Permissions Boundaries for IAM Entities

AWS supports *permissions boundaries* for IAM entities (users or roles). A permissions boundary is an advanced feature for using a managed policy to set the maximum permissions that an identity-based policy can grant to an IAM entity. An entity's permissions boundary allows it to perform only the actions that are allowed by both its identity-based policies and its permissions boundaries.

For more information about policy types, see Policy Types (access_policies.html#access_policy-types) .

You can use an AWS managed policy or a customer managed policy to set the boundary for an IAM entity (user or role). That policy limits the maximum permissions for the user or role.

For example, assume that the IAM user named `ShirleyRodriguez` should be allowed to manage only Amazon S3, Amazon CloudWatch, and Amazon EC2. To enforce this rule, you can use the following policy to set the permissions boundary for the `ShirleyRodriguez` user:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudwatch:*",
                "ec2:*"
            ],
            "Resource": "*"
        }
    ]
}
```

When you use a policy to set the permissions boundary for a user, it limits the user's permissions but does not provide permissions on its own. In this example, the policy sets the maximum permissions of `ShirleyRodriguez` as all operations in Amazon S3, CloudWatch, and Amazon EC2. Shirley can never perform operations in any other service, including IAM, even if she has a permissions policy that allows it. For example, you can add the following policy to the `ShirleyRodriguez` user:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
```

```
    "Action": "iam:CreateUser",
    "Resource": "*"
  }
}
```

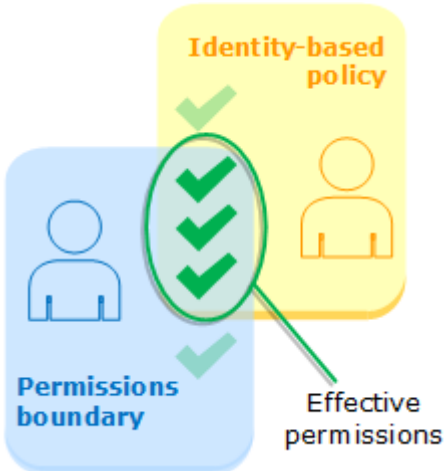
This policy allows creating a user in IAM. If you attach this policy to the `ShirleyRodriguez` user, and Shirley tries to create a user, the operation fails. It fails because the policy evaluation logic checks the policy used as the permissions boundary, which does not allow the `iam:CreateUser` operation. To allow Shirley to perform any operations in AWS, you must add a permissions policy with actions in Amazon S3, Amazon CloudWatch, or Amazon EC2. Alternatively, you could update the permissions boundary to allow her to create a user in IAM.

Evaluating Effective Permissions with Boundaries

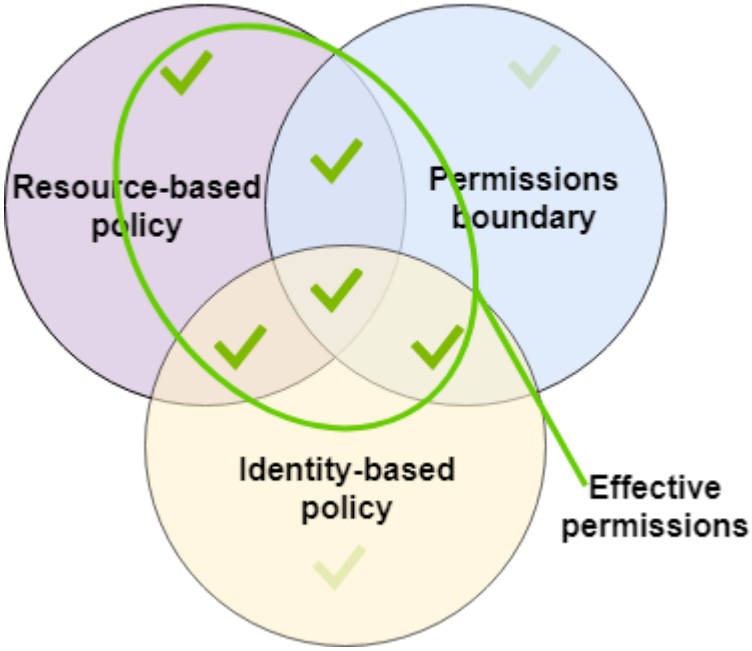
The permissions boundary for an IAM entity (user or role) sets the maximum permissions that the entity can have. This can change the effective permissions for that user or role. The effective permissions for an entity are the permissions that are granted by all the policies that affect the user or role. Within an account, the permissions for an entity can be affected by identity-based policies, resource-based policies, permissions boundaries, Organizations SCPs, or session policies. For more information about the different types of policies, see Policies and Permissions (access_policies.html) .

If any one of these policy types explicitly denies access for an operation, then the request is denied. The permissions granted to an entity by multiple permissions types are more complex. For more details about how AWS evaluates policies, see Policy Evaluation Logic (reference_policies_evaluation-logic.html) .

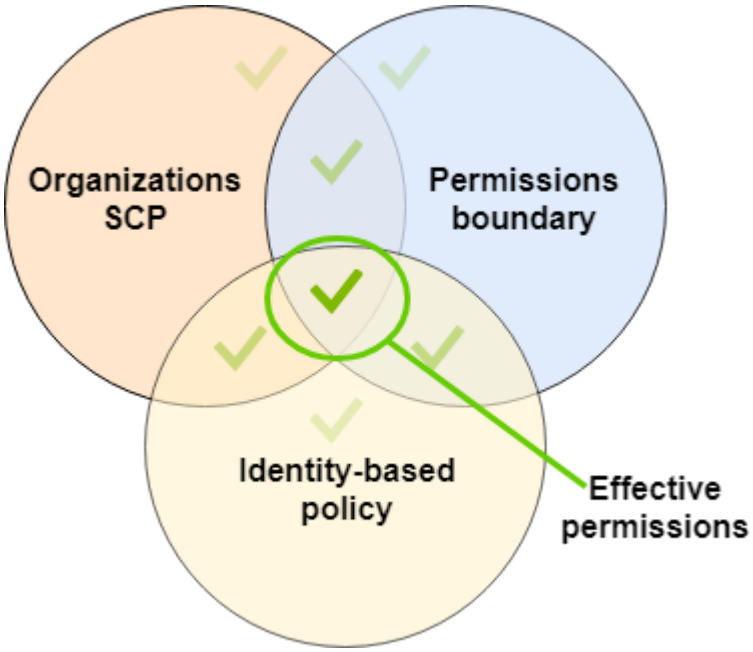
Identity-based policies with boundaries – Identity-based policies are inline or managed policies that are attached to a user, group of users, or role. Identity-based policies grant permission to the entity, and permissions boundaries limit those permissions. The effective permissions are the intersection of both policy types. An explicit deny in either of these policies overrides the allow.



Resource-based policies – Resource-based policies control how the specified principal can access the resource to which the policy is attached. Within an account, permissions boundaries do not reduce the permissions granted by resource-based policies. Permissions boundaries reduce permissions that are granted to an entity by identity-based policies, and then resource-based policies provide additional permissions to the entity. In this case, the effective permissions are everything that is allowed by the resource-based policy *and* the intersection of the permissions boundary and the identity-based policy. An explicit deny in any of these policies overrides the allow.

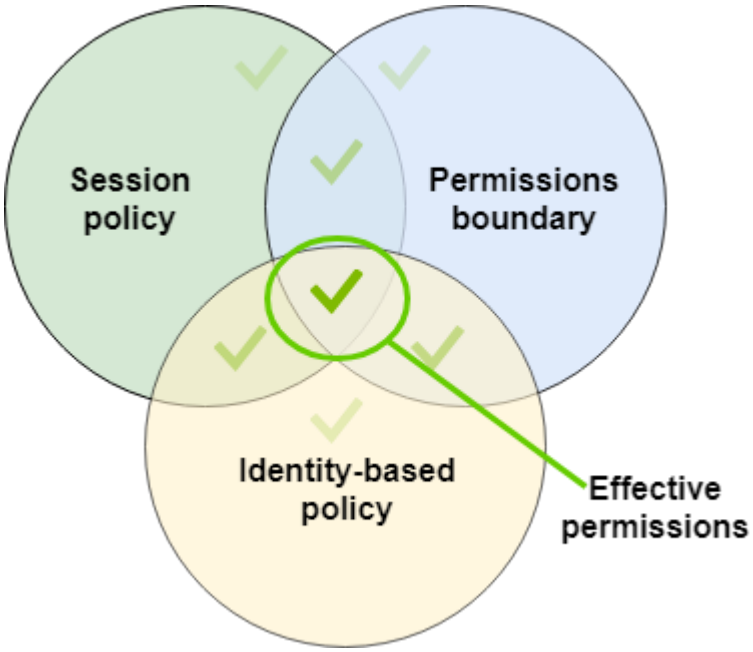


Organizations SCPs – SCPs are applied to an entire AWS account. They limit permissions for every request made by a principal within the account. An IAM entity (user or role) can make a request that is affected by an SCP, a permissions boundary, and an identity-based policy. In this case, the request is allowed only if all three policy types allow it. The effective permissions are the intersection of all three policy types. An explicit deny in any 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_vie 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.

Session policies – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The permissions for a session come from the IAM entity (user or role) used to create the session and from the session policy. The entity's identity-based policy permissions are limited by the session policy and the permissions boundary. The effective permissions for this set of policy types are the intersection of all three policy types. An explicit deny in any of these policies overrides the allow. For more information about session policies, see Session Policies (https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#policies_session)
.



Delegating Responsibility to Others Using Permissions Boundaries

You can use permissions boundaries to delegate permissions management tasks, such as user creation, to IAM users in your account. This permits others to perform tasks on your behalf within a specific boundary of permissions.

For example, assume that María is the administrator of the X-Company AWS account. She wants to delegate user creation duties to Zhang. However, she must ensure that Zhang creates users that adhere to the following company rules:

- Users cannot use IAM to create or manage users, groups, roles, or policies.
- Users are denied access to the Amazon S3 `logs` bucket and cannot access the `i-1234567890abcdef0` Amazon EC2 instance.
- Users cannot remove their own boundary policies.

To enforce these rules, María completes the following tasks, for which details are included below:

1. María creates the `XCompanyBoundaries` managed policy to use as a permissions boundary for all new users in the account.
2. María creates the `DelegatedUserBoundary` managed policy and assigns it as the permissions boundary for Zhang.
3. María creates the `DelegatedUserPermissions` managed policy and attaches it as a permissions policy for Zhang.
4. María tells Zhang about his new responsibilities and limitations.

Task 1: María must first create a managed policy to define the boundary for the new users. María will allow Zhang to give users the permissions policies they need, but she wants those users to be restricted. To do this, she creates the following customer managed policy with the name `XCompanyBoundaries`. This policy does the following:

- Allows users full access to several services
- Allows limited self-managing access in the IAM console. This means they can change their password after signing into the console. They cannot use access keys to programmatically change their password. They also can't set their initial password. To allow this, add the `"*LoginProfile"` action to the `AllowManageOwnPasswordAndAccessKeys` statement.
- Denies users access to the Amazon S3 logs bucket or the `i-1234567890abcdef0` Amazon EC2 instance

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ServiceBoundaries",
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudwatch:*",
                "ec2:*",
                "dynamodb:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowIAMConsoleForCredentials",
            "Effect": "Allow",
            "Action": [
                "iam:ListUsers",
                "iam:GetAccountPasswordPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowManageOwnPasswordAndAccessKeys",
            "Effect": "Allow",
            "Action": [
                "iam:*AccessKey*",
                "iam:ChangePassword",
                "iam:GetUser",
                "iam:*ServiceSpecificCredential*",
                "iam:*SigningCertificate*"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "DenyS3Logs",
            "Effect": "Deny",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::logs",
                "arn:aws:s3:::logs/*"
            ]
```

```
        },
        {
            "Sid": "DenyEC2Production",
            "Effect": "Deny",
            "Action": "ec2:*",
            "Resource": "arn:aws:ec2:*:*:instance/i-1234567890abcdef0"
        }
    ]
}
```

Each statement serves a different purpose:

1. The `ServiceBoundaries` statement of this policy allows full access to the specified AWS services. This means that a new user's actions in these services are limited only by the permissions policies that are attached to the user.
2. The `AllowIAMConsoleForCredentials` statement allows access to list all IAM users. This access is necessary to navigate the **Users** page in the AWS Management Console. It also allows viewing the password requirements for the account, which is necessary when changing your own password.
3. The `AllowManageOwnPasswordAndAccessKeys` statement allows the users manage only their own console password and programmatic access keys. This is important if Zhang or another administrator gives a new user a permissions policy with full IAM access. In that case, that user could then change their own or other users' permissions. This statement prevents that from happening.
4. The `DenyS3Logs` statement explicitly denies access to the `logs` bucket.
5. The `DenyEC2Production` statement explicitly denies access to the `i-1234567890abcdef0` instance.

Task 2: María wants to allow Zhang to create all X-Company users, but only with the `XCompanyBoundaries` permissions boundary. She creates the following customer managed policy named `DelegatedUserBoundary`. This policy defines the maximum permissions that Zhang can have.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateOrChangeOnlyWithBoundary",
            "Effect": "Allow",
            "Action": [
                "iam:CreateUser",
                "iam:DeleteUserPolicy",
                "iam:AttachUserPolicy",
                "iam:DetachUserPolicy",
                "iam:PutUserPermissionsBoundary"
            ],
            "Resource": "*",
            "Condition": {"StringEquals":
```

```
                {"iam:PermissionsBoundary":
"arn:aws:iam::111122223333:policy/XCompanyBoundaries"}}
        },
        {
            "Sid": "CloudWatchAndOtherIAMTasks",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:*",
                "iam:GetUser",
                "iam:ListUsers",
                "iam:DeleteUser",
                "iam:UpdateUser",
                "iam:CreateAccessKey",
                "iam:CreateLoginProfile",
                "iam:GetAccountPasswordPolicy",
                "iam:GetLoginProfile",
                "iam:*Group*",
                "iam:CreatePolicy",
                "iam:DeletePolicy",
                "iam:DeletePolicyVersion",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetUserPolicy",
                "iam:GetRolePolicy",
                "iam:ListPolicies",
                "iam:ListPolicyVersions",
                "iam:ListEntitiesForPolicy",
                "iam:ListUserPolicies",
                "iam:ListAttachedUserPolicies",
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:PutUserPolicy",
                "iam:SetDefaultPolicyVersion",
                "iam:SimulatePrincipalPolicy",
                "iam:SimulateCustomPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "NoBoundaryPolicyEdit",
            "Effect": "Deny",
            "Action": [
                "iam:CreatePolicyVersion",
                "iam:DeletePolicy",
                "iam:DeletePolicyVersion",
                "iam:SetDefaultPolicyVersion"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:policy/XCompanyBoundaries",
                "arn:aws:iam::123456789012:policy/DelegatedUserBoundary"
            ]
        },
        {
            "Sid": "NoBoundaryUserDelete",
            "Effect": "Deny",
            "Action": "iam:DeleteUserPermissionsBoundary",
            "Resource": "*"
        }
    ]
}
```

Each statement serves a different purpose:

1. The `CreateOrChangeOnlyWithBoundary` statement allows Zhang to create IAM users but only if he uses the `XCompanyBoundaries` policy to set the permissions boundary. This statement also allows him to set the permissions boundary for existing users but only using that same policy. Finally, this statement allows Zhang to manage permissions policies for users with this permissions boundary set.

2. The `CloudWatchAndOtherIAMTasks` statement allows Zhang to complete other user, group, and policy management tasks. Note that Zhang does not have the permission to delete the permissions boundary from himself or any other user.

3. The `NoBoundaryPolicyEdit` statement denies Zhang access to update the `XCompanyBoundaries` policy. He is not allowed to change any policy that is used to set the permissions boundary for himself or other users.

4. The `NoBoundaryUserDelete` statement denies Zhang access to delete the permissions boundary for himself or other users.

María then assigns the `DelegatedUserBoundary` policy as the permissions boundary (id_users_change-permissions.html#users_change_permissions-set-boundary-console) for the `Zhang` user.

Task 3: Because the permissions boundary limits the maximum permissions, but does not grant access on its own, Maria must create a permissions policy for Zhang. She creates the following policy named `DelegatedUserPermissions`. This policy defines the operations that Zhang can perform, within the defined boundary.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "IAM",
            "Effect": "Allow",
            "Action": "iam:*",
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchLimited",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetDashboard",
                "cloudwatch:GetMetricData",
                "cloudwatch:ListDashboards",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "S3BucketContents",
            "Effect": "Allow",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::ZhangBucket"
        }
```

```
    ]
}
```

Each statement serves a different purpose:

1. The `IAM` statement of the policy allows Zhang full access to IAM. However, because his permissions boundary allows only some IAM operations, his effective IAM permissions are limited only by his permissions boundary.
2. The `CloudWatchLimited` statement allows Zhang to perform five actions in CloudWatch. His permissions boundary allows all actions in CloudWatch, so his effective CloudWatch permissions are limited only by his permissions policy.
3. The `S3BucketContents` statement allows Zhang to list the `ZhangBucket` Amazon S3 bucket. However, his permissions boundary does not allow any Amazon S3 action, so he cannot perform any S3 operations, regardless of his permissions policy.

María then attaches the `DelegatedUserPermissions` policy as the permissions policy for the `Zhang` user.

Task 4: She gives Zhang instructions to create a new user. She tells him that he can create new users with any permissions that they need, but he must assign them the `XCompanyBoundaries` policy as a permissions boundary.

Zhang completes the following tasks:

1. Zhang creates a user (id_users_create.html#id_users_create_console) with the AWS Management Console. He types the user name `Nikhil` and enables console access for the user. He clears the checkbox next to **Requires password reset**, because the policies above allow Zhang to change his password only after he is signed in to the IAM console.
2. On the **Set permissions** page, Zhang chooses the **IAMFullAccess** and **AmazonS3ReadOnlyAccess** permissions policies that allow Nikhil to do his work.
3. Zhang skips the **Set permissions boundary** section, forgetting María's instructions.
4. Zhang reviews the user details and chooses **Create user**.

   The operation fails and access is denied. Zhang's `DelegatedUserBoundary` permissions boundary requires that any user he creates have the `XCompanyBoundaries` policy used as a permissions boundary.
5. Zhang returns to the previous page. In the **Set permissions boundary** section, he chooses the `XCompanyBoundaries` policy.
6. Zhang reviews the user details and chooses **Create user**.

The user is created.

When Nikhil signs in, he has access to IAM and Amazon S3, except those operations that denied by the permissions boundary. For example, he can change his own password in IAM but can't create another user or edit his policies. Nikhil has read-only access to all the buckets that he owns in Amazon S3. However, even if someone grants him ownership to the `logs` bucket, he cannot view it. For more information about bucket ownership, see Managing Access Permissions to your Amazon S3 Resources (https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-access-control.html) in the *Amazon Simple Storage Service Developer Guide*.

If someone adds a resource-based policy to the `logs` bucket that allows Nikhil to put an object in the bucket, he still cannot access the bucket. The reason is that any actions on the `logs` bucket are explicitly denied by his permissions boundary. An explicit deny in any policy type results in a request being denied. However, if a resource-based policy attached to a Secrets Manager secret allows Nikhil to perform the `secretsmanager:GetSecretValue` action, then Nikhil can retrieve and decrypt the secret. The reason is that Secrets Manager operations are not explicitly denied by his permissions boundary, and implicit denies in permissions boundaries do not limit resource-based policies.