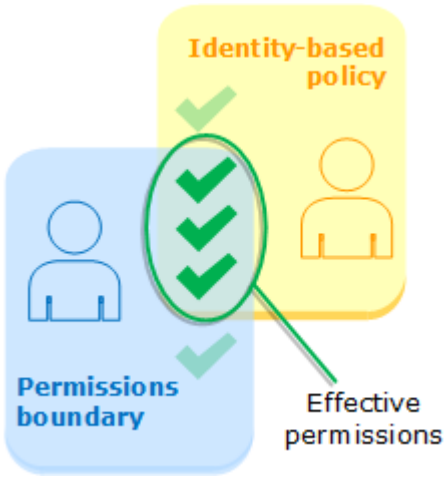
Evaluating Identity-Based Policies with Resource-Based Policies

Identity-based policies and resource-based policies grant permissions to the identities or resources to which they are attached. When an IAM entity (user or role) requests access to a resource within the same account, AWS evaluates all the permissions granted by the identity-based and resource-based policies. The resulting permissions are the total permissions of the two types. If an action is allowed by an identity-based policy, a resource-based policy, or both, then AWS allows the action. An explicit deny in either of these policies overrides the allow.



Evaluating Identity-Based Policies with Permissions Boundaries

When AWS evaluates the identity-based 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 identity-based 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. An explicit deny in either of these policies overrides the allow. To view information about how other policy types are evaluated with permissions boundaries, see Evaluating Effective Permissions with Boundaries (access_policies_boundaries.html#access_policies_boundaries-eval-logic) .



Evaluating Identity-Based Policies with Organizations SCPs

When a user belongs to an account that is a member of an organization, the resulting permissions are the intersection of the user's policies and the SCP. This means that an action must be allowed by both the identity-based policy and the SCP. An explicit deny in either of these policies overrides the allow.

You can learn whether your account is a member of an organization (https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_org_details.html#orgs_viev in AWS Organizations. Organization members might be affected by an SCP. To view this data using the AWS CLI command or AWS API operation, you must have permissions for the `organizations:DescribeOrganization` action for your Organizations entity. You must have additional permissions to perform the operation in the Organizations console. To learn whether an SCP is denying access to a specific request, or to change your effective permissions, contact your AWS Organizations administrator.

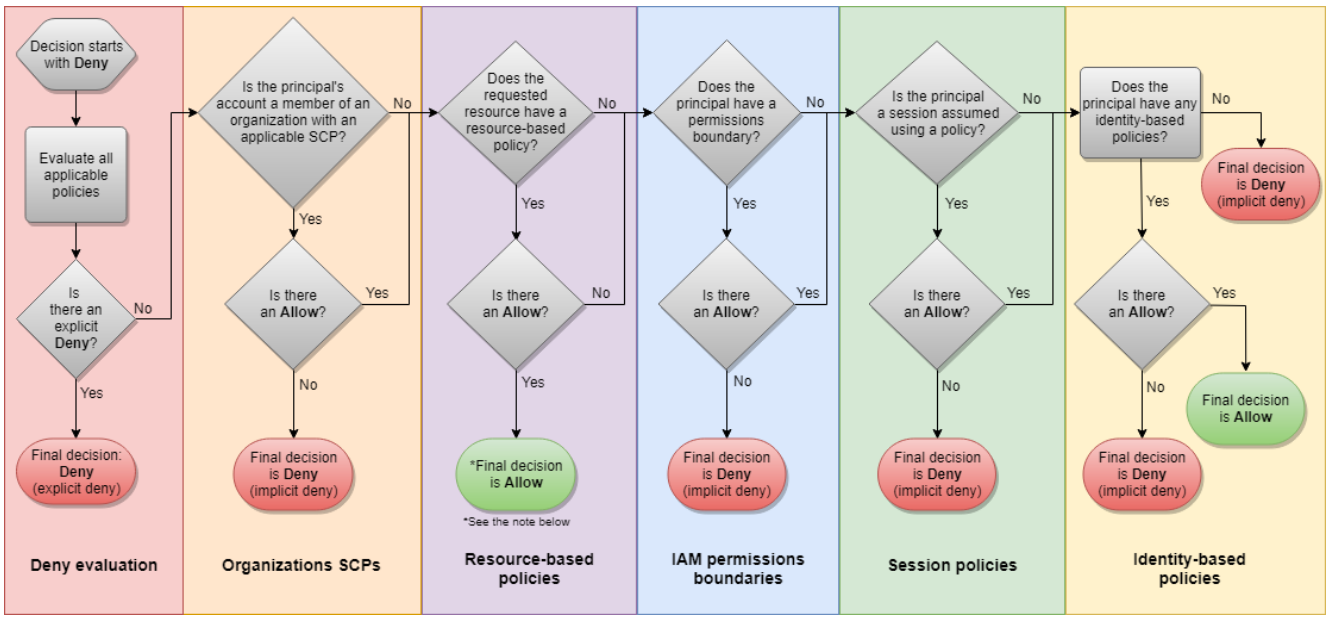
Determining Whether a Request Is Allowed or Denied Within an Account

Assume that a principal sends a request to AWS to access a resource in the same account as the principal's entity. The AWS enforcement code decides whether the request should be allowed or denied. AWS gathers all of the policies that apply to the request context. The

following is a high-level summary of the AWS evaluation logic on those policies within a single account.

- By default, all requests are implicitly denied. (Alternatively, by default, the AWS account root user has full access.)
- An explicit allow in an identity-based or resource-based policy overrides this default.
- If a permissions boundary, Organizations SCP, or session policy is present, it might override the allow with an implicit deny.
- An explicit deny in any policy overrides any allows.

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



1. **Deny evaluation** – By default, all requests are denied. This is called an implicit deny (reference_policies_evaluation-logic.html#AccessPolicyLanguage_Interplay) . The AWS enforcement code evaluates all policies within the account that apply to the request. These include AWS Organizations SCPs, resource-based policies, IAM permissions boundaries, role session policies, and identity-based policies. In all those policies, the enforcement code looks for a `Deny` statement that applies to the request. This is called an explicit deny (reference_policies_evaluation-logic.html#AccessPolicyLanguage_Interplay) . If the code finds even one explicit deny that applies, the code returns a final decision of **Deny**. If there is no explicit deny, the code continues.

2. **Organizations SCPs** – Then the code evaluates AWS Organizations service control policies (SCPs) that apply to the request. SCPs apply if the request is made in an account to which the SCP is attached. If the enforcement code does not find any applicable `Allow` statements in the SCPs, then the request is implicitly denied. The code returns a final decision of **Deny**. If there is no SCP, or if the SCP allows the requested action, the code continues.

3. **Resource-based policies** – If the requested resource has a resource-based policy that allows the principal entity to perform the requested action, then the code returns a final decision of **Allow**. If there is no resource-based policy, or if the policy does not include an `Allow` statement, then the code continues.

   Note

   This logic can behave differently if you specify the ARN of an IAM role or user as the principal of the resource-based policy. Someone can use session policies to create a temporary credential session for that role or federated user. In that case, the effective permissions for the session might not exceed those allowed by the identity-based policy of the user or role. For more information, see Session Policies (https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#policies_session .

4. **IAM permissions boundaries** – The enforcement code then checks whether the IAM entity that is used by the principal has a permissions boundary. If the policy that is used to set the permissions boundary does not allow the requested action, then the request is implicitly denied. The code returns a final decision of **Deny**. If there is no permissions boundary, or if the permissions boundary allows the requested action, the code continues.

5. **Session policies** – The code then checks whether the principal entity is using a session that was assumed by passing a session policy. You can pass a session policy while using the AWS CLI or AWS API to get temporary credentials for a role or federated user. If the session policy is present and does not allow the requested action, then the request is implicitly denied. The code returns a final decision of **Deny**. If there is no session policy, or if the policy allows the requested action, the code continues.

6. **Identity-based policies** – The code then checks the identity-based policies for the principal entity. For an IAM user, these include user policies and policies from groups to which the user belongs. If any statement in any applicable identity-based policies allows the requested action, then the enforcement code returns a final decision of **Allow**. If there are no statements that allow the requested action, then the request is implicitly denied, and the code returns a final decision of **Deny**.

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

Example Identity-Based and Resource-Based Policy 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 `DenyS3Logs` statement denies Carlos access to any S3 bucket with `log` in its name.

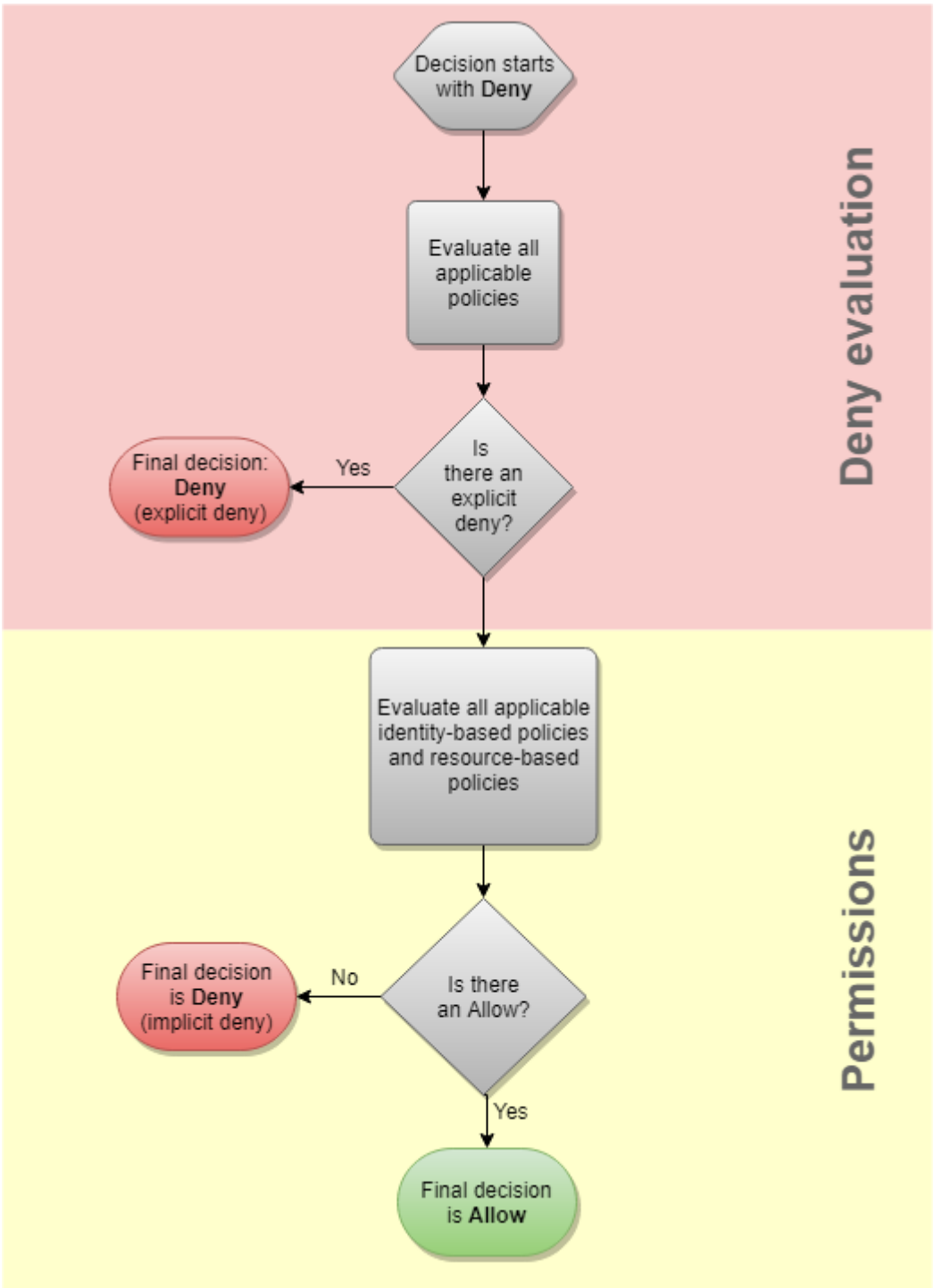
Additionally, the following resource-based policy (called a bucket policy) is attached to the `carlossalazar` bucket.

```
{
    "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 `Deny` statement 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 full access to all resources in AWS, but explicitly deny access to billing. If someone adds another policy to this administrator 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": "*"
    }
}
```