This is the companion deck to Permissions Boundaries workshop:
https://awssecworkshops.com/workshops/identity-round-robin/permission-boundaries/

Version 1.4

Remove slide before presenting

Identity Round Robin Workshop

Permissions Boundaries

Agenda

- Permissions boundary intro & basics
- Demo
- Permission categories
- Permissions boundary mechanism
- Resource restrictions

What are permissions boundaries?

Mechanism to delegate the permission to create users and/or roles while preventing privilege escalation or unnecessarily broad permissions. Controls the maximum permissions that a user and/or role can have, but does not provide any permissions.

Method to safely grant actions like:
"iam:CreateRole"
"iam:PassRole"

Before and After Permissions Boundaries

Before

- Certain IAM policy actions (e.g. PutUserPolicy, AttachRolePolicy) are essentially full admin-like permissions.
- Doing any form of self-service permissions management was non-trivial.

Now

- Administrators can grant these full admin-like permissions, but specify a "permissions boundary."
- Allows developers to create principals for their applications and attach policies, but only within the boundary.

Use cases

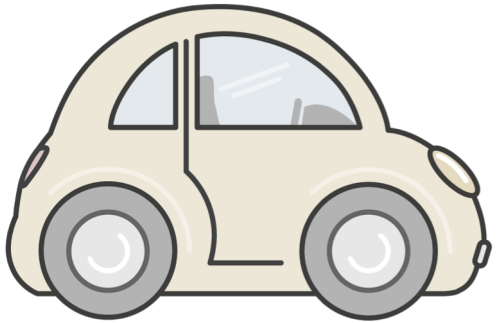
- Developers that need to create roles for Lambda functions
- Application owners that need to create roles for EC2 instances
- Admins that need to be able to create users for particular use cases
- Any others?

Just the facts

- Supported only for user and roles
- The mechanism at the policy level is just a condition context key
- Not all IAM actions support the condition context key
- It’s just a managed policy
- The user or role can do only the actions allowed by both the attached identity-based policies and the permissions boundary

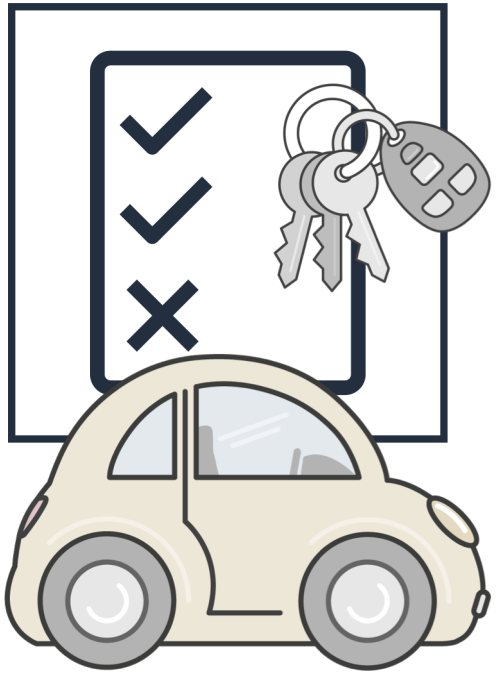
Analogy – giving the keys to your teenager

- Car keys give a lot of power: drive fast, drive anywhere and even drink and drive.
- Can set rules: don't speed, don't go beyond 20 mile range, etc, but that is trust based.
- Only other option is detect controls to verify compliance (check odometer, see if they got a speeding ticket or got into an accident.)



Analogy – giving the keys to your teenager

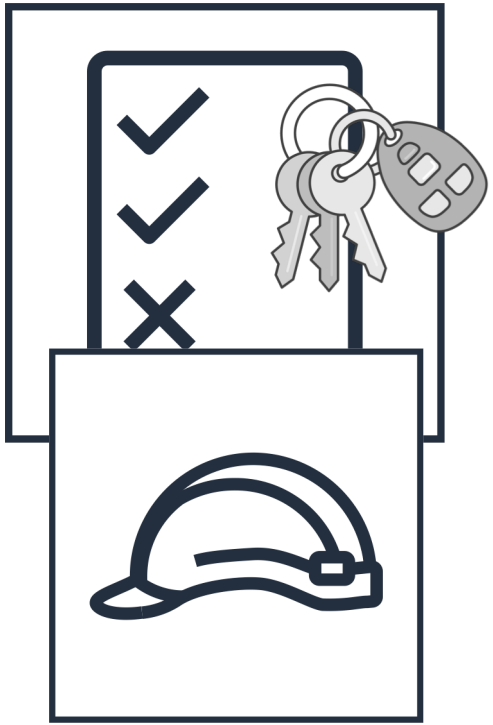
- Some auto makes have special keys and programming that allows you to let them drive but they are restricted by your settings.
- Their ability in the car (drive fast, blast the radio or even spin the tires) is the intersection between their desire and your settings.



aws

Analogy – giving the keys to a developer

- In the same way you can give the keys to a developer (ability to create user or roles) and all the power that comes with that.
- The developer can attach an identity-based policy with full admin rights (their desire) to the role but they must also attach a permissions boundary (like the auto restriction settings).
- Effective permission of the role is the intersection of the two.



Permissions boundary basics

Permissions Boundaries – workflow

Admins

Create delegated admins

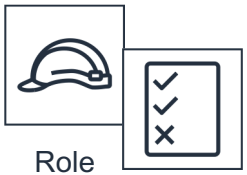
Requirement: users and roles created by delegated admins must have a permissions boundary

Delegated admins

Create “bound” users & roles

Ability: can create users and roles that have permissions boundaries attached

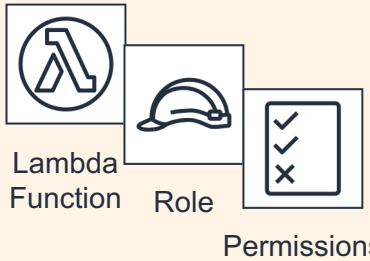
“Bound” IAM users and roles



Users and roles restricted by permissions boundaries

Result: Permissions boundary restrict the permissions of the users and roles

Restricted resources



Permissions for resources restricted

Permissions of the roles attached to resources like Lambda functions are limited by the permissions boundary

An IAM condition context key

```
"Condition": {"StringEquals":
    {"iam:PermissionsBoundary":
    "arn:aws:iam::ACCOUNT_ID:policy/permissionboundary"
    }
}
```


aws

… applied to principal creation actions (users and roles)

```
"Effect": "Allow",
"Action": ["iam:CreateRole"],
"Resource": ["arn:aws:iam::ACCOUNT_ID:role/path/"],
"Condition": {"StringEquals":
    {"iam:PermissionsBoundary":
    "arn:aws:iam::ACCOUNT_ID:policy/permissionboundary"
    }
}
```


aws

End user experience

Create role for a Lambda function

 # Step 1: Create role and attach permissions boundary

```
$ aws iam create-role –role-name roleforlambda
–assume-role-policy-document file://Role_Trust_Policy_Text.json
–permissions-boundary arn:aws:iam::<ACCOUNT_NUMBER>:policy/department_a/boundary_1
```

 # Step 2: Create identity-based policy

```
No change
```

 # Step 3: Attach identity-based policy

```
No change
```

Demo

- **User requirements**:
  - Lambda function that reads from an S3 bucket
  - Lambda function must have an IAM role to access the bucket
  - Role must be created with the correct permissions
- **Company requirements**:
  - Policies attached to the role must not allow privilege escalation or unneeded permissions
  - Don't get in the way of the user

Create:
- **Policy for a user (plus read only policies)**
- **Permissions boundary policy**
- **User**

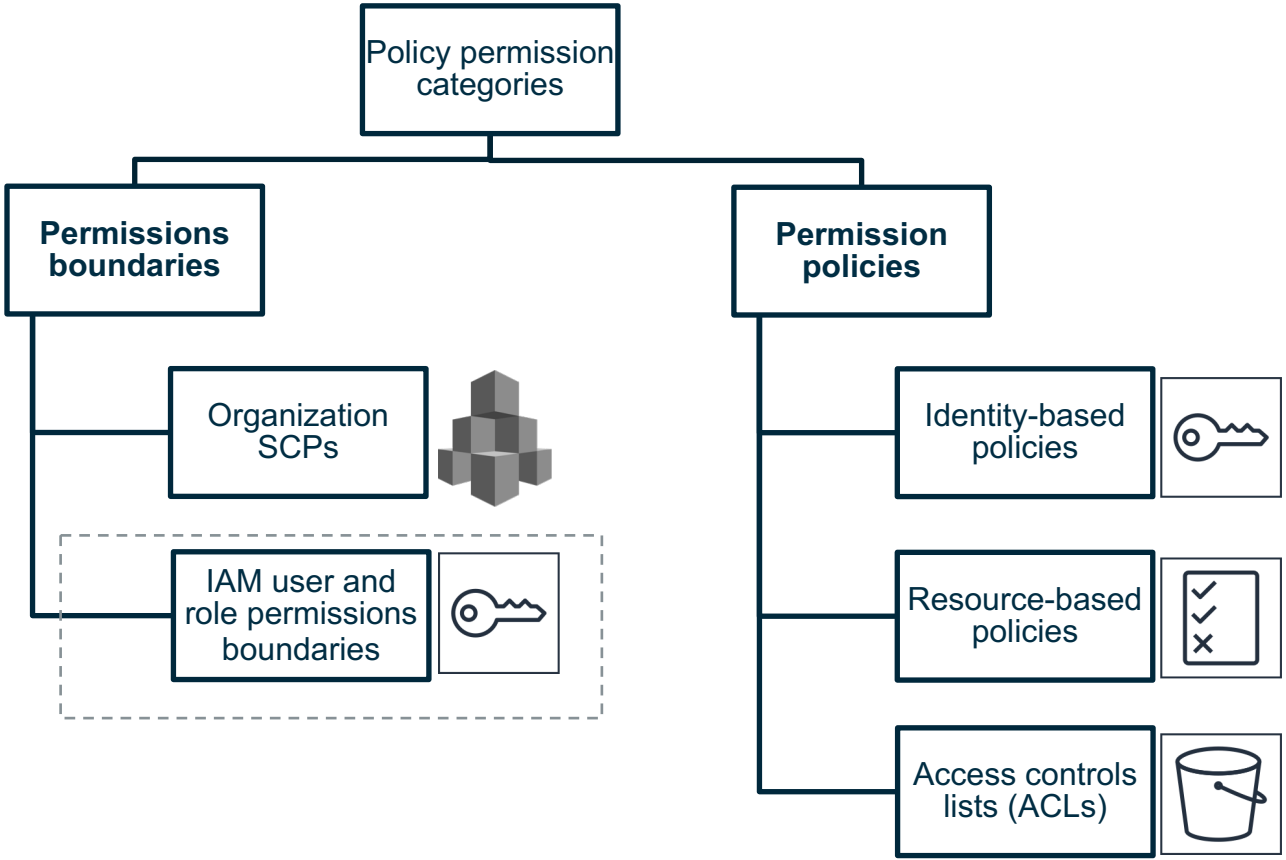
Create:
- **Policy for a role**
- **Role**
- **Lambda function**

Lambda function restricted by permissions boundary

Policy categories

Policy permission categories

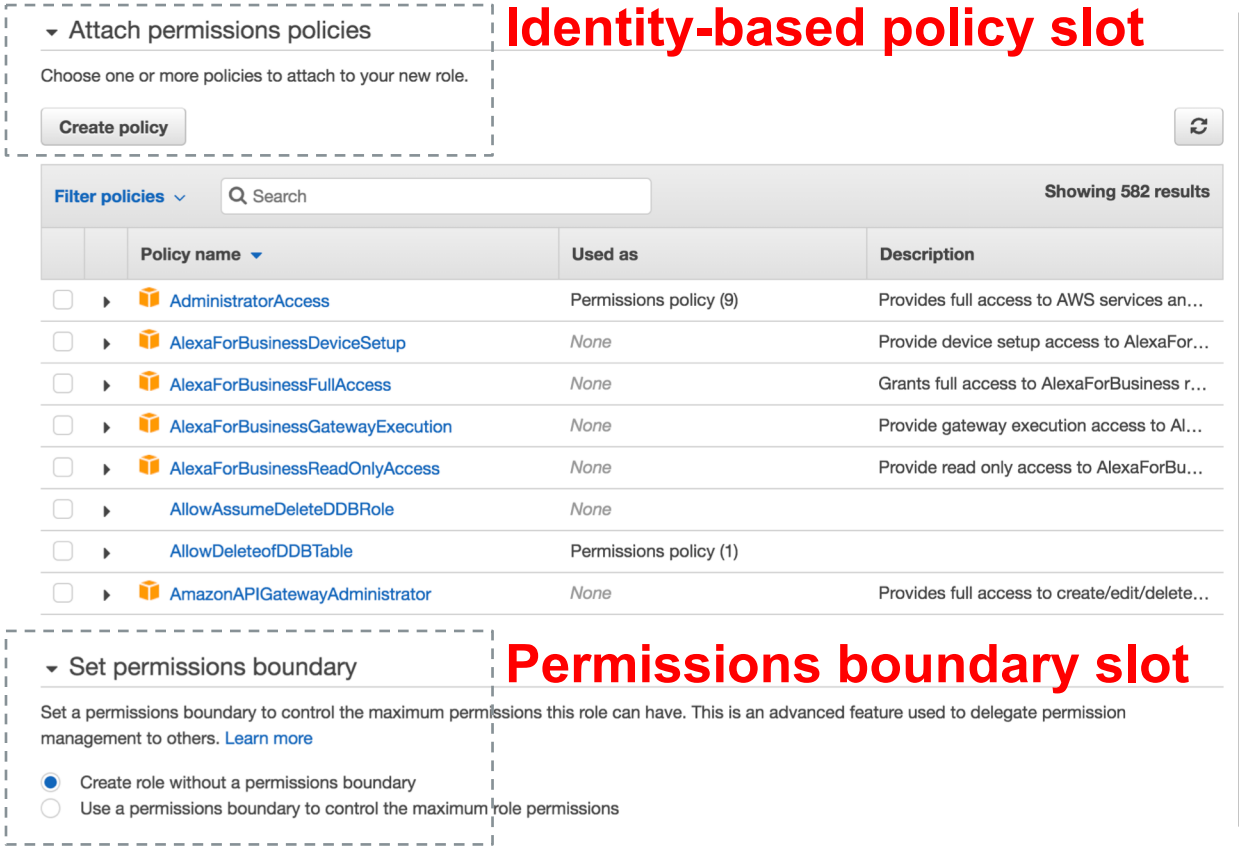
Policy permission categories



But, it’s just a managed IAM policy right?

IAM policy

But, it's just an IAM policy right?

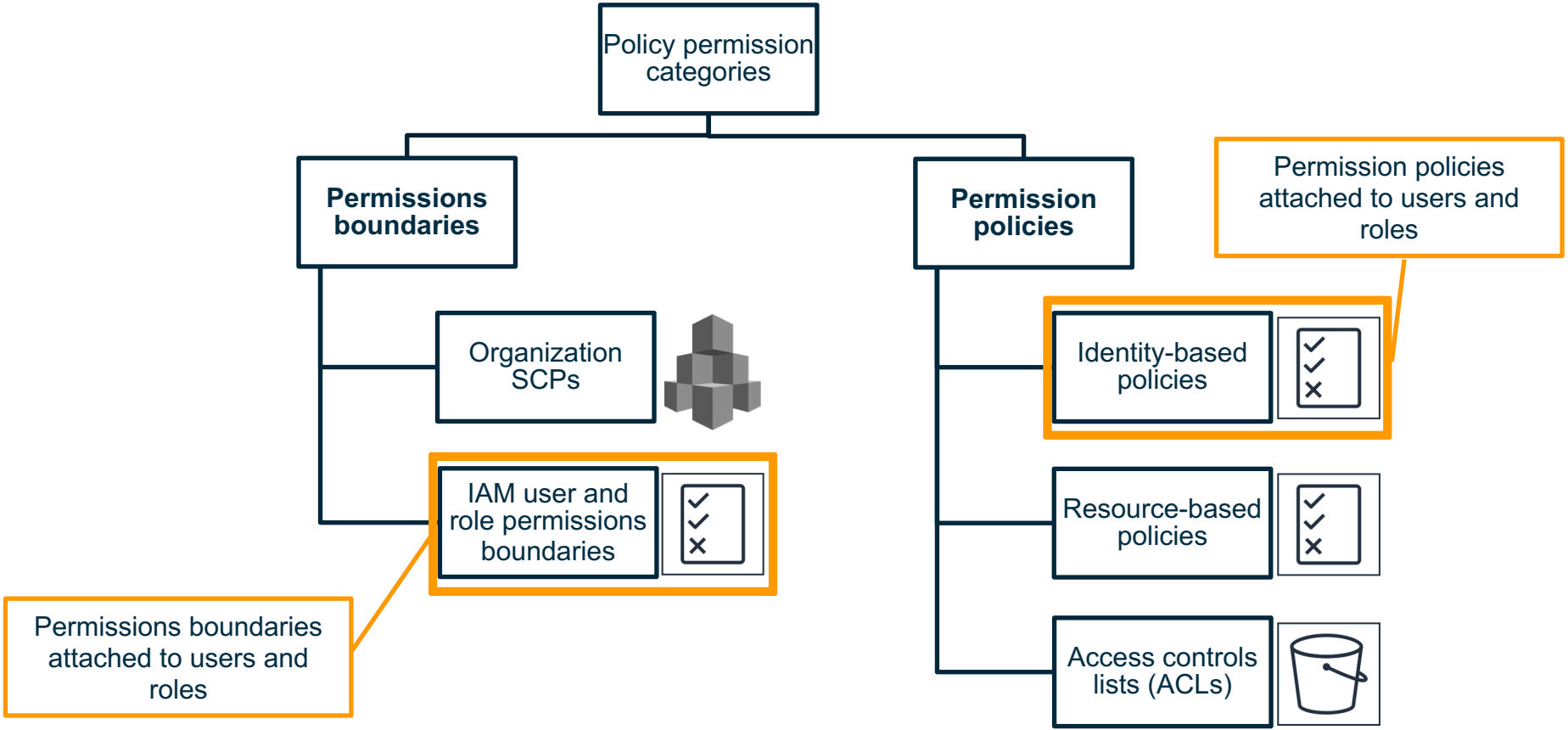


Presentation questions 1

- What is the condition context key used for permissions boundaries?
- How does a permissions boundary differ from an identity-based policy?
- What are some permissions boundary use cases?

Permissions boundary mechanism

Policy permission categories



Everything after authentication

1. **Authenticate** the principal

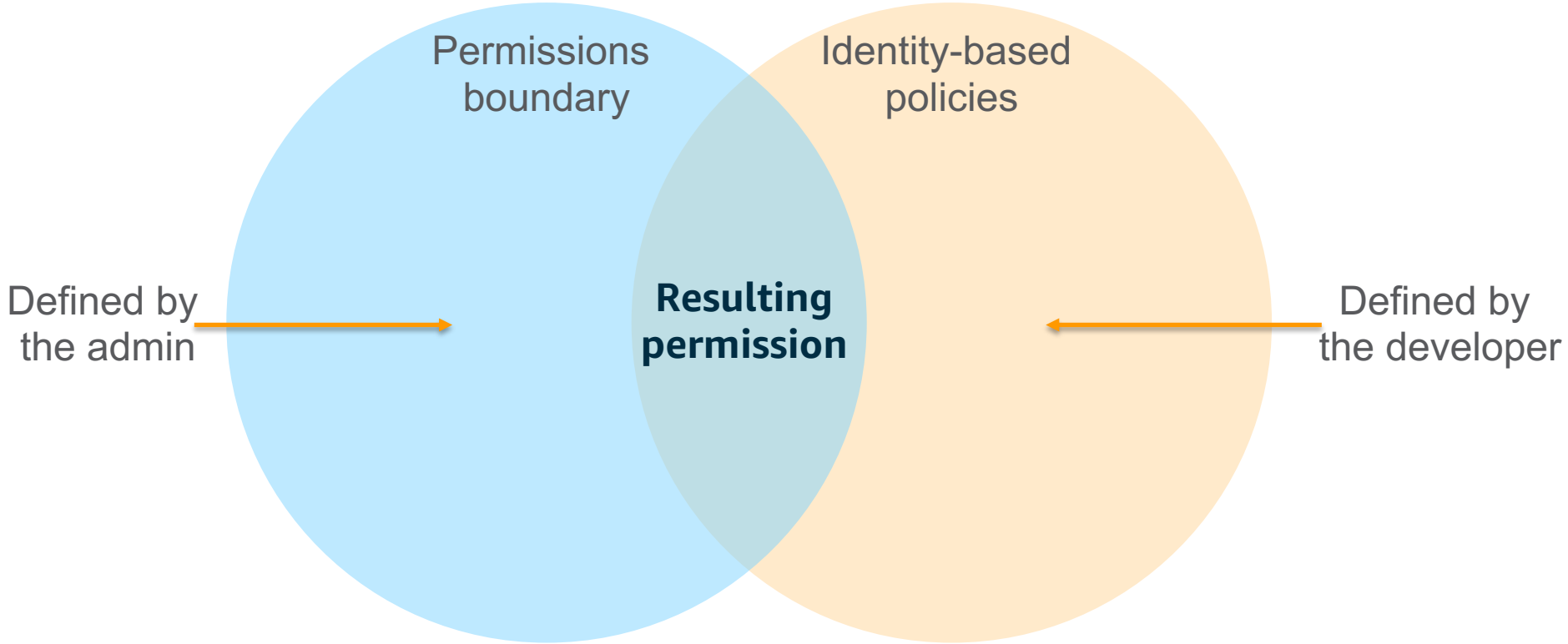
2. Determine which **policies** apply to the request

3. **Evaluate** the different policy types that apply which affect the order in which they are evaluated.

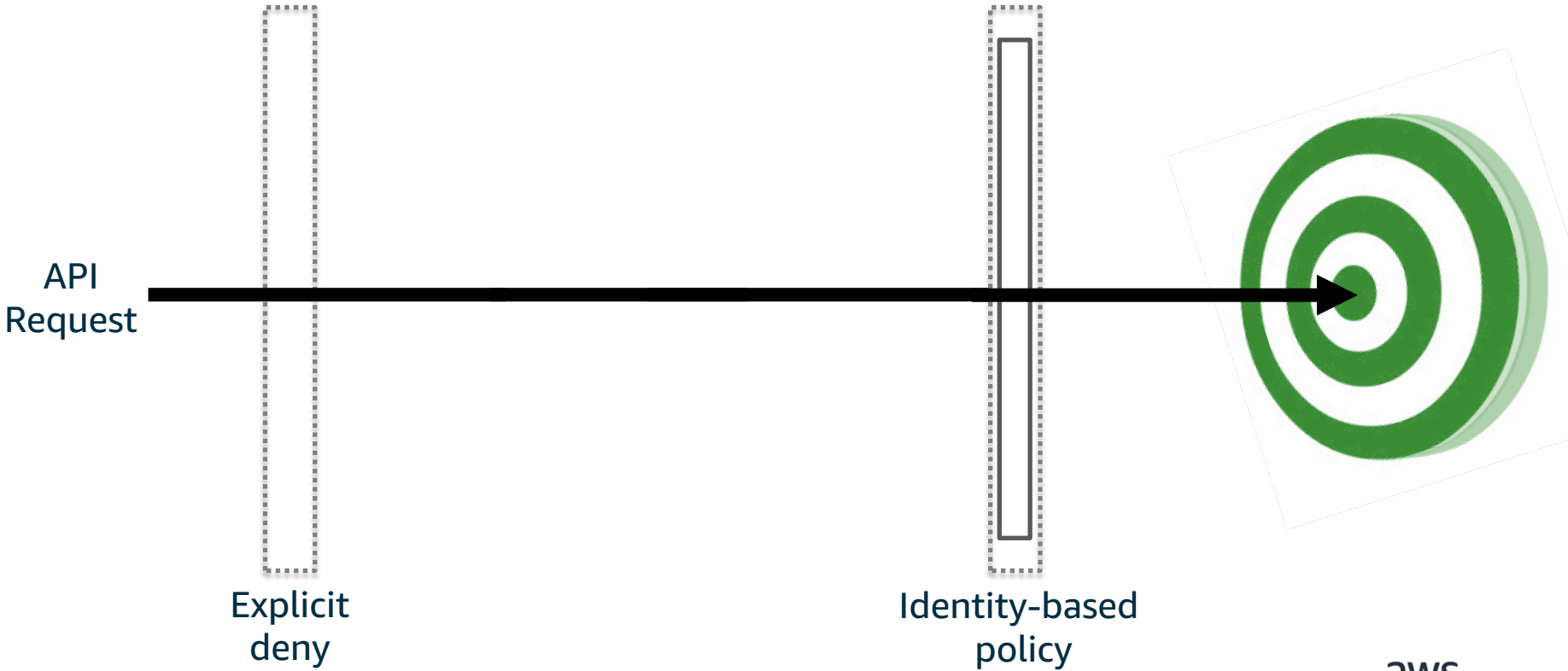
4. **Allow or Deny** the request

aws

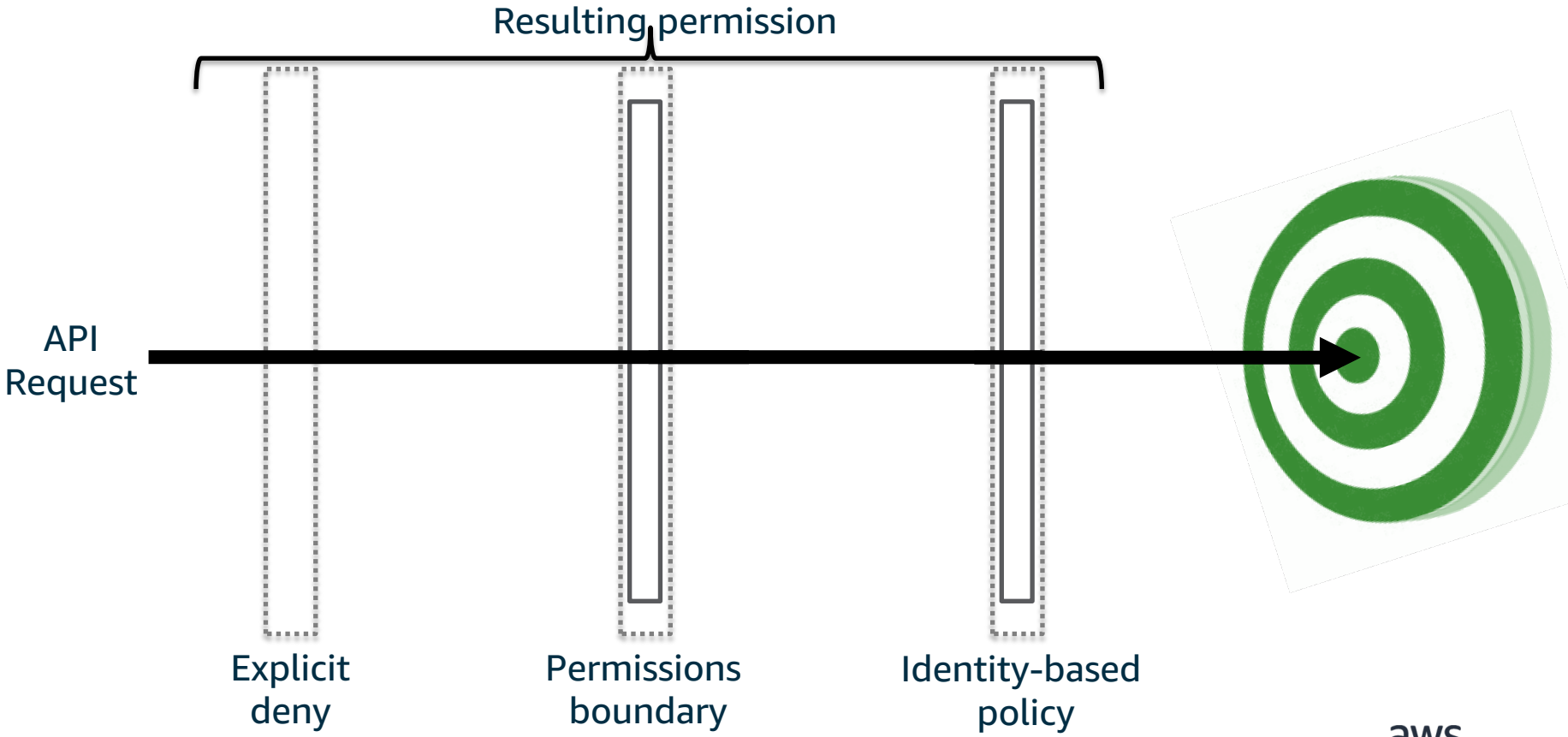
Effective Permissions – Venn diagram



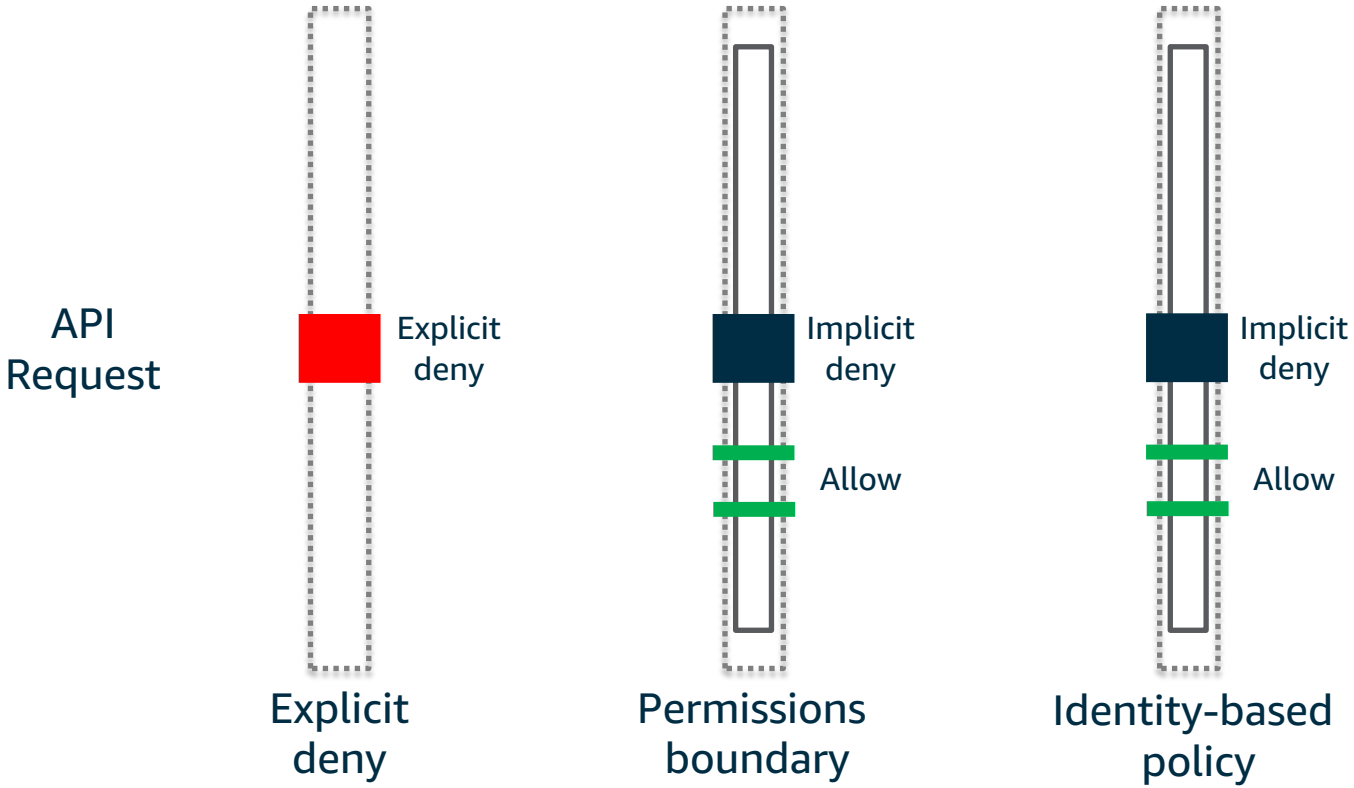
Effective permissions – mechanism



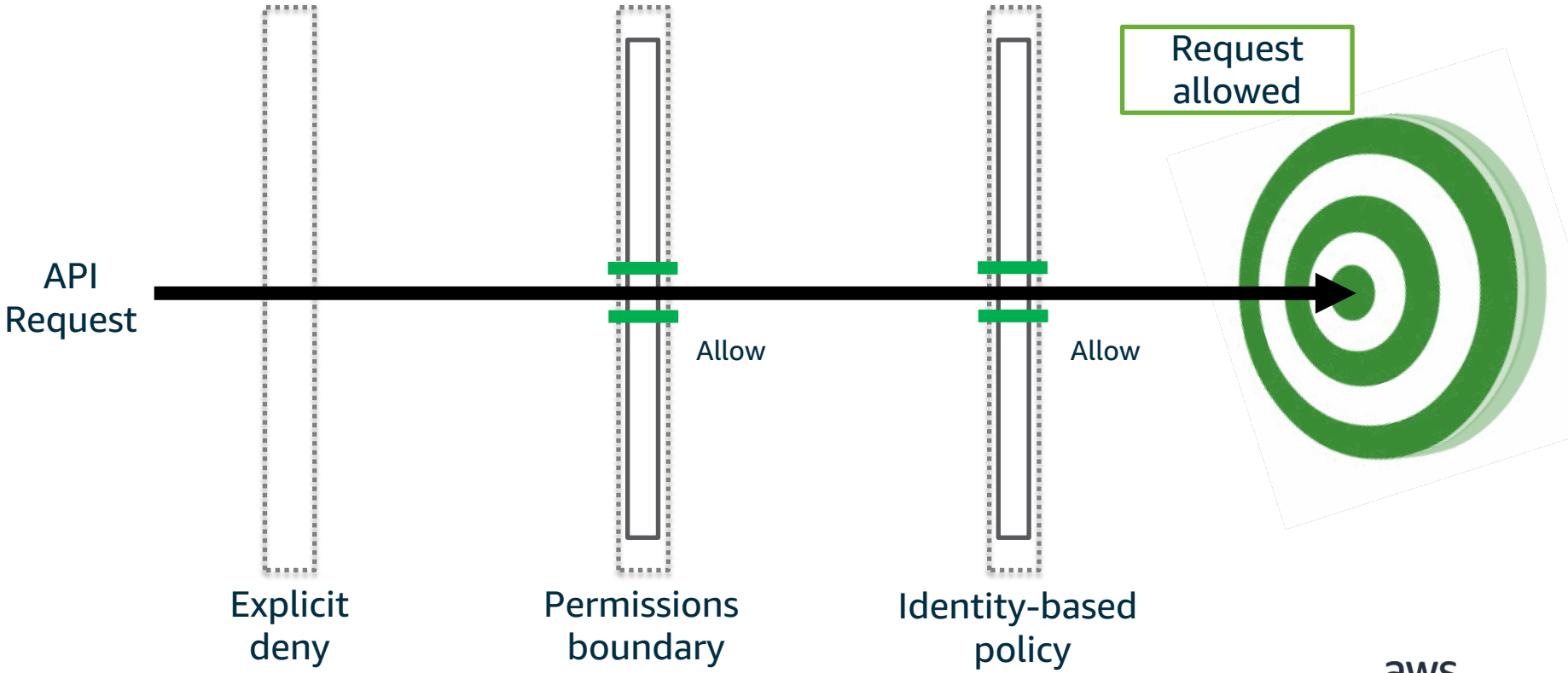
Effective permissions – mechanism



Effective permissions – mechanism



Effective permissions – allow example



Effective permissions – scenario 1

Request: s3:GetObject / bucket name: example1

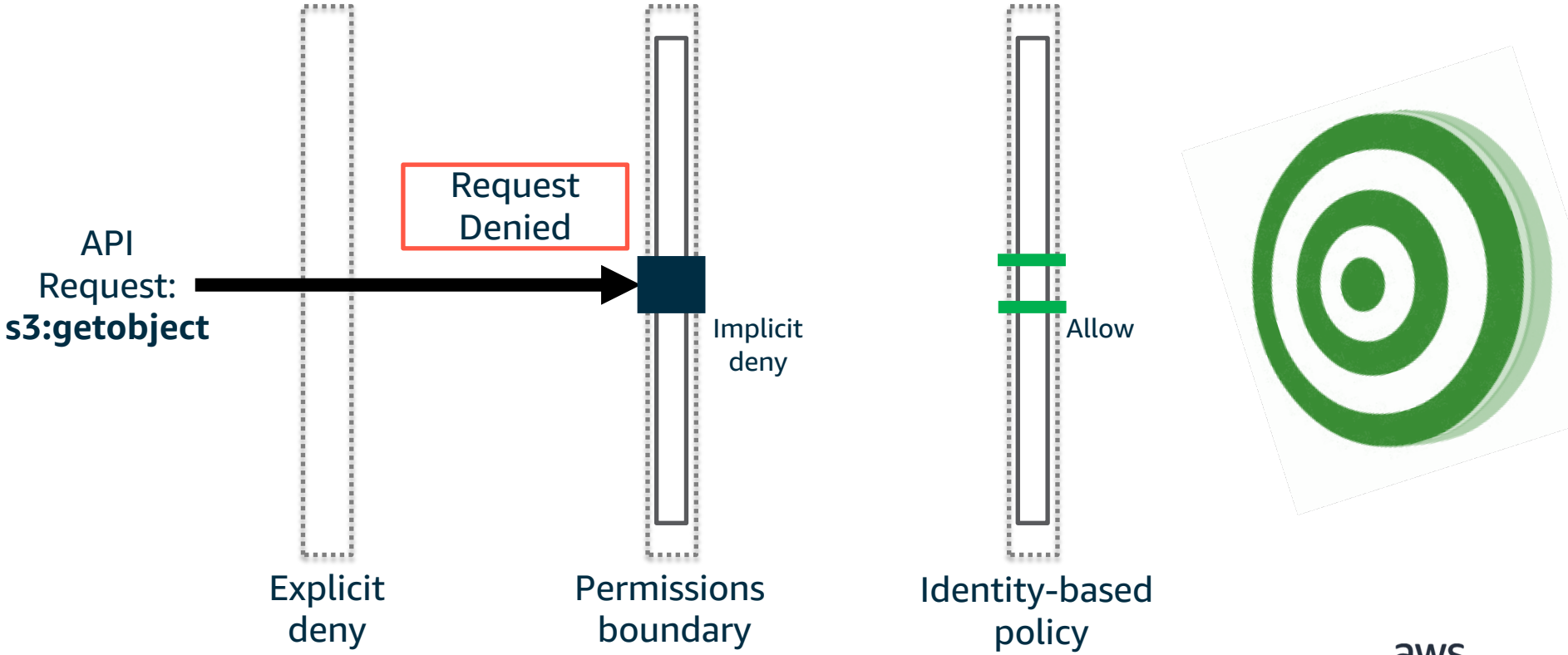
Permissions boundary

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "logs:CreateLogGroup",
         "logs:CreateLogStream”,
         "logs:PutLogEvents”
       ],
     "Resource": "arn:aws:logs:*:*:*"
     }
}
```

Identity-based Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "logs:CreateLogGroup",
         "logs:CreateLogStream",
         "logs:PutLogEvents",
         "s3:*"
        ],
      "Resource": "*"
      }
   ]
}
```

Effective permissions – result



Effective permissions – scenario 2

Request: s3:GetObject / bucket name: example1

Permissions boundary

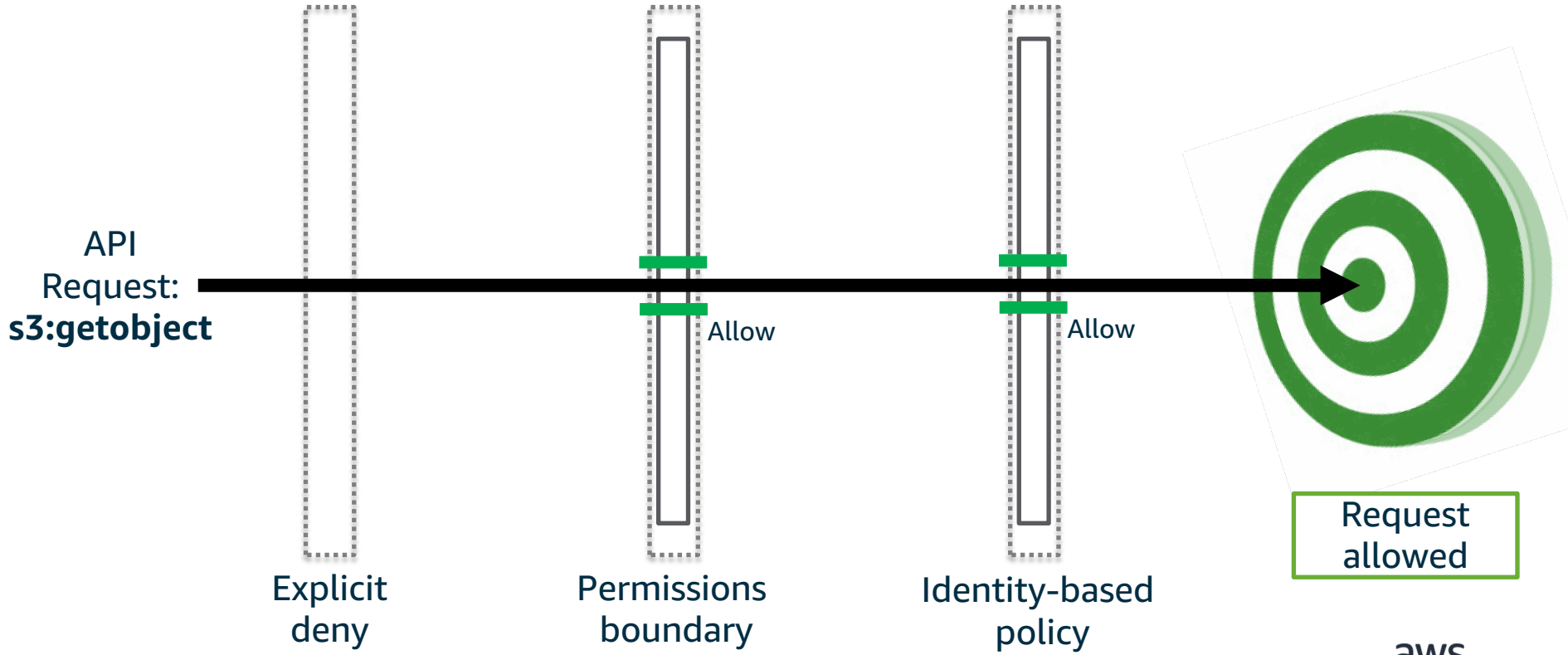
```
{
  "Version": "2012-10-17",
  "Statement": [
     {
       "Effect": "Allow",
       "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream”,
            "logs:PutLogEvents”
       ],
     "Resource": "arn:aws:logs:*:*:*"
     },
     {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource”:"arn:aws:s3:::example1/*"
     }
}
```

Identity-based policy

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
       "Effect": "Allow",
       "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "s3:*"
        ],
      "Resource": "*"
      }
   ]
}
```


aws

Effective permissions – result



Effective permissions – scenario 3

Request: s3:GetObject / bucket name: example1

Permissions boundary

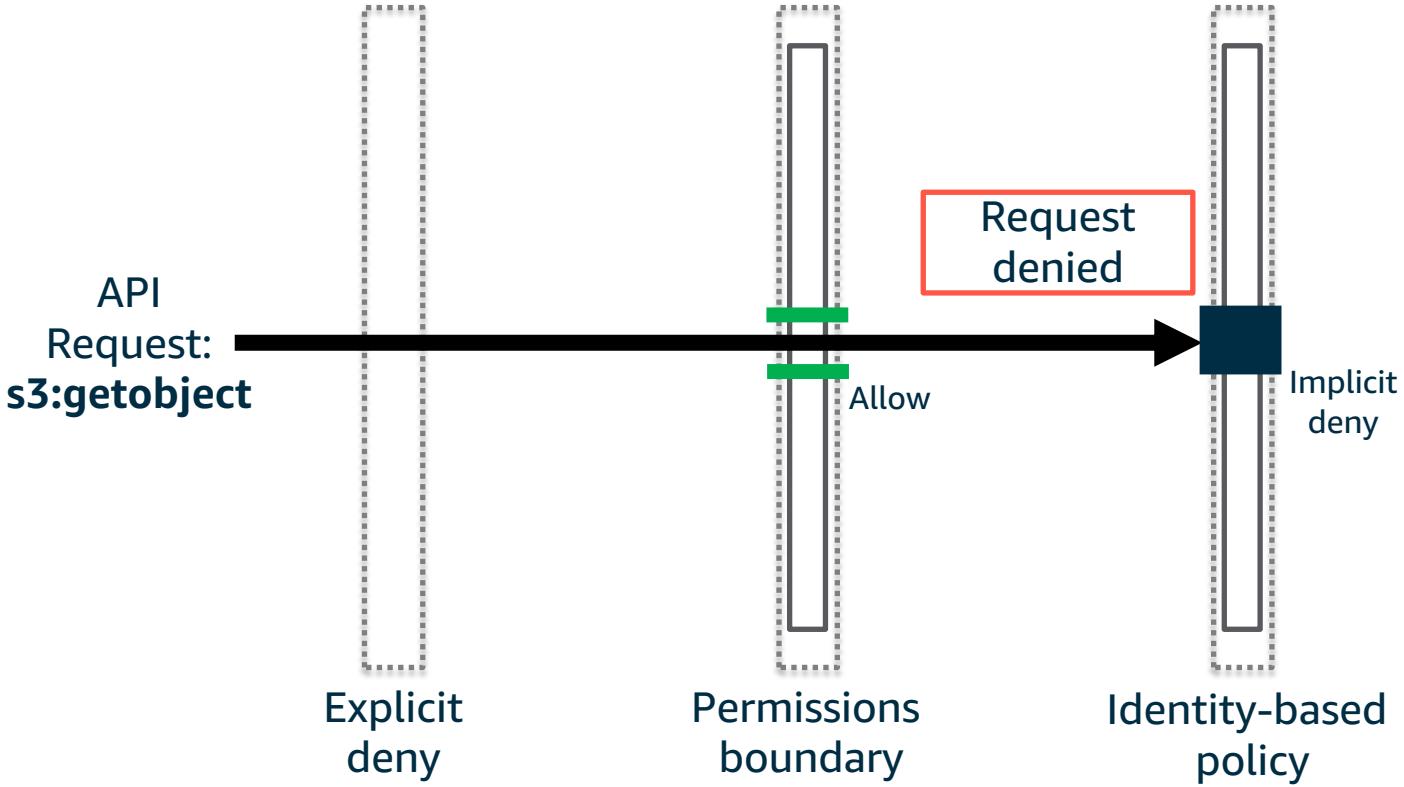
```
{
  "Version": "2012-10-17",
  "Statement": [
     {
       "Effect": "Allow",
       "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream”,
            "logs:PutLogEvents”
       ],
     "Resource": "arn:aws:logs:*:*:*"
     },
     {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource”:"arn:aws:s3:::example1/*"
     }
}
```

Identity-based policy

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
       "Effect": "Allow",
       "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
        ],
      "Resource": "*"
      }
   ]
}
```

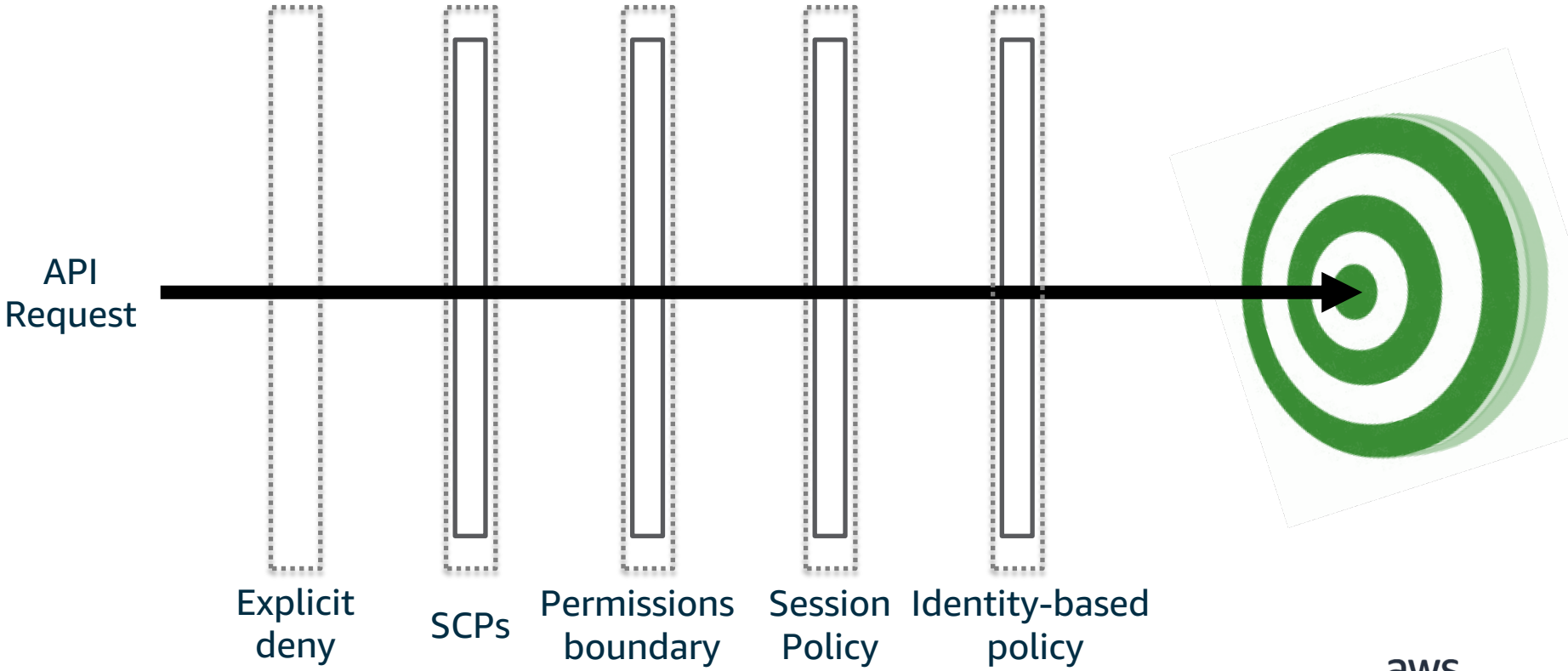

aws

Effective permissions – result

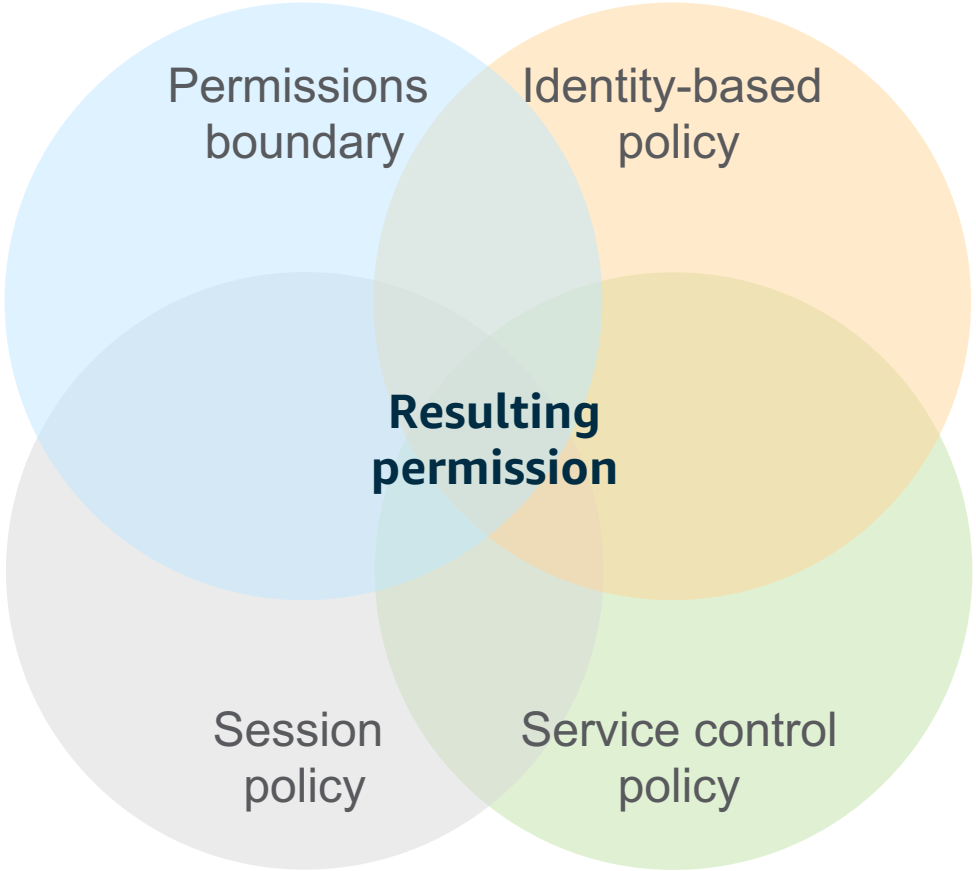


aws

Effective permissions – mechanism expanded

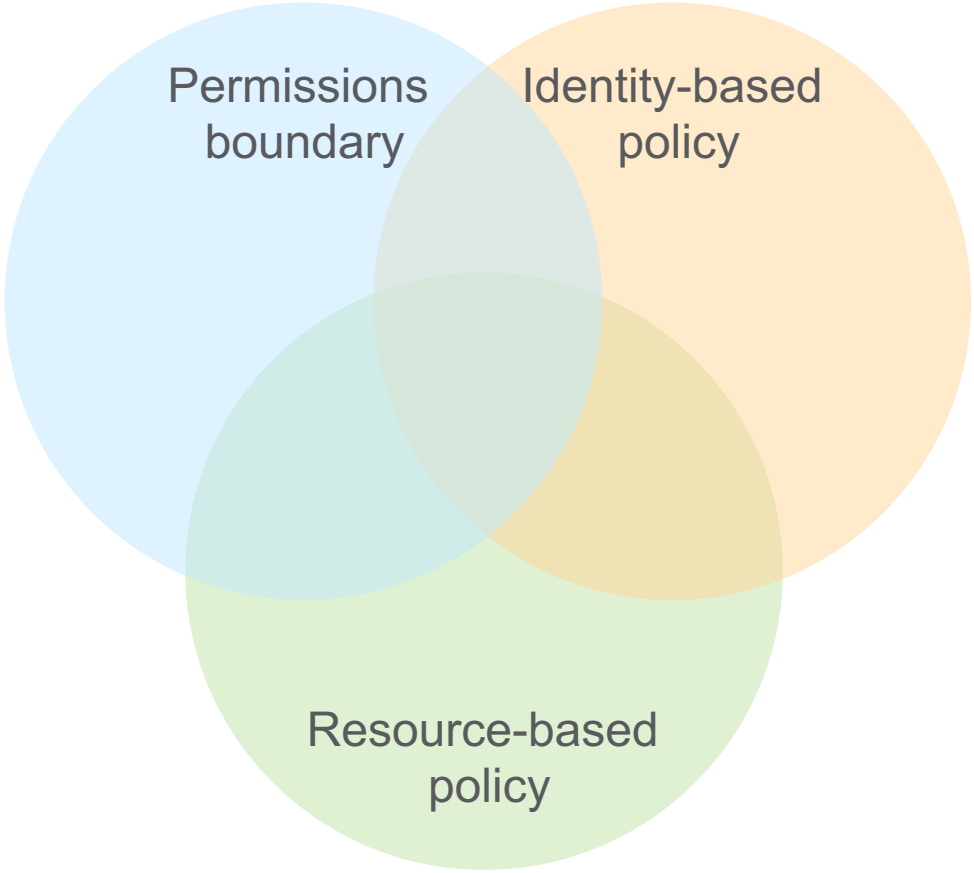


Effective Permissions – intersection expanded



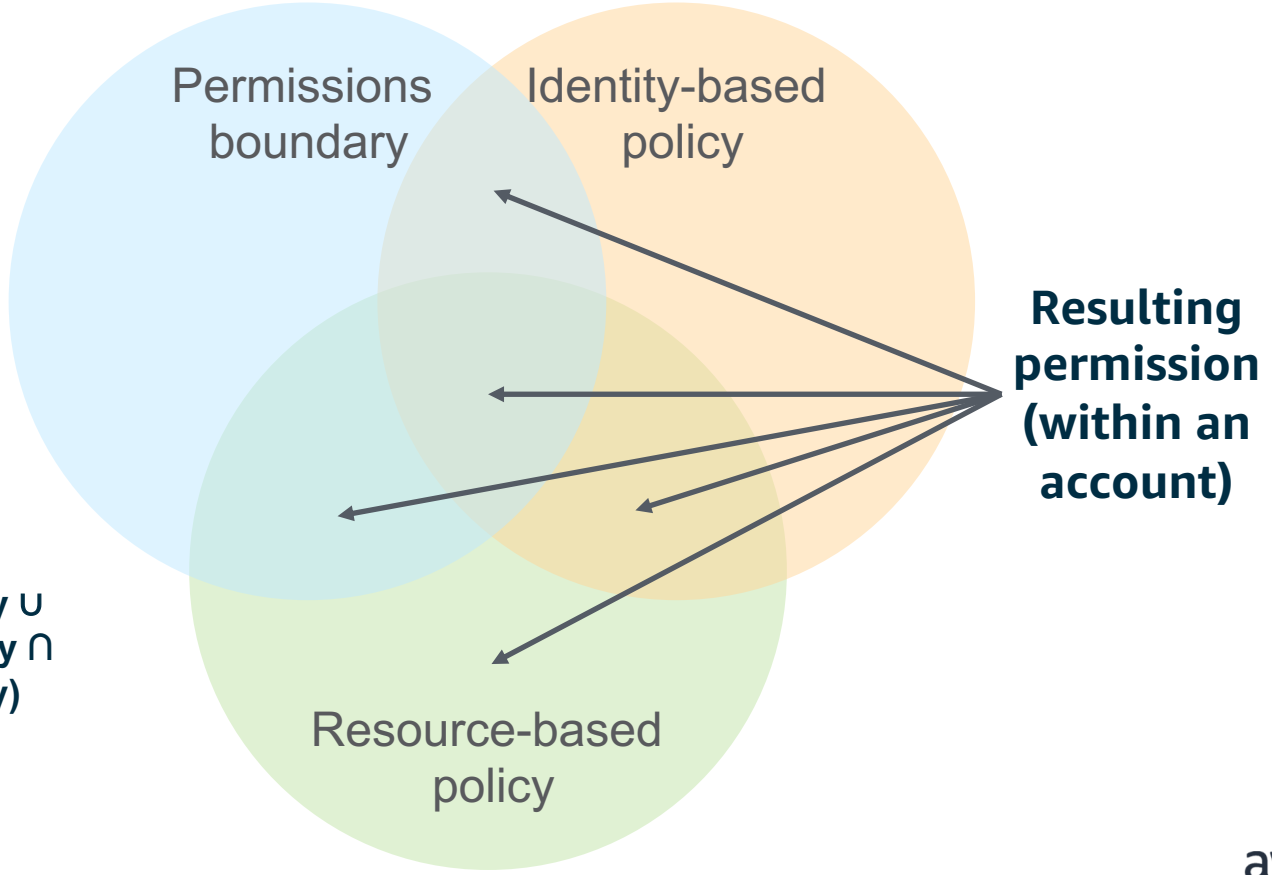
aws

Question: what about resource policies?



aws

What about resource policies?



Resource-based policy ∪ (Permissions boundary ∩ Identity-based policy)

aws

Resource Restrictions

Goal: carve out a space for the delegated admins to be able to modify resources without impacting other resources.

Paths are preferred but require the CLI. Naming (e.g. department1*) can also be used.

https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html#arns-paths

https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_identifiers.html#identifiers-arns

Resource Restrictions - examples

Resource restriction using **paths**:

"Resource": "arn:aws:iam::123456789012:role**/department1/***"

Example role:

arn:aws:iam::123456789012:role/department1/role1

Resource restriction using **names**:

"Resource" : "arn:aws:iam::123456789012:policy**/development-users***"

Example policy:

arn:aws:iam::123456789012:policy/development-users-policy1

aws

Resource Restrictions - policies

- If there is not a resource restriction then the delegated admins could modify any customer managed policies

```
"Effect": "Allow",
"Action": [
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:CreatePolicyVersion",
        "iam:DeletePolicyVersion",
        "iam:SetDefaultPolicyVersion"
        ],
"Resource": "*"
```

vs

```
"Effect": "Allow",
"Action": [
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:CreatePolicyVersion",
        "iam:DeletePolicyVersion",
        "iam:SetDefaultPolicyVersion"
        ],
"Resource":
"arn:aws:iam::ACCOUNT_ID:policy/path/name*"
```


aws

Resource Restrictions - roles

```
"Effect": "Allow",
"Action": [
        "iam:UpdateRole",
        "iam:DeleteRole”
        ],
"Resource": "*"
```

vs

```
"Effect": "Allow",
"Action": [
        "iam:UpdateRole",
        "iam:DeleteRole”
        ],
"Resource":
"arn:aws:iam::ACCOUNT_ID:role/path/name*"
```

Action that support the Condition context key

Here are the actions that the support the permissions boundary condition:

- AttachRolePolicy
- AttachUserPolicy
- CreateRole
- CreateUser
- DeleteRolePermissionsBoundary
- DeleteUserPermissionsBoundary
- DeleteRolePolicy
- DeleteUserPolicy
- DetachRolePolicy
- DetachUserPolicy
- PutRolePermissionsBoundary
- PutUserPermissionsBoundary
- PutRolePolicy
- PutUserPolicy

Final Q & A

Presentation questions 2

- What is the risk of implementing permissions boundaries without resource restrictions?
- Can the same IAM policy be used as both a permissions boundary and a Identity-based policy?
- Is one resource restriction method preferred over the other?