Identity Round Robin Builder Session Permission Boundaries

What are permission boundaries?

Mechanism to delegate the permission to create users and roles while preventing privilege escalation or unnecessarily broad permissions.
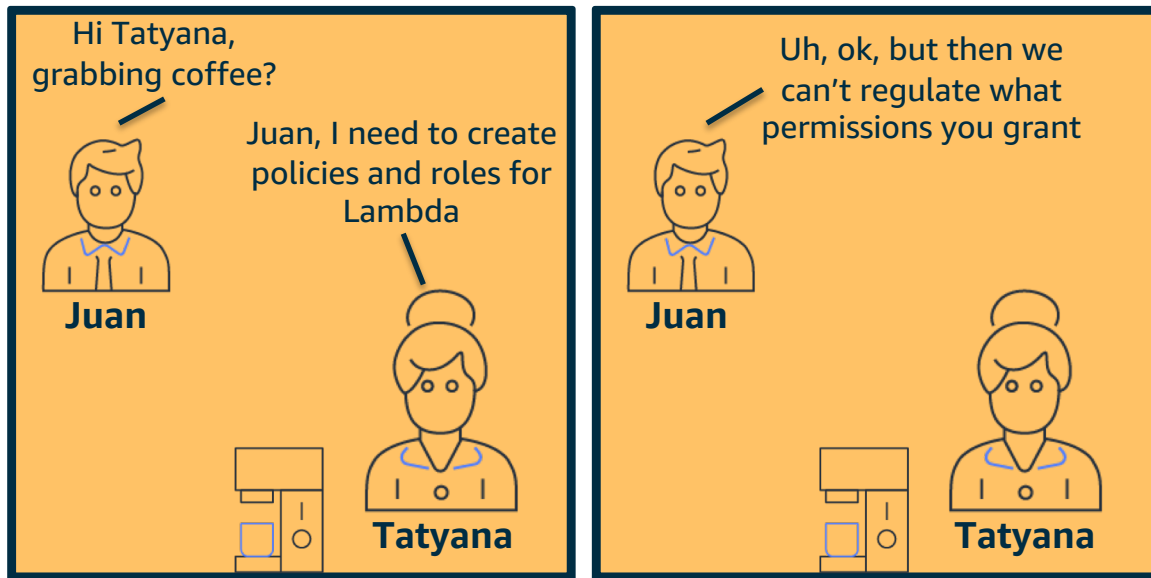
Method to safely grant actions like:
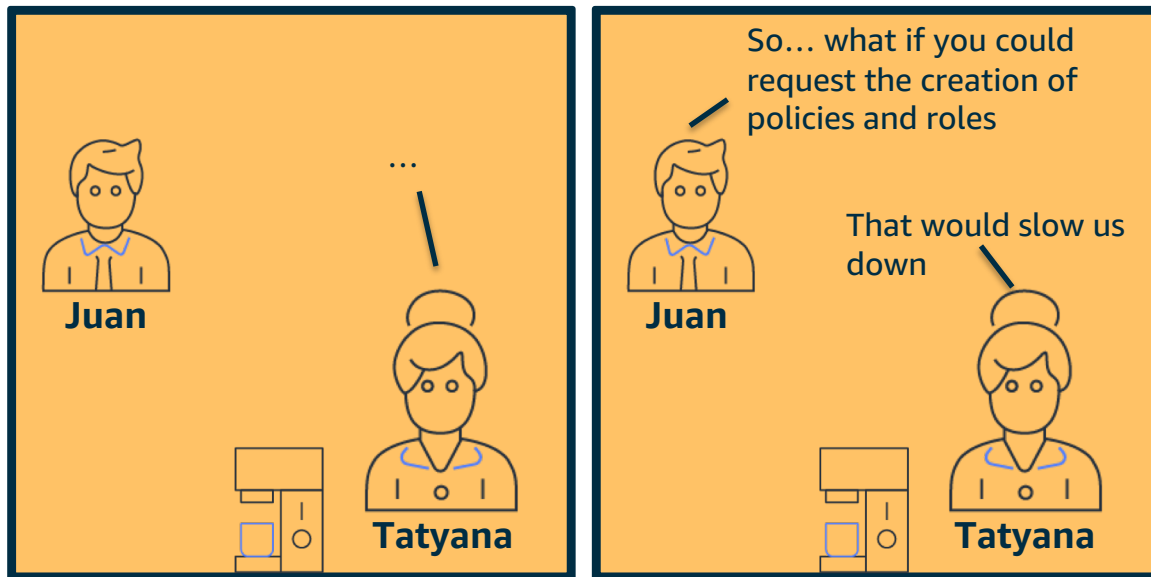"iam:CreateRole"
"iam:PassRole"

aws

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
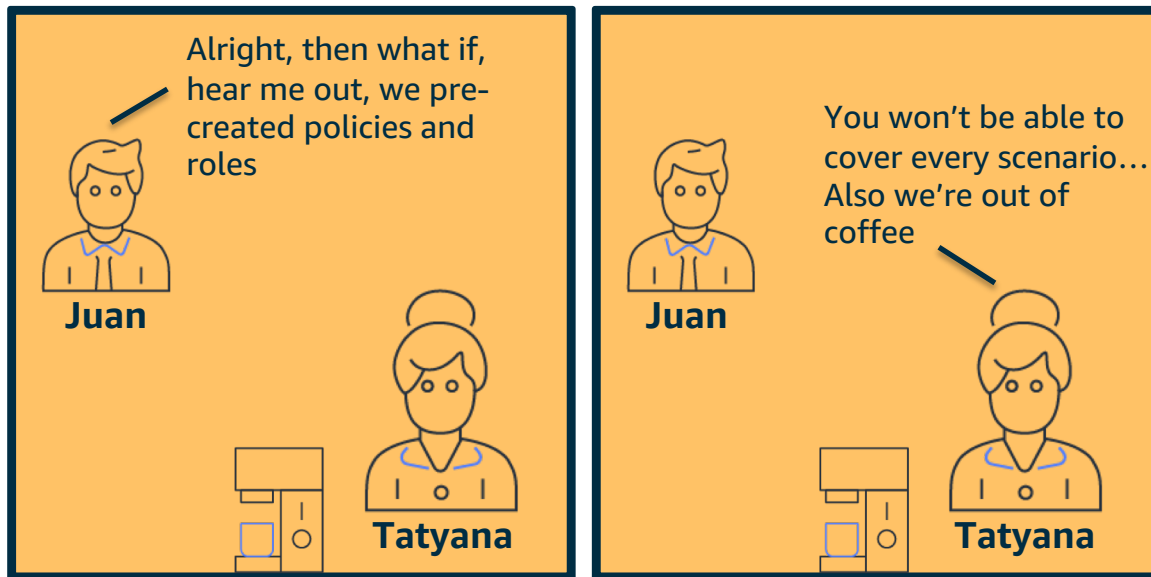
- Any others?

aws

Two viewpoints – the problem

Two viewpoints – old solution 1

Two viewpoints – old solution 2

Permission boundary basics

aws

Before and After Permission Boundaries

Before

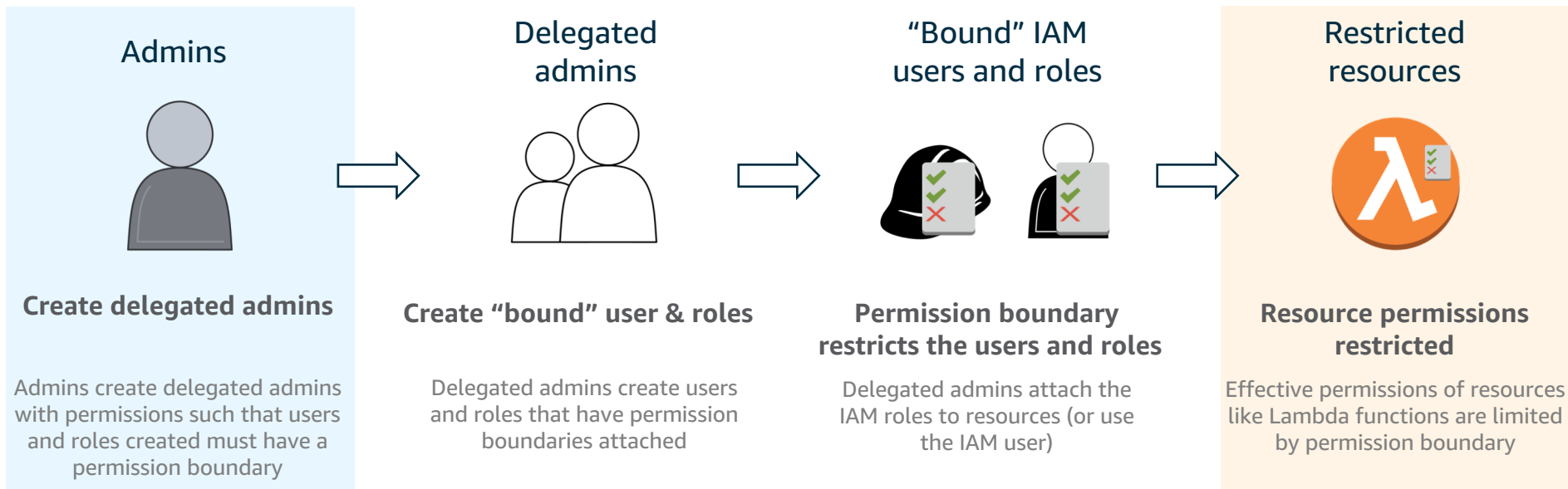- Certain IAM policy actions (e.g. PutUserPolicy, AttachRolePolicy) are essentially full admin-like permissions.

- Doing any form of self-service permissions management was non-trivial.

Now

- Administrator can grant full admin-like permissions, but specify a "permissions boundary."

- Allow developers to create principals for their applications and attach policies, but only within the boundary.

aws

Permission Boundaries – mechanism

Admins

Create delegated admins

Admins create delegated admins with permissions such that users and roles created must have a permission boundary

Delegated admins

Create "bound" user & roles

Delegated admins create users and roles that have permission boundaries attached

"Bound" IAM users and roles

Permission boundary restricts the users and roles

Delegated admins attach the IAM roles to resources (or use the IAM user)

Restricted resources

Resource permissions restricted

Effective permissions of resources like Lambda functions are limited by permission boundary

aws

A condition

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

aws

A condition 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

Mechanism

Developer creates a role for a Lambda function

 # Step 1: Create role

```
$ 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
```
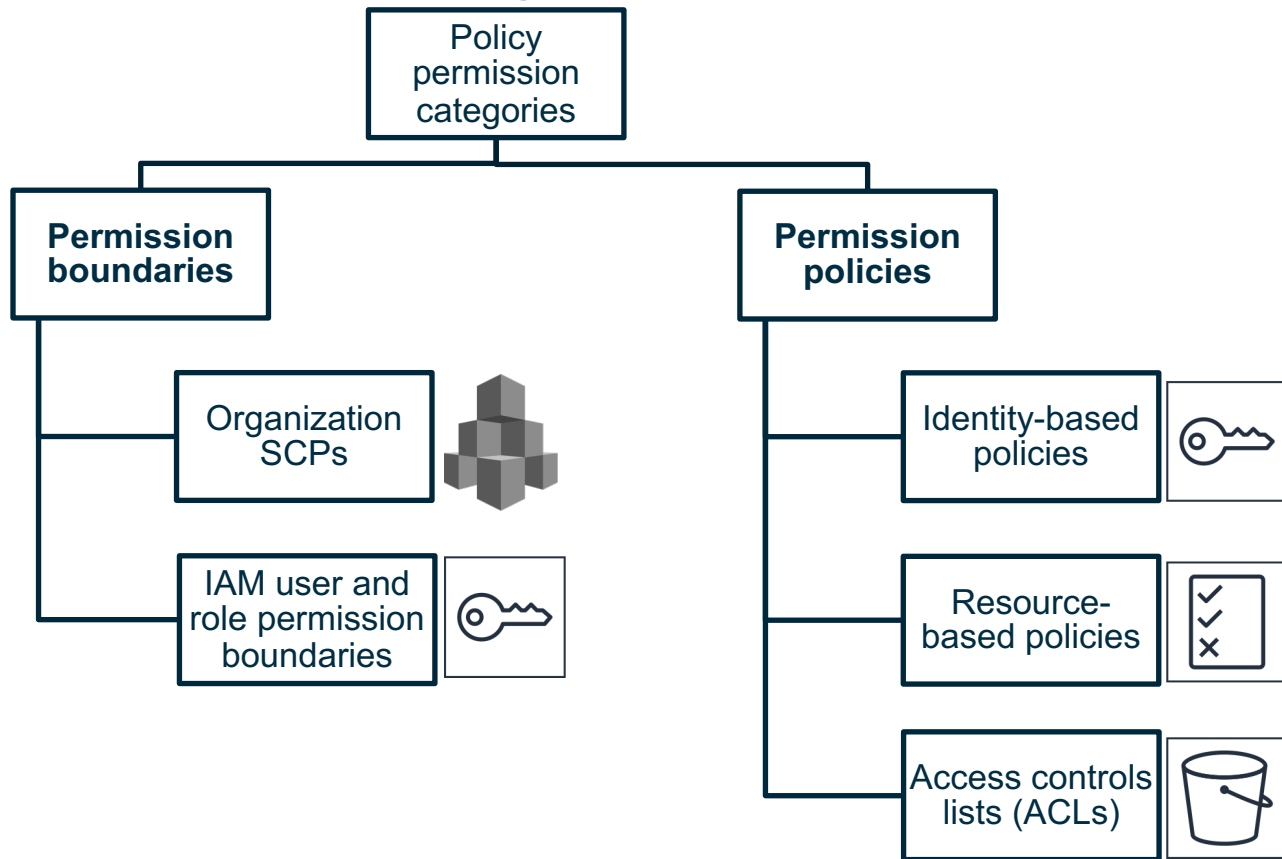
 # Step 2: Create policy
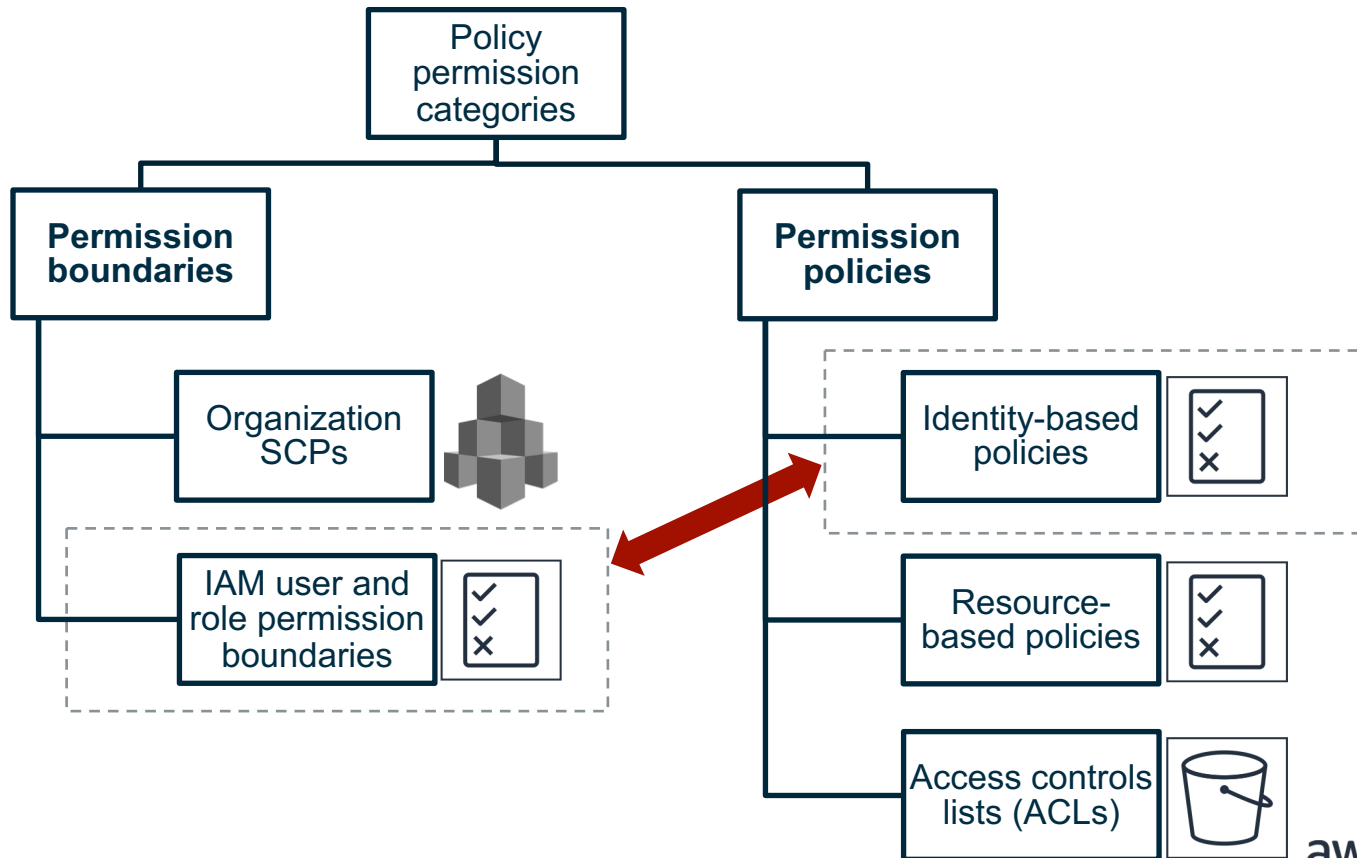
```
No change
```

 # Step 3: Attach policy

```
No change
```

aws

Demo

aws

Policy categories

Policy permission categories

Policy permission categories

Permission boundaries
- Organization SCPs
- IAM user and role permission boundaries

Permission policies
- Identity-based policies
- Resource-based policies
- Access controls lists (ACLs)

aws

Are these two things the same?

```
                    ┌─────────────────┐
                    │     Policy      │
                    │   permission    │
                    │   categories    │
                    └─────────────────┘
```

Permission boundaries

- Organization SCPs
- IAM user and role permission boundaries

Permission policies

- Identity-based policies
- Resource-based policies
- Access controls lists (ACLs)

aws

But, it's just an IAM policy right?

Identity-based policy slot

Identity-based policy

Permission boundary slot

Permission boundary

IAM policies

IAM role

aws

But, it's just an IAM policy right?
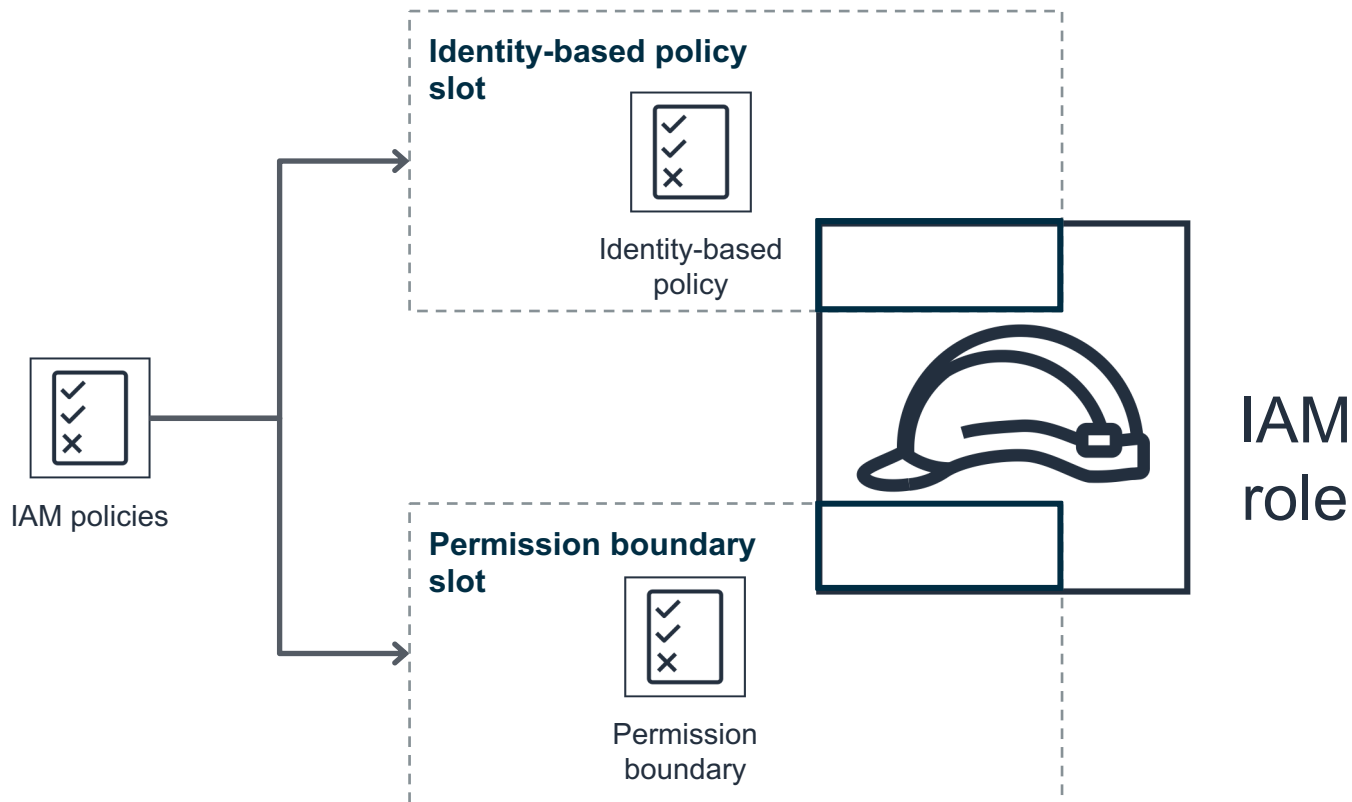
Identity-based policy slot

▾ Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy

Filter policies ⌄   🔍 Search   Showing 582 results

		Policy name ▾	Used as	Description
☐	▸ 📦	AdministratorAccess	Permissions policy (9)	Provides full access to AWS services an…
☐	▸ 📦	AlexaForBusinessDeviceSetup	*None*	Provide device setup access to AlexaFor…
☐	▸ 📦	AlexaForBusinessFullAccess	*None*	Grants full access to AlexaForBusiness r…
☐	▸ 📦	AlexaForBusinessGatewayExecution	*None*	Provide gateway execution access to Al…
☐	▸ 📦	AlexaForBusinessReadOnlyAccess	*None*	Provide read only access to AlexaForBu…
☐	▸	AllowAssumeDeleteDDBRole	*None*	
☐	▸	AllowDeleteofDDBTable	Permissions policy (1)	
☐	▸ 📦	AmazonAPIGatewayAdministrator	*None*	Provides full access to create/edit/delete…

Permission boundary slot

▾ Set permissions boundary

Set a permissions boundary to control the maximum permissions this role can have. This is an advanced feature used to delegate permission management to others. Learn more
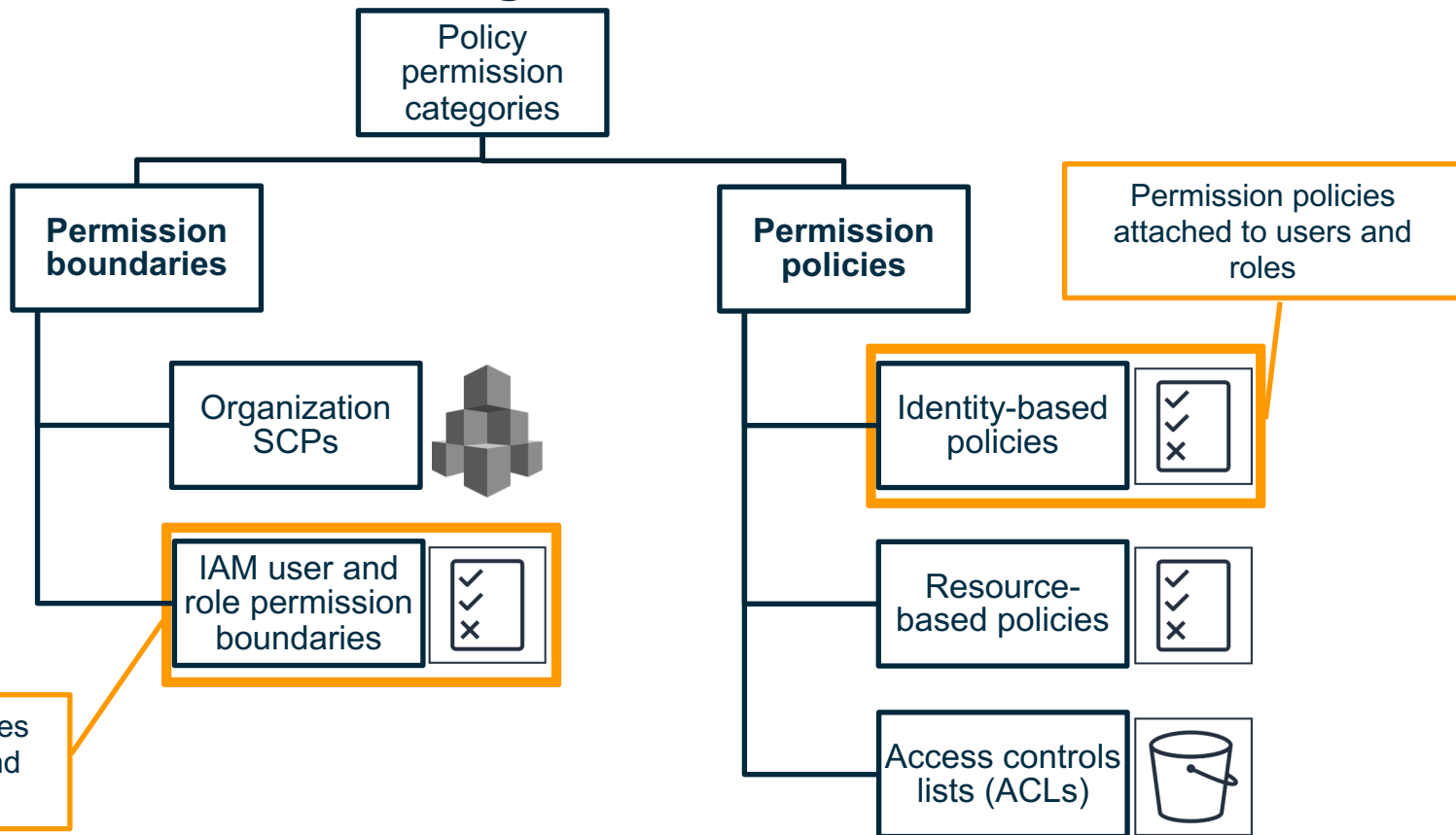
🔘 Create role without a permissions boundary
⚪ Use a permissions boundary to control the maximum role permissions

aws

Permission boundary mechanism

aws

Policy permission categories

Policy permission categories

Permission boundaries

Organization SCPs

IAM user and role permission boundaries

Permission boundaries attached to users and roles

Permission policies

Identity-based policies

Resource-based policies

Access controls lists (ACLs)

Permission policies attached to users and roles
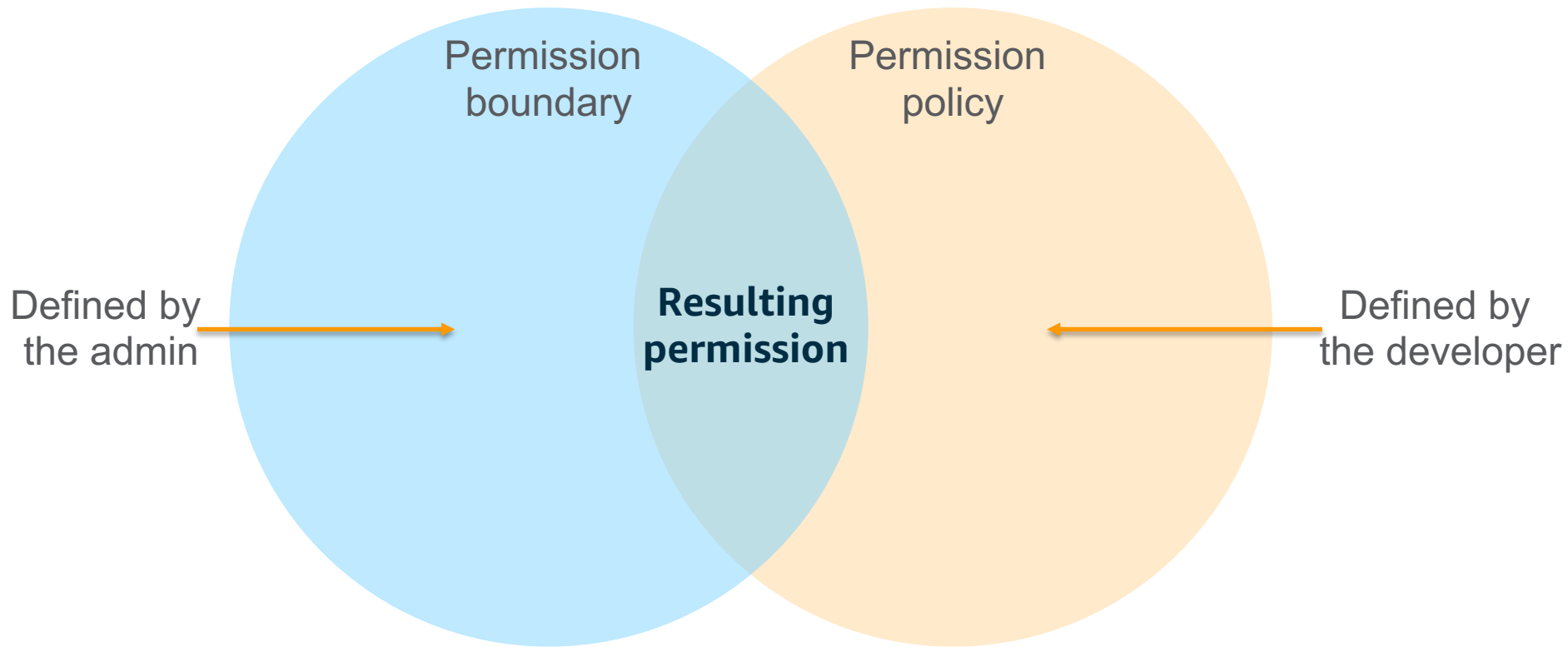
Everything after authentication

1. **Authenticate** the principal

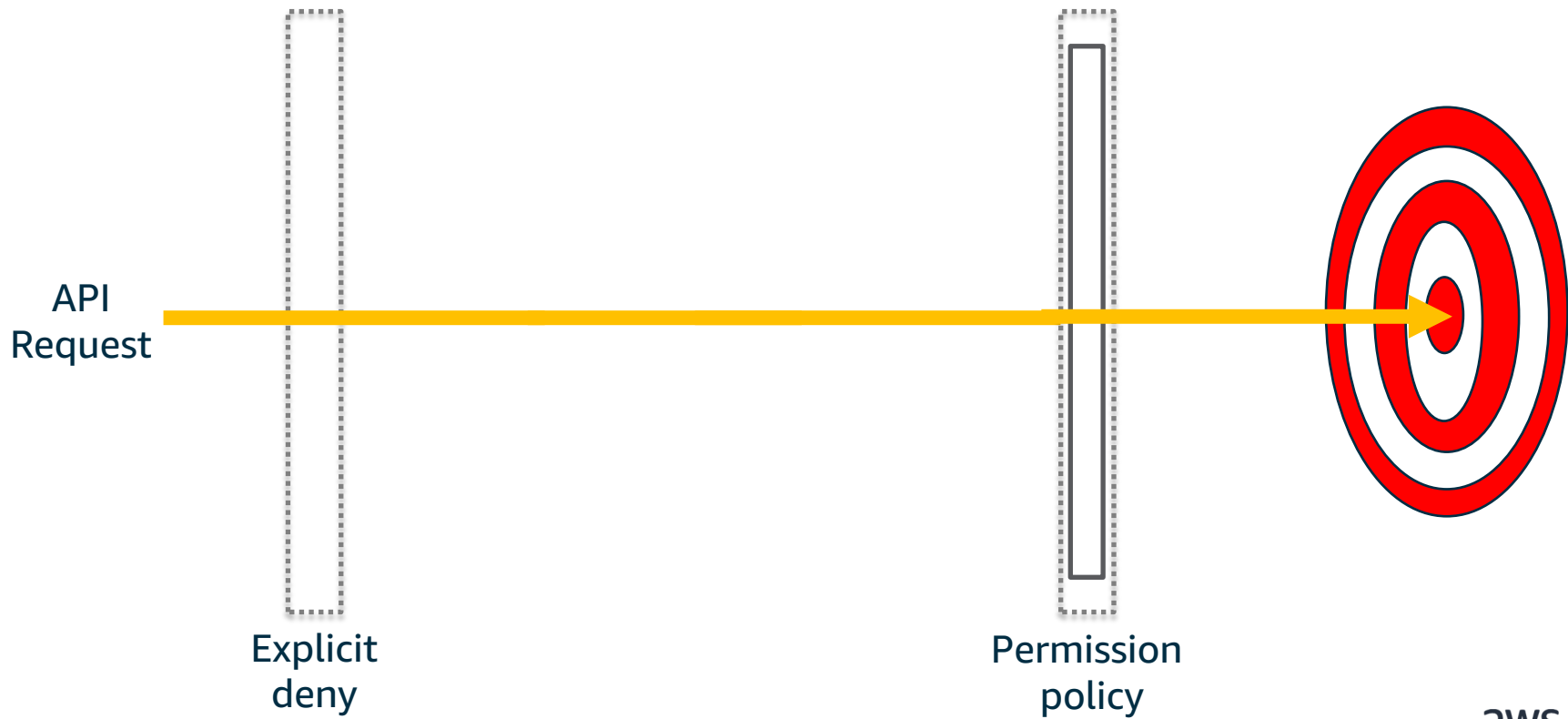2. Determine which **policies** apply to the request

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

4. **Allow or Deny** the request
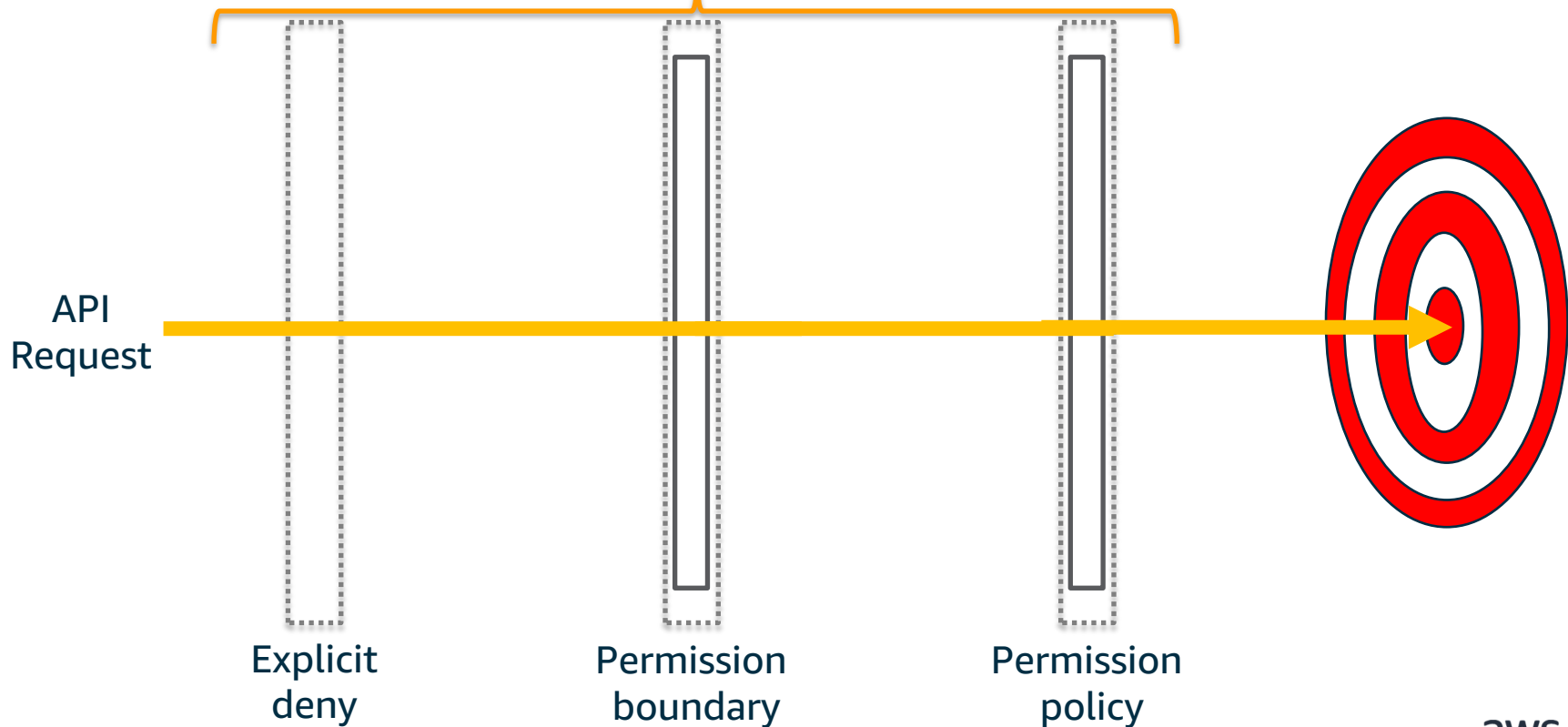
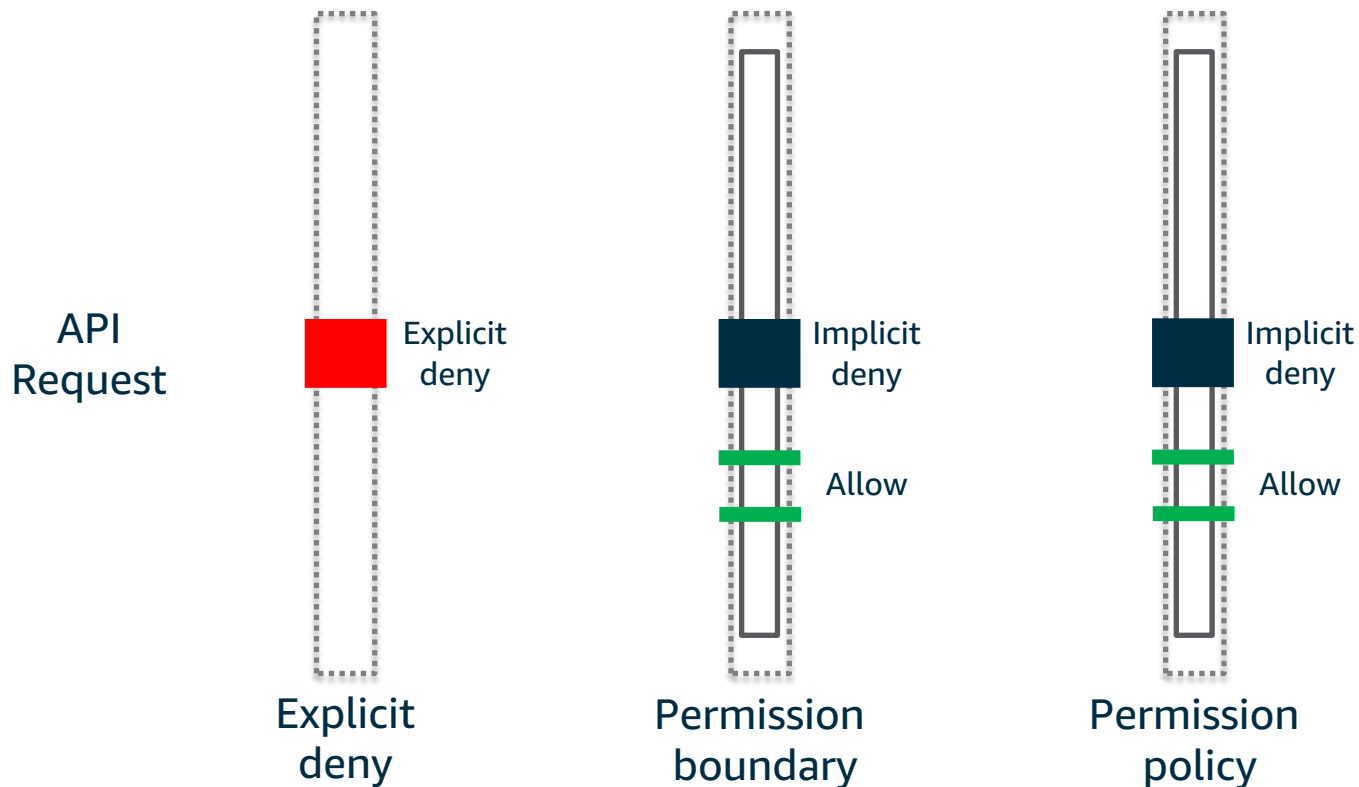aws

Effective Permissions - intersection

Permission boundary

Permission policy

Defined by the admin

Resulting permission

Defined by the developer

aws

Effective permissions – mechanism

API
Request

Explicit
deny

Permission
policy

aws

Effective permissions – mechanism

Resulting permission

API
Request

Explicit
deny

Permission
boundary

Permission
policy

aws

Effective permissions – mechanism

API
Request

Explicit
deny

Implicit
deny

Allow

Implicit
deny

Allow

Explicit
deny

Permission
boundary

Permission
policy

aws

Effective permissions – allow example

API
Request

Allow

Allow

Explicit
deny

Permission
boundary

Permission
policy

Request
allowed

aws
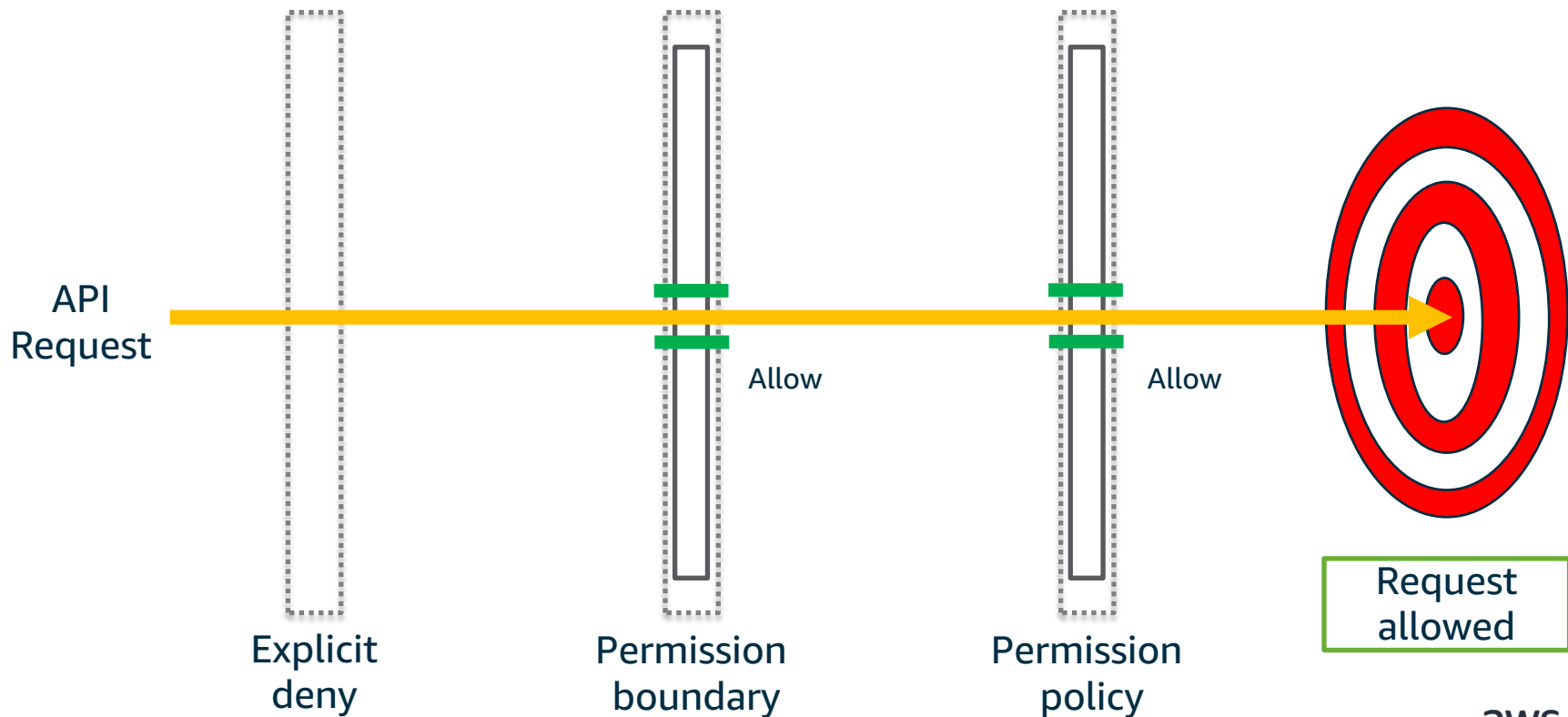
Effective permissions – scenario 1

Request: s3:GetObject / bucket name: example1

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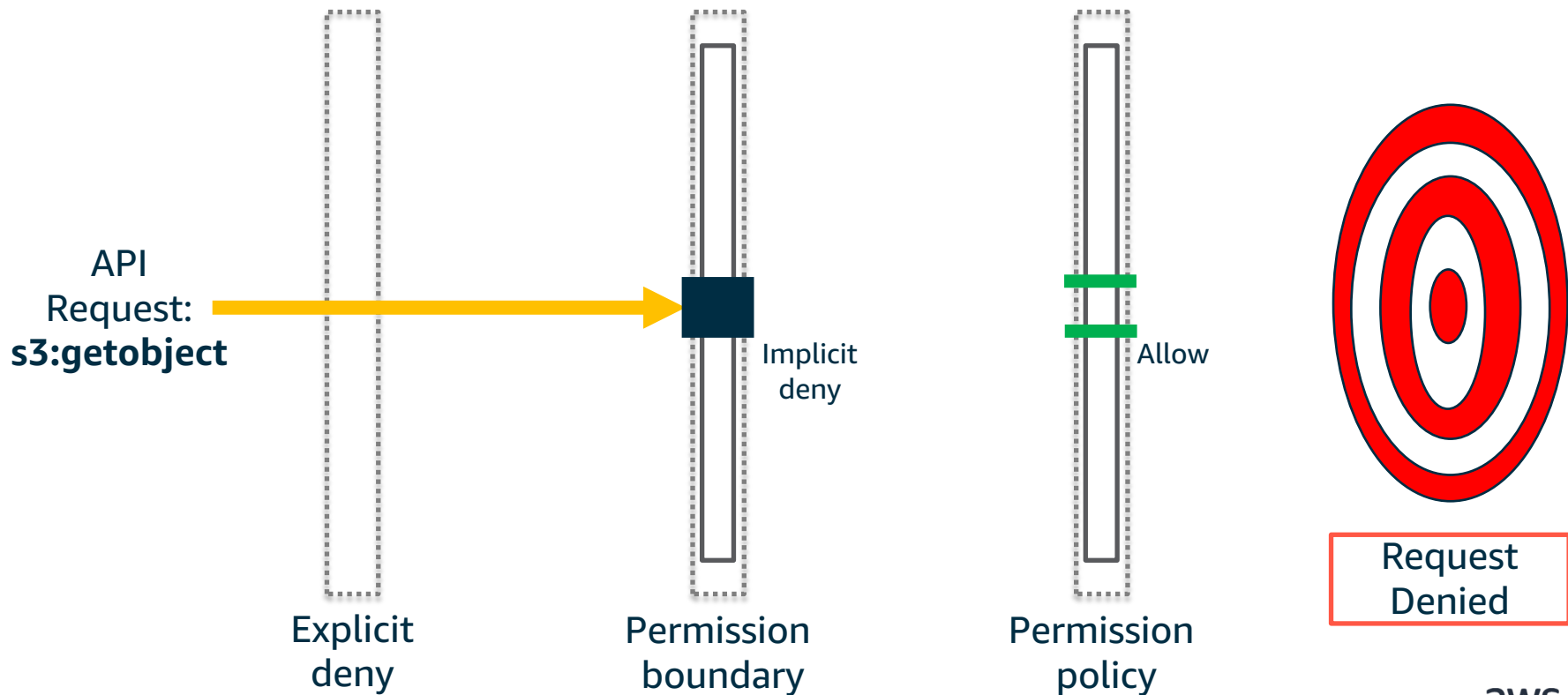
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:*"
      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

Permission Policy

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

aws

Effective permissions – result

API
Request:
s3:getobject

Explicit
deny

Implicit
deny

Permission
boundary

Allow

Permission
policy

Request
Denied

aws

Effective permissions – scenario 2

Request: s3:GetObject / bucket name: example1

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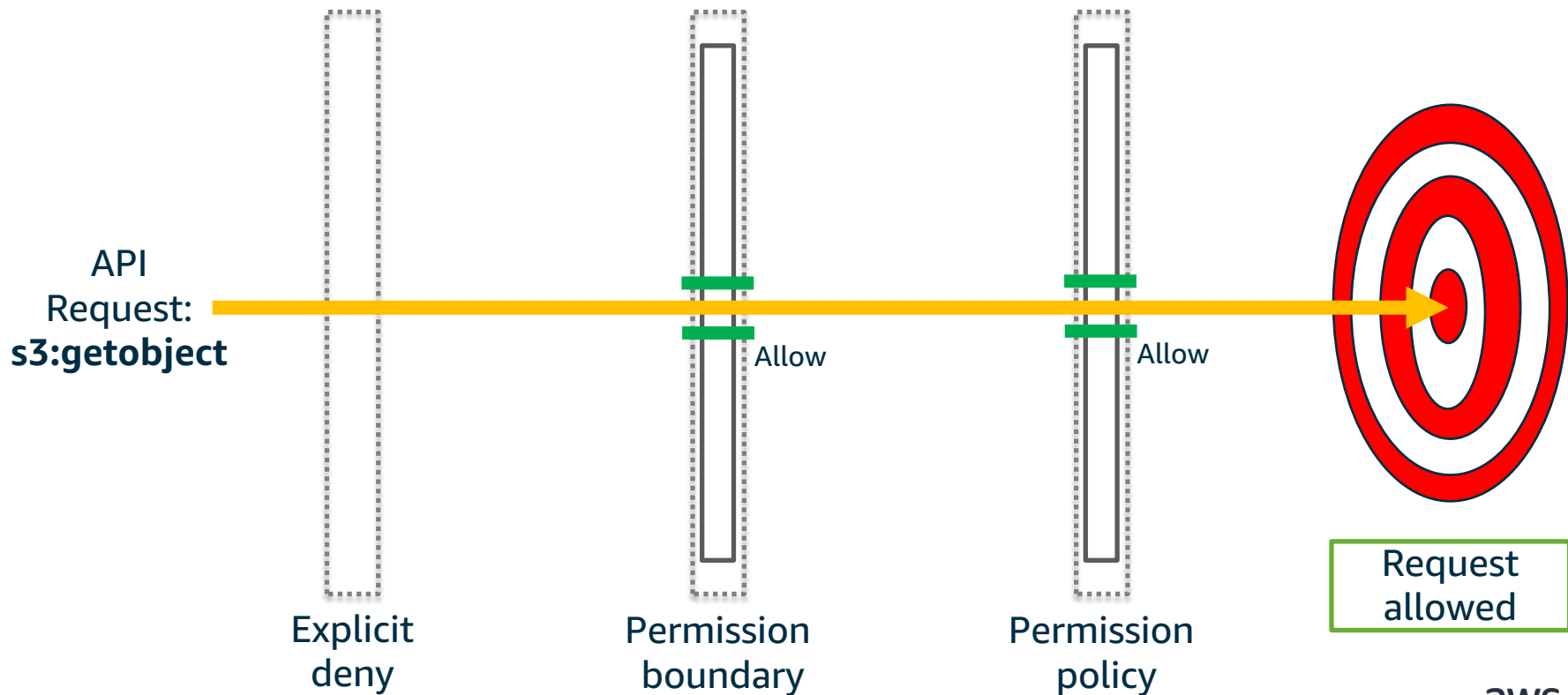
```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
              "logs:CreateLogGroup",
              "logs:CreateLogStream",
              "logs:PutLogEvents"
         ],
         "Resource": "arn:aws:logs:*:*:*"
      },
      {
         "Effect":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

Permission Policy

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

aws

Effective permissions – result

API
Request:
s3:getobject

Allow

Allow

Explicit
deny

Permission
boundary

Permission
policy

Request
allowed

aws

Effective permissions – scenario 3

Request: s3:GetObject / bucket name: example1

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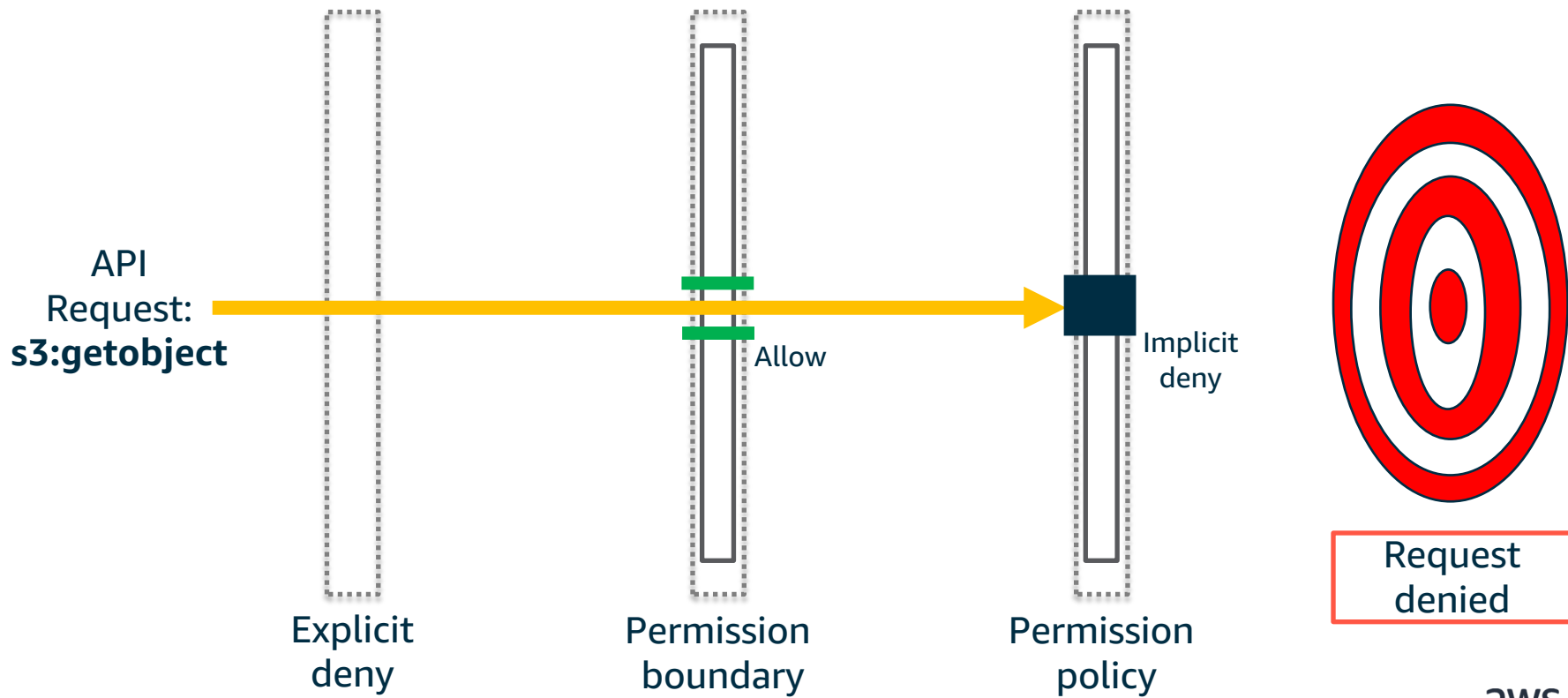
```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
              "logs:CreateLogGroup",
              "logs:CreateLogStream",
              "logs:PutLogEvents"
         ],
         "Resource": "arn:aws:logs:*:*:*"
      },
      {
       "Effect":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

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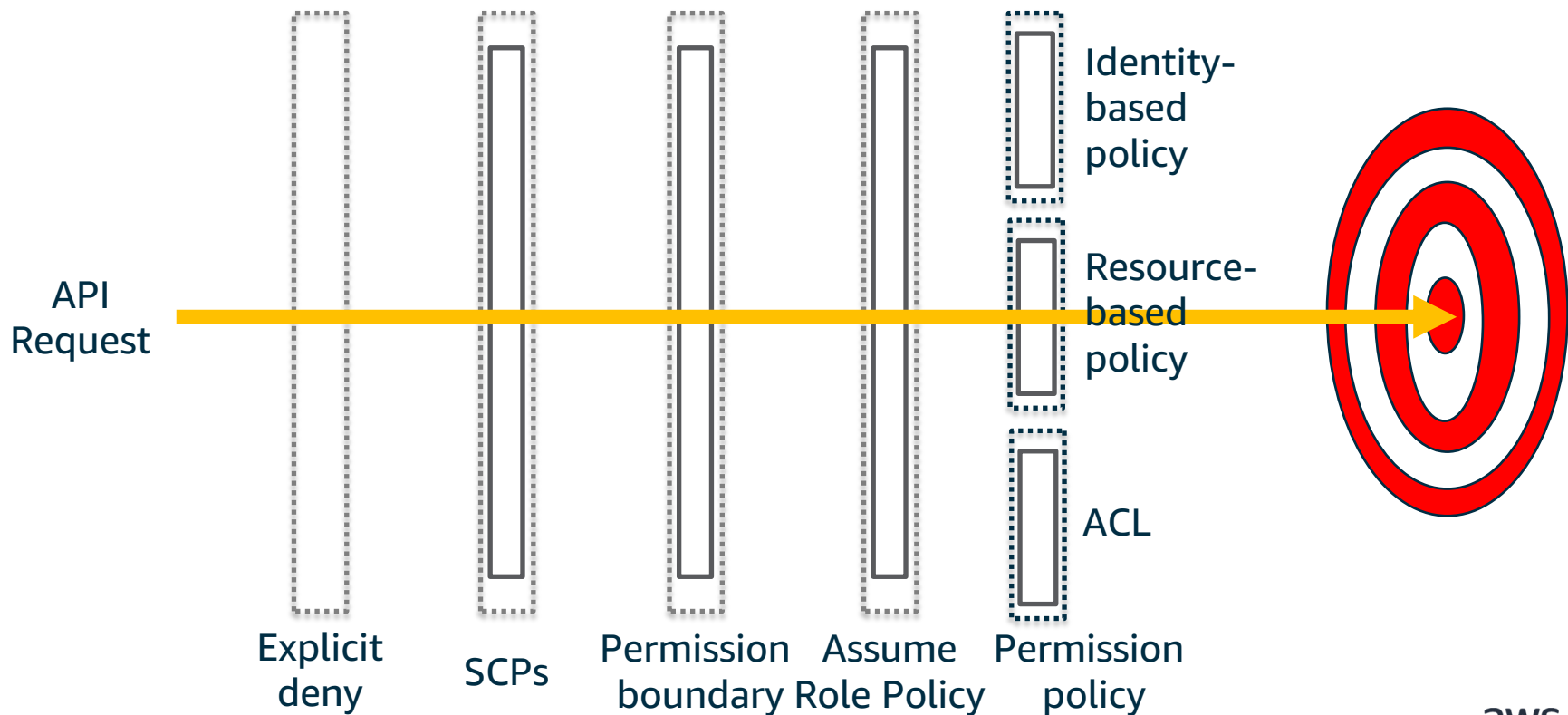
```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
              "logs:CreateLogGroup",
              "logs:CreateLogStream",
              "logs:PutLogEvents",
         ],
         "Resource": "*"
      }
   ]
}
```
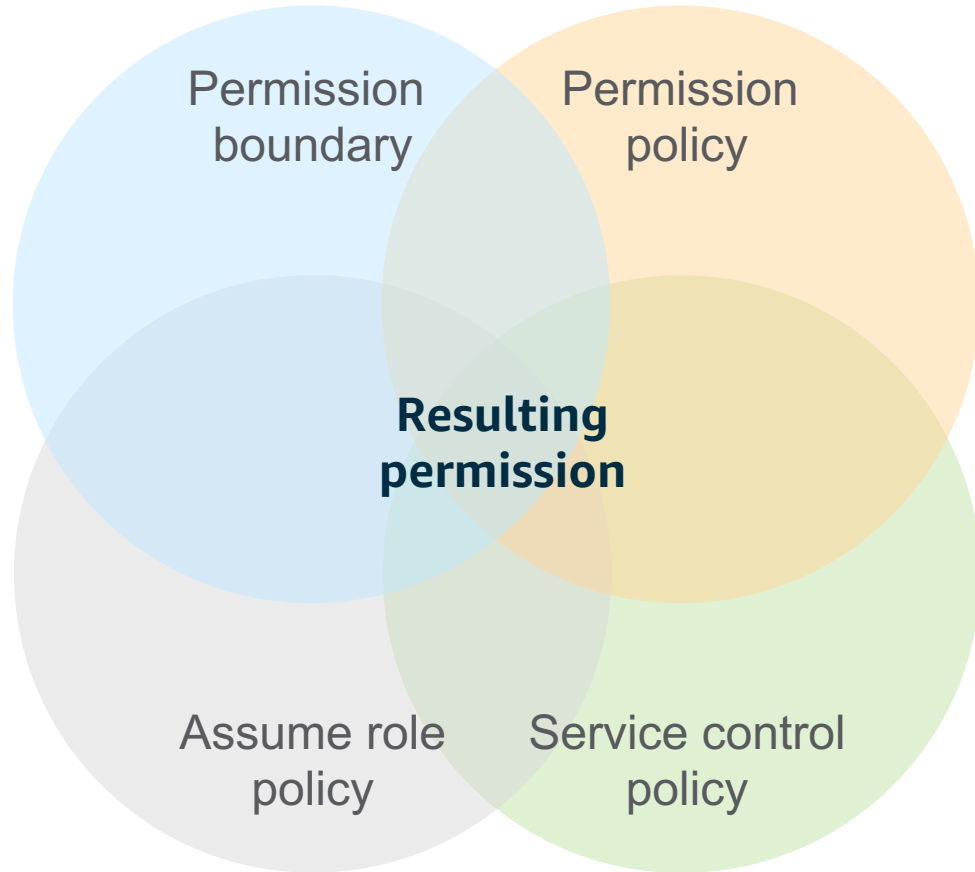
aws

Effective permissions – result

API Request:
s3:getobject

Allow

Implicit deny

Explicit deny

Permission boundary

Permission policy

Request denied

aws

Effective permissions – mechanism expanded

API Request

Explicit deny

SCPs

Permission boundary

Assume Role Policy

Permission policy

Identity-based policy

Resource-based policy

ACL

aws

Effective Permissions – intersection expanded

Resource Restrictions

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

Paths are preferred by request the CLI. Naming (name*) 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

aws

Resource Restrictions

- We are granting admin-like permissions (create and delete policies and roles)
- Permission boundary is one part of the delegation
- The other part is restricting the policies and roles they can impact then they could

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

aws

Resource Restrictions – examples

Resource restriction using paths:

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

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

Resource restriction using names:

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

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

aws

Resource Restrictions - policies

- Examine permissions assigned to a delegated admin to create 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

- Just like with policies we want to carve out a safe space for roles.
- Permission boundaries play a part here, but not all actions support the condition
- In addition different teams could be using the same permission boundaries

```
"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*"
```

aws

Presentation Q & A

aws

Presentation Q & A

- **How does a permission boundary differ from a standard IAM policy?**

- **What would happen if we delegated permissions without resource restrictions?**

- **The scenario where you have user in an account that need to be able to create IAM polices, roles and Lambda functions is common. How was this situation handled before permission boundaries came along?**

Final Q & A

Final Q & A

- What is the fundamental mechanism of permission boundaries?

- Why do we not allow the web admins to attach any role to the Lambda functions?

- There are two ways of doing resource restrictions (naming and pathing.) Which option allows you to create policies using both the AWS Console and CLI?

- Can an IAM policy be used as both a permission boundary and a permission policy? Is this a good practice?

- Is there an advantage to using one over the other?