AWS re:Inforce

JULY 26 – 27, 2022 | BOSTON, MA

IAM331

Best practices for delegating access on AWS

Liam Wadman
Solutions Architect, AWS Identity
Amazon Web Services

Michael Chan
Senior Solutions Architect, AWS Identity
Amazon Web Services

What we’ll cover

- Delegation in AWS Identity and Access Management (IAM)
- Service-linked roles (SLRs), service roles, and when to use them
- Using iam:PassRole to give IAM roles to Amazon Web Services resources
- Managing access to roles through role trusts
- IAM permissions boundaries for self-service access

General best practices for IAM

- Use AWS Single Sign-On (AWS SSO) or federation for human access
- Avoid using IAM users
- Use IAM roles
- Use a multi-account strategy
- Avoid using root users
- Apply least privilege

What is delegating access in AWS?

- Delegating access: giving an identity or service the ability to assume a role or the permissions to perform an action

Examples

- An AWS service or SAML-federated user assuming a role
- A resource policy granting permissions to a principal
- Passing a role to a service through iam:PassRole
- Creating an SLR for a service to assume

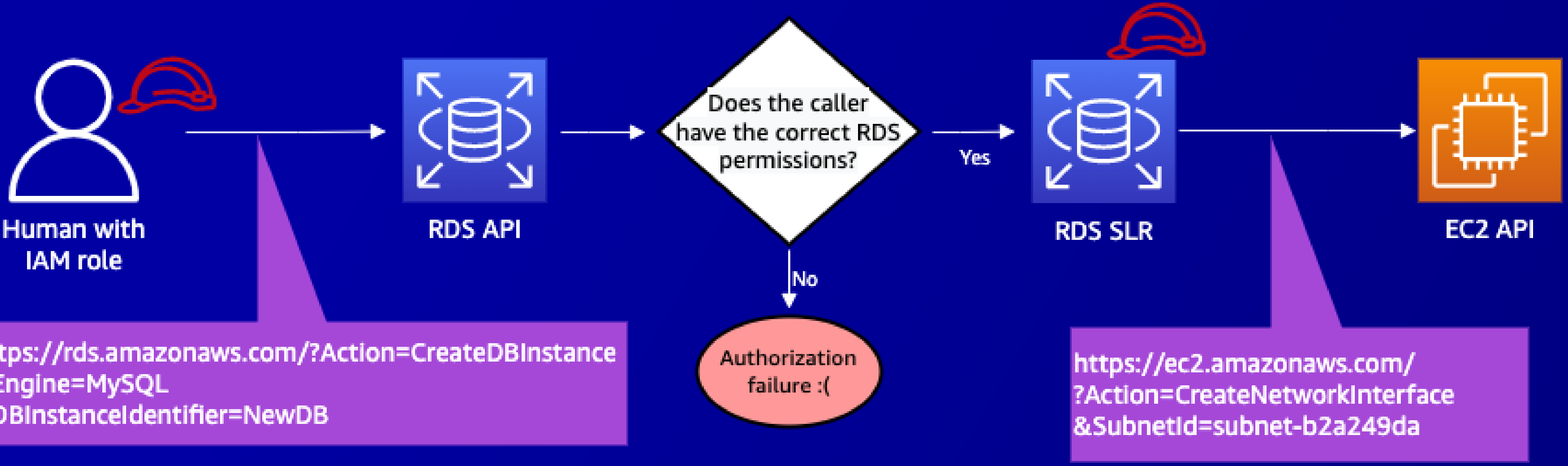
How do I let AWS services access my AWS resources securely?

You give them an IAM role

Service-linked roles (SLRs): What are they?

- A service-linked role is used by an AWS service to interact with resources in your account on your behalf
- Example: Amazon RDS has an SLR for managing networking
- The principal calling Amazon RDS does not need network entitlements
- Entitlements are predefined and managed by the service

SLRs: How they work



https://rds.amazonaws.com/?Action=CreateDBInstance
&Engine=MySQL
&DBInstanceIdentifier=NewDB

https://ec2.amazonaws.com/
?Action=CreateNetworkInterface
&SubnetId=subnet-b2a249da

SLRs: Best practices

- Focus on entitlements to AWS services, not entitlements of SLRs
- `iam:CreateServiceLinkedRole` allows developer self-service
- Where possible, use SLRs over service roles

Service roles: What are they?

- A service role is a role given to an AWS resource

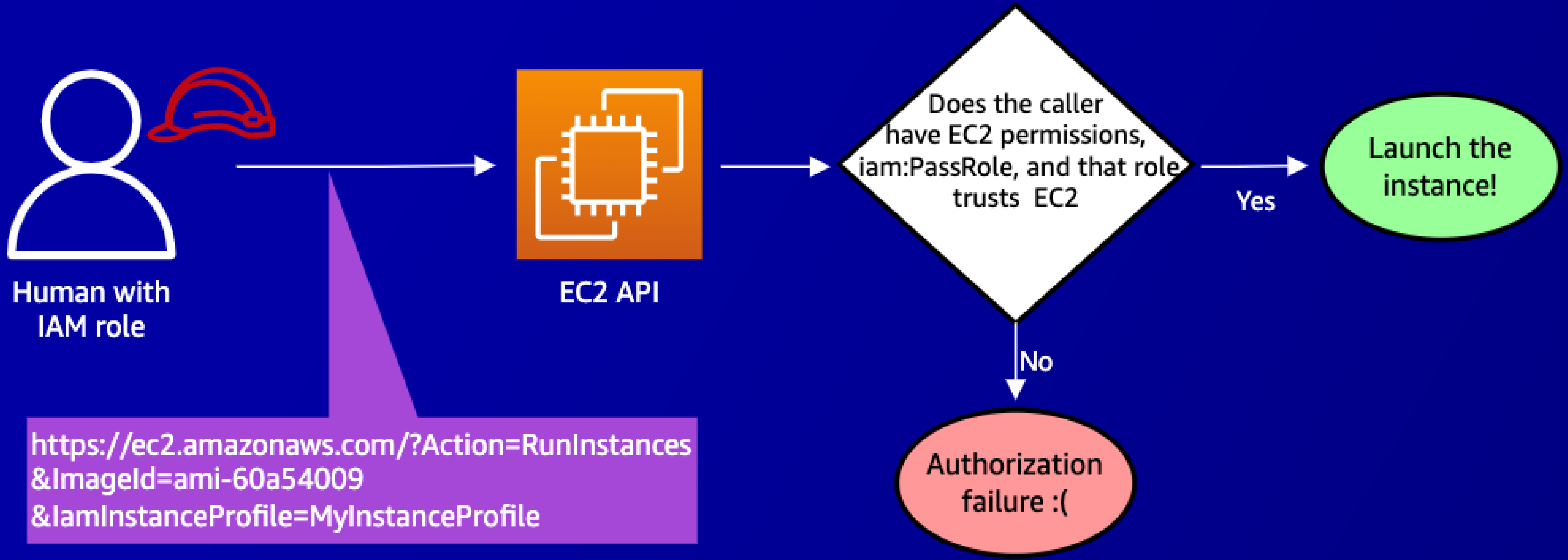
  Examples

  - Giving an AWS Lambda function a role to query your Amazon DynamoDB
  - Launching an Amazon EC2 instance with `ec2:RunInstances`
  - An Amazon S3 replication job uses a role to access your buckets
- Service roles have entitlements and trust policies managed by you
- Service roles are passed to AWS resources

IAM PassRole: What is it?

- An entitlement in AWS
- Allows creating resources with associated roles
- It is not an AWS API call
- It’s logged in the call that passed the role

IAM PassRole


Human with IAM role
EC2 API
Does the caller have EC2 permissions, iam:PassRole, and that role trusts EC2
Yes
Launch the instance!
No
Authorization failure :(

https://ec2.amazonaws.com/?Action=RunInstances
&ImageId=ami-60a54009
&IamInstanceProfile=MyInstanceProfile

IAM PassRole: The three truths

- A principal must have the `iam:PassRole` entitlement to pass a role
- The role's trust policy must allow the service to assume it
- The principal, role to be passed, and resource must be in the same account when passing

IAM PassRole: Best practices

Use **IAM paths** to constrain what roles can be passed

```
{
   "Sid": "AllowPassingAppRoles",
   "Effect": "Allow",
   "Action": "iam:PassRole",
   "Resource":"arn:aws:iam::*:role/approles/*"
}
```

IAM PassRole: Best practices

Use **wildcards** for account IDs in PassRole statements

```
{
    "Sid": "PrincipalTagInResourcePath",
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource":"arn:aws:iam::*:role/MyRole"
}
```

IAM PassRole: Best practices

Place PassRole entitlements in its **own policy statement**

```
{
   "Sid": "AllowPassingAppRoles",
   "Effect": "Allow",
   "Action": "iam:PassRole",
   "Resource":"arn:aws:iam::*:role/approles/*"
}
```

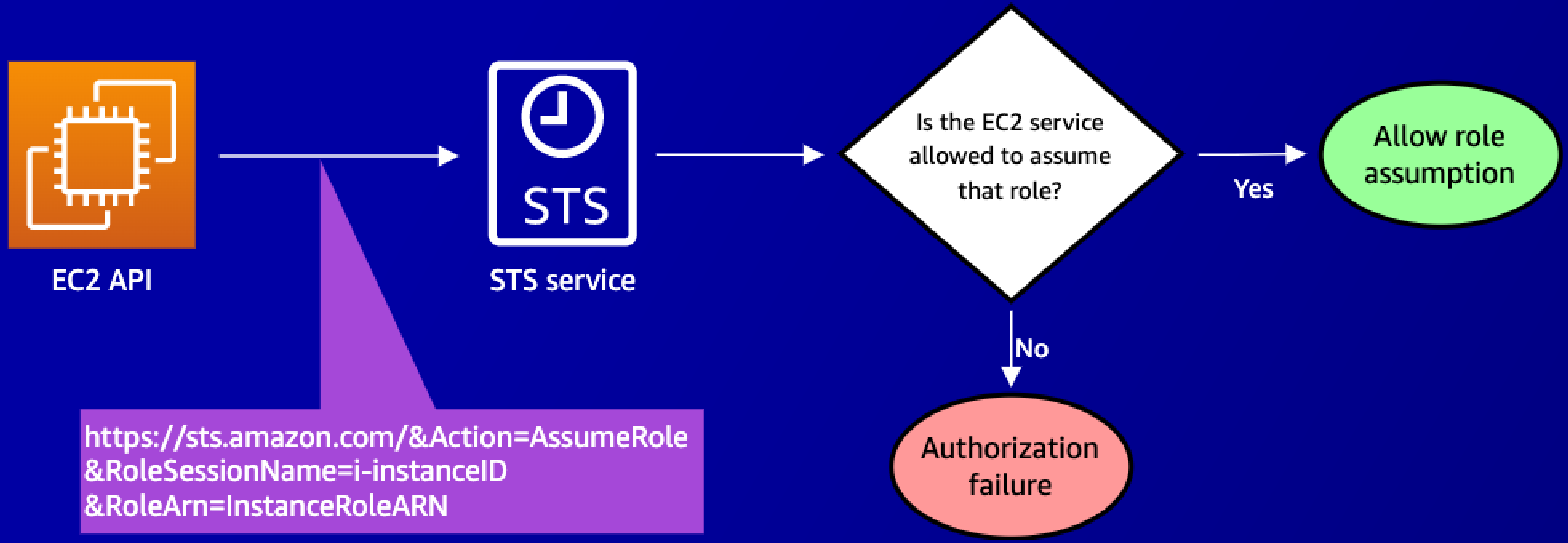
How does a human/workload/ AWS get access to an AWS role’s credentials?

They assume the role

AssumeRole: What is it?

- AssumeRole is an AWS API call that returns IAM role credentials
  - AWS Services assume service linked and service roles
  - AWS SSO/other identity providers (IdPs) assume roles to give humans access to AWS
  - Allow roles from other accounts to assume into your account
- Every role has a trust policy that says who can assume it

AssumeRole



```
https://sts.amazon.com/&Action=AssumeRole
&RoleSessionName=i-instanceID
&RoleArn=InstanceRoleARN
```

AssumeRole: Four truths

- IAM roles can assume other roles (role chaining)
- Roles assumed through SAML and OpenID Connect can only be assumed by IdPs in the same account
- AssumeRole events are logged in AWS CloudTrail
- The role trust policy controls who can assume and under what conditions

AssumeRole: Trust policy examples

Role Trust Policy allowing **cross-account**

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::444455556666:role/OtherRole"
  },
  "Action": "sts:AssumeRole"
}
```

AssumeRole: Trust policy examples

Allow a role with be **assumed with SAML**

```
{
  "Effect": "Allow",
  "Principal": {
    "Federated":"arn:aws:iam::111122223333:samlprovider/CorpSAML"
  },
  "Action": "sts:AssumeRoleWithSaml"
 }
```

AssumeRole: Trust policy examples

Enforce that a role **cannot be assumed from outside your organization,** **without denying AWS services access to the role**

```
{
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sts:AssumeRole",
  "Condition": {
    "StringNotEquals": {
      "aws:PrincipalOrgId": "${aws:ResourceOrgId}"
    },
    "BoolIfExists": {
      "aws:PrincipalIsAWSService": "false"
} } }
```

IAM role best practices

- Avoid role chaining within the same account
- Use Access Analyzer to detect cross-account role trusts
- Avoid using `:<account_id>:root` in trust policies
- Use dedicated roles for the different components of your workloads

Do you feel comfortable letting your developers create IAM roles and policies in self-service?

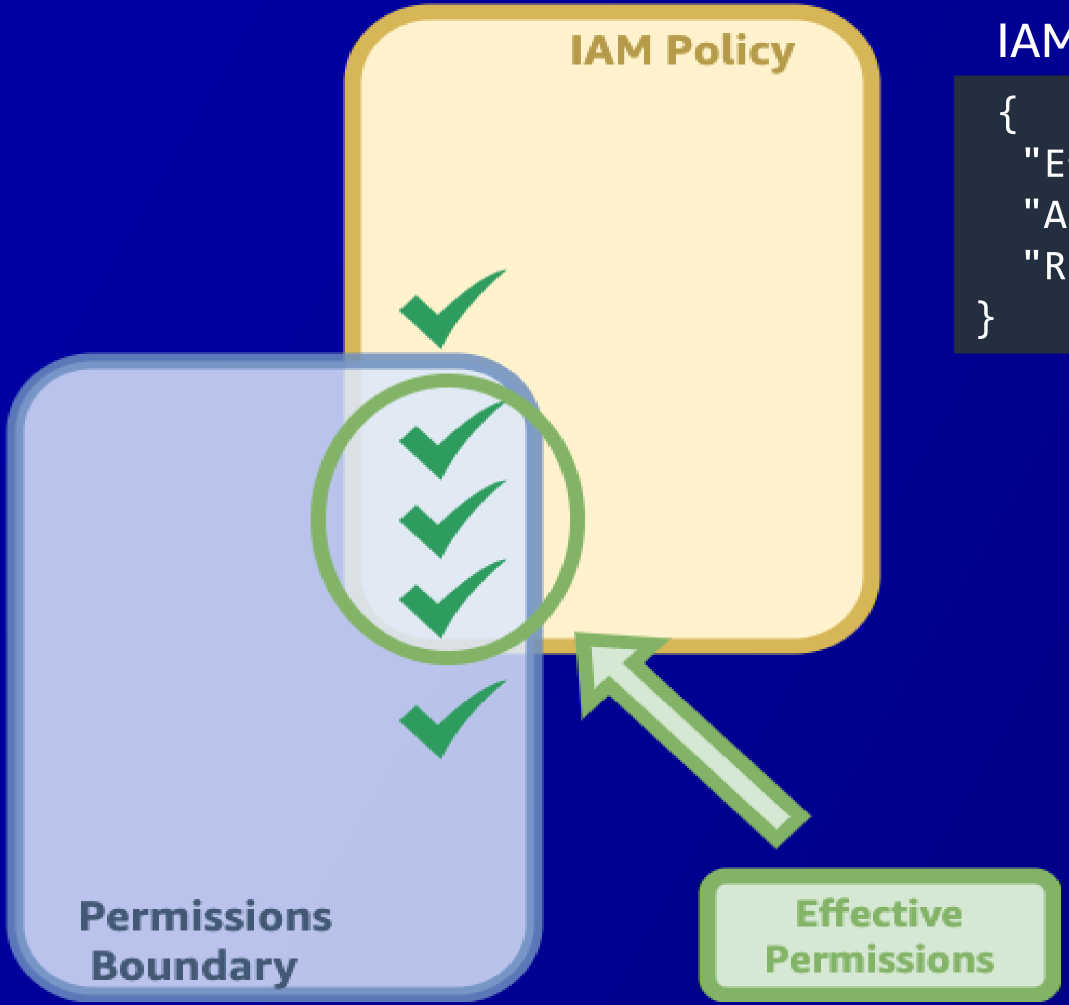
Why should I let my developers create IAM roles and policies in self-service?

How can I safely let my developers create IAM roles and policies in self-service?

Permissions boundaries: What are they?

- Permissions boundaries are attached to roles and limit their actions
  - They can explicitly deny actions with a Deny statement
  - They can implicitly deny actions with the lack of an Allow statement
  - They never grant an entitlement
- Use `iam:PermissionsBoundary` condition key to enforce usage
- Use the same language as permissions policies

What is the effective entitlement?



IAM Policy

```
{
  "Effect": "Allow",
  "Action": "*",
  "Resource": "*"
}
```

Permissions boundary

```
{
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "*"
}
```

What is the effective entitlement?

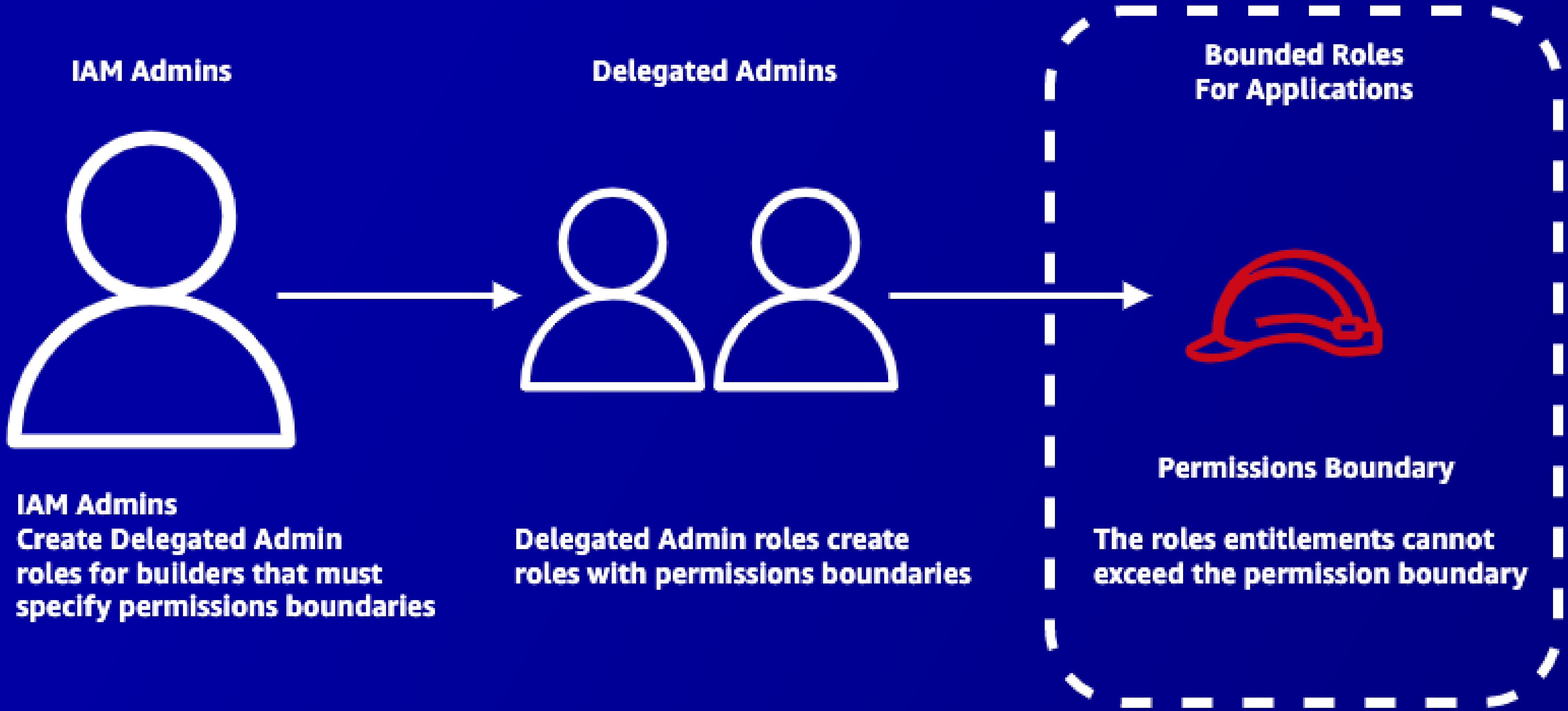
```
{
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "*"
}
```

How can I use permissions boundaries to delegate IAM access to my developers?

Create a permissions boundary for applications to do application-like actions

https://github.com/aws-samples/example-permissions-boundary

Identify your delegated admin roles

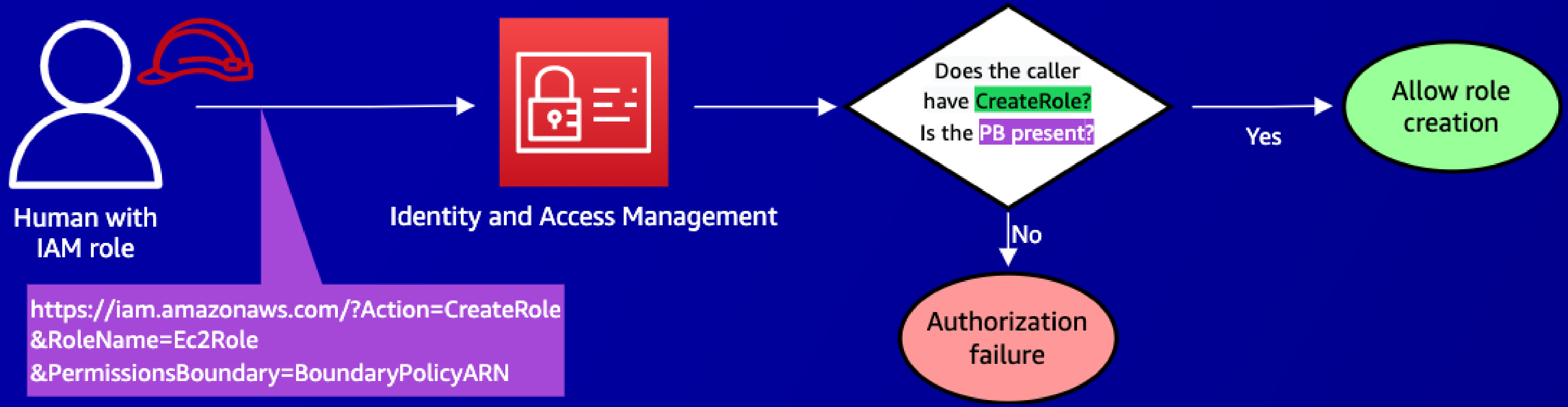


IAM Admins
Create Delegated Admin roles for builders that must specify permissions boundaries

Delegated Admin roles create roles with permissions boundaries

The roles entitlements cannot exceed the permission boundary

Enforce that permissions boundaries are used


Human with IAM role
Identity and Access Management
Does the caller have CreateRole? Is the PB present?
Yes
Allow role creation
No
Authorization failure

https://iam.amazonaws.com/?Action=CreateRole
&RoleName=Ec2Role
&PermissionsBoundary=BoundaryPolicyARN

```
{ "Effect": "Deny",
  "Action": [ "iam:CreateRole",
              "iam:PutRolePolicy"
              "iam:AttachRolePolicy" ],
  "Resource": "*",
  "Condition": {
    "StringNotLike": {
      "iam:PermissionsBoundary": "arn:aws:iam::*:policy/secure/permissionsboundarypolicy"
} } },
{ "Effect": "Allow",
  "Action": "iam:CreateRole",
  "Resource": "arn:aws:iam::*:role/applicationroles/*" }
```

Delegating access to developers

SCP mandating that all roles **have a permissions boundary**

```
{
  "Effect": "Deny",
  "Action": [ "iam:CreateRole",
              "iam:PutRolePolicy",
              "iam:AttachRolePolicy" ],
  "Resource": "*",
  "Condition": {
    "StringNotLike": {
      "iam:PermissionsBoundary":
        "arn:aws:iam::*:policy/secure/permissionsboundarypolicy"
    },
    "StringLike": {
      "aws:PrincipalArn": "arn:aws:iam::*:role/developer*"
} } }
```

Delegating access to developers

SCP **denying modification** of the permissions boundary policy

```
 {
  "Effect": "Deny",
  "Action": [
    "iam:DeletePolicy",
    "iam:CreatePolicyVersion",
    "iam:CreatePolicy",
    "iam:DeletePolicyVersion",
    "iam:SetDefaultPolicyVersion"
  ],
  "Resource": "arn:aws:iam::*:policy/secure/permissionsboundarypolicy",
  "Condition": {
    "StringLike": {
      "aws:PrincipalArn": "arn:aws:iam::*:role/developer*"
} } }
```

Delegating access to developers

IAM policy requiring all roles be under a **specific path**

```
{
  "Effect": "Allow",
  "Action": [ "iam:CreateRole",
              "iam:PutRolePolicy",
              "iam:AttachRolePolicy" ],
  "Resource": "arn:aws:iam::*:role/applicationroles/*"
}
```

Delegating access to developers

SCP **denying modification** of their own IAM roles

```
 {
  "Effect": "Deny",
  "Action": [
    "iam:PutRolePolicy",
    "iam:AttachRolePolicy",
    "iam:UpdateRole"],
  "Resource": "arn:aws:iam::*:role/developerroles/*",
  "Condition": {
    "StringLike": {
      "aws:PrincipalArn": "arn:aws:iam::*:role/developerroles/*"
} } }
```

Delegating access to developers

Constrain creation/modification of policies **in a specific path**

```
{
  "Effect": "Allow",
  "Action": [
    "iam:DeletePolicy",
    "iam:CreatePolicyVersion",
    "iam:CreatePolicy",
    "iam:DeletePolicyVersion",
    "iam:SetDefaultPolicyVersion"
  ],  "Resource": "arn:aws:iam::*:policy/applicationpolicies/*"
}
```

How do I start with permissions boundaries?

- Keep permissions boundaries wide and reusable
- Don’t include actions that applications shouldn’t need to do
  - PassRole, update IAM, modify virtual private clouds
- Reduce bounded IAM role permissions to what’s needed

Takeaways

- Manage who/what can pass/assume roles
- Use IAM paths to segregate roles and policies
- Enforce that permissions boundaries are used
- Protect your permissions boundaries
- Use SLRs instead of service roles where possible

Thank you!

Liam Wadman
liwadman@amazon.com

Michael Chan
mmch@amazon.com