1. What is the IAM Role?

An IAM role is an **IAM Identity** that you can create in your account that has specific permissions.

An IAM role is similar to an IAM user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS.

However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it.

Also, a **role does not have standard long-term credentials** such as a password or access keys associated with it. Instead, when you assume a role, it provides you with **temporary security credentials for your role session**.

When a user assumes a role, **temporary security credentials are created dynamically and provided to that user - making it a more secure way to grant access**

Roles are the primary way to **grant cross-account access**

IAM roles are **not associated** with a specific user or group. Instead, trusted entities assume roles, such as IAM users, applications, or AWS services such as EC2.

Modification of a Role can be done anytime and the changes are reflected across all the entities associated with the Role immediately

2. Why to use a Role?

It allows you to delegate access with defined permissions to trusted entities **without** having to **share long-term access keys**

Never share security credentials between accounts to allow users from another AWS Account to access resources in your AWS account. Instead, always use IAM Roles.

3. When to use the IAM Role?

- One AWS service accesses another AWS service – When an AWS service needs access to other AWS services or functions, you can create a role that will grant that access.
- One AWS account accesses another AWS account – This use case is commonly referred to as a cross-account role pattern. This allows human or machine IAM principals from other AWS accounts to assume this role and act on resources in this account.

- A third-party web identity needs access – This use case allows users with identities in third-party systems like Google and Facebook, or Amazon Cognito, to use a role to access resources in the account.
- Authentication using SAML2.0 federation – This is commonly used by enterprises with Active Directory that want to connect using an IAM role so that their users can use single sign-on workflows to access AWS accounts.

4. How to assume an IAM role?

You assume an IAM role by calling the **AWS Security Token Service (STS) AssumeRole APIs**

These are the 3 ways to assume a role:

1. AssumeRole
2. AssumeRoleWithWebIdentity
3. AssumeRolewithSAML

These APIs return a set of temporary security credentials that applications can then use to sign requests to AWS service APIs.

You can associate one IAM Role with an EC2 instance at this time.

5. Role involves defining two policies

Trust policy

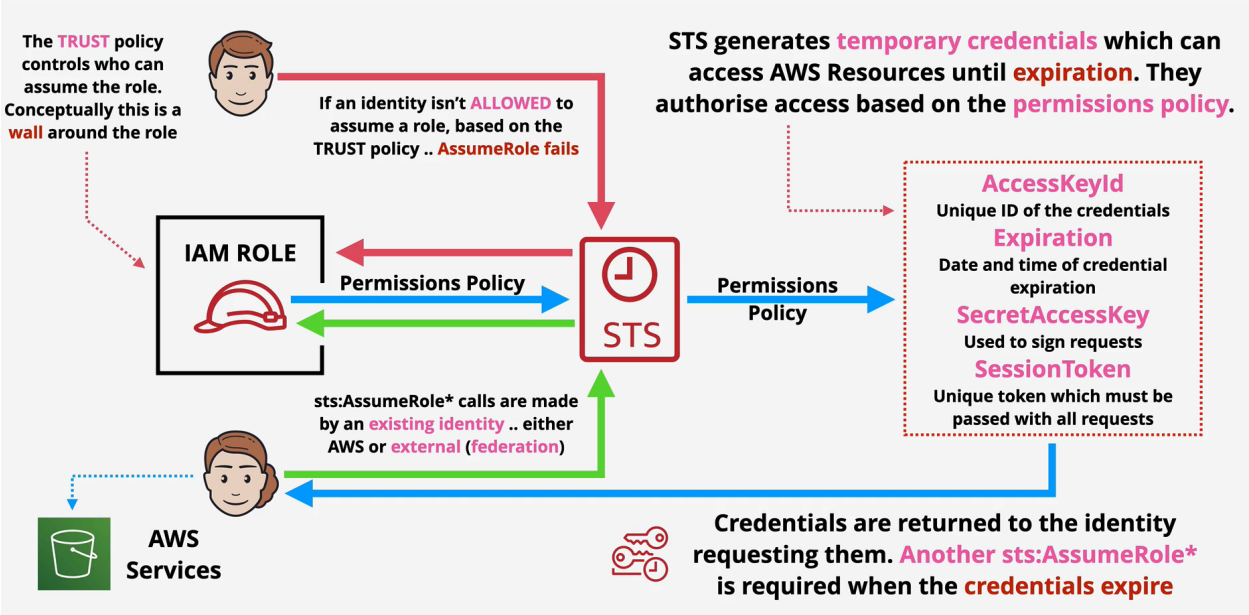
- Trust policy defines – who can assume the role
- Trust policy involves setting up a trust between the account that owns the resource (trusting account) and the account who owns the user that needs access to the resources (trusted account)

A role can be assumed by a human user or a machine principal, such as an Amazon Elastic Compute Cloud (Amazon EC2) instance or an AWS Lambda function.

Permissions policy

- Permissions policy defines – what they can access
- Permissions policy determines authorization, which grants the user of the role with the needed permissions to carry out the desired tasks on the resource

6. Steps to assume a role



Step 1: Bob and Julie are trying to assume a Role. So, they send STS:AssumeRole to STS Service

Step 2: First STS will check the TRUST POLICY in the IAM Role to verify the given identity has allowed permission to assume the role or not.

Step 3: In the above case, Bob is not allowed to assume the role based on the Trust Policy. So, Assume Role fails. However, Julie is allowed to assume the role based on Trust Policy. So, she can assume Role

Step 4: Once the Identity is allowed then STS will assume the Permission Policy from the given Role to check what are actions that given Identity can perform on the specific resource

Step 5: Based on the permission policy the STS will generate temporary credentials which can access AWS Resources until expiration. They authorize actions based on the permission policy.

Step 6: Credentials are returned to the identity requesting them. Another STS:AssumeRole is required when the credentials expire

NOTE:

Roles can be assumed by many identities. Who all get the same permissions.

You can NOT manually invalidate temporary credentials, it will only get expired

7. Revoking Temporary Credentials

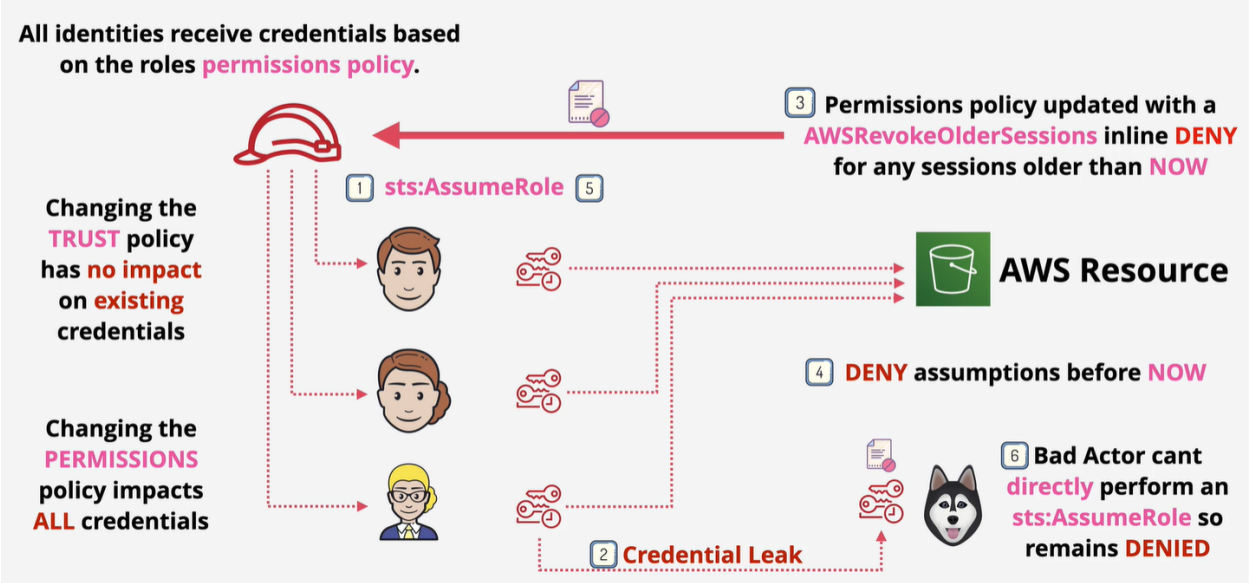
So, let's see we have the following 3 users and Bob, Julile, and Emy (the one with glass).

So, all 3 have assumed a role and accidentally Emy committed her temporary credentials in github.

And that data is consumed by a bad guy woofy.

Now, with the temporary credentials he can access the AWS Resources.

How can you implement a design that will not affect the other users access, but deny the woofy user?



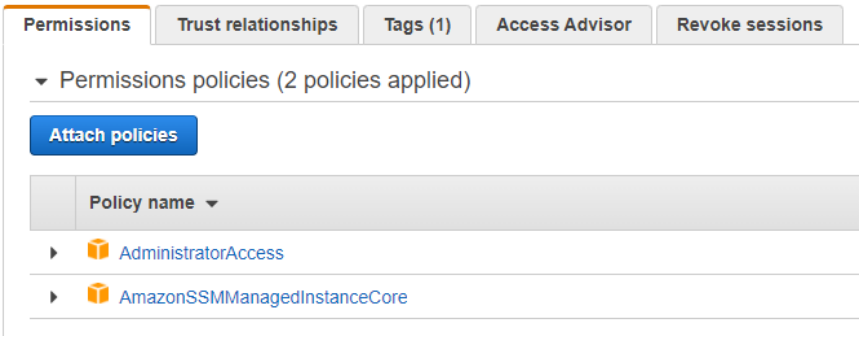
Step 1: Changing Trust Policy won't make any difference since woofy already has the temporary security credentials. So, he doesn't need to get validated again with Trust Policy to get the temporary credentials

Step 2: Changing Permissions Policy seems to be solving an issue. However, it will impact all the other existing users. We don't want that based on the given scenario

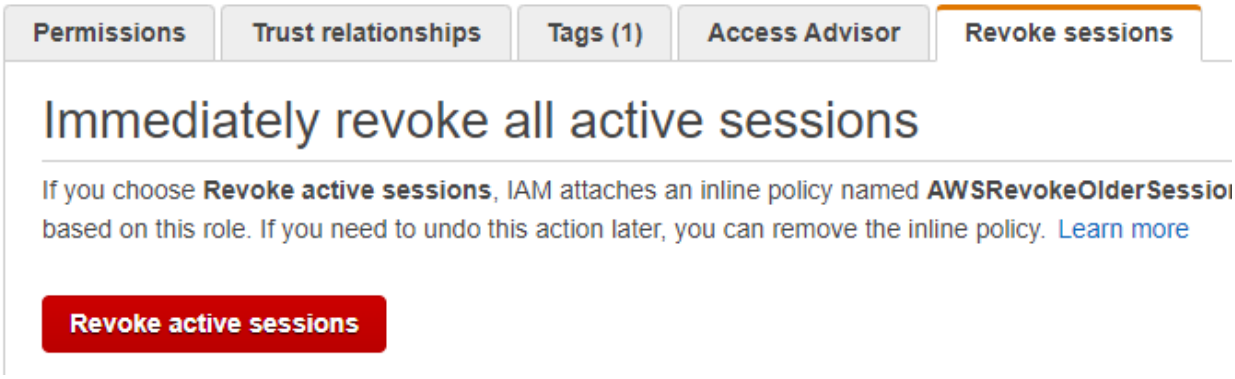
Step 3: If we can update the **Permission policy** with **AWSRevokeOlderSessions inline DENY** for any sessions older than **NOW** will resolve the issue.

Step 4: Now woffy can't use the existing credentials to access the resource due to explicit deny also he can't assume a new Role since he is not under Trust Policy. So, he will be loosing the access to access any AWS Resources

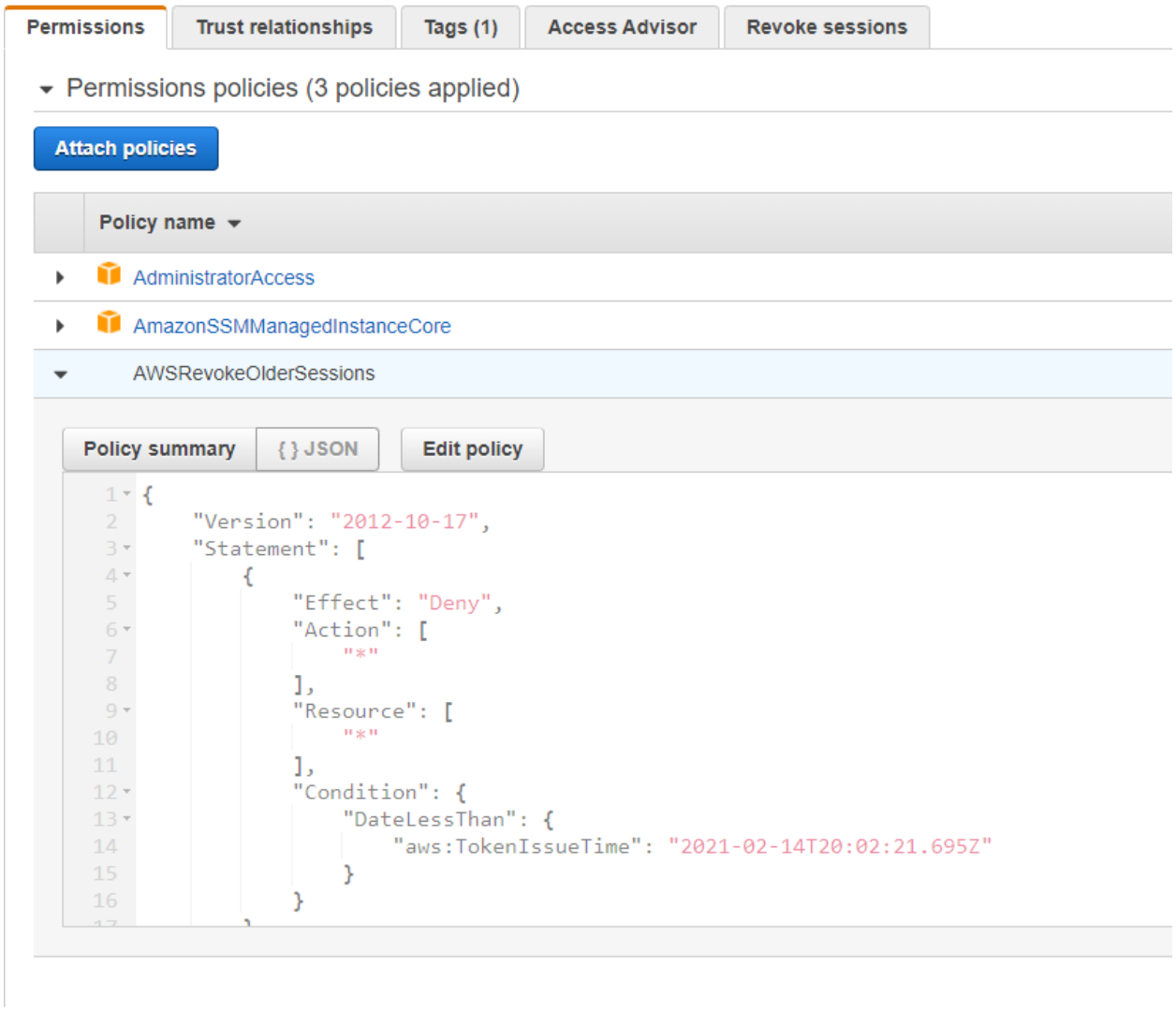
Points to remember: Before fix



Solution: Attache Revoke Session



After fix:



6. IAM Role Limitations

You can **NOT** associate **more than ONE IAM role with an EC2 instance**.

This limit of one role per instance can not be increased

You can NOT add an IAM Role to an **IAM Group**

You **CAN NOT** attach an EC2 IAM Role to **on-premises servers.**

Refer to the following question.

Question: I have an on-premises personal server that I would like to use to perform AWS API calls.

a. **I should run 'aws configure' and put my credentials there. Invalidate them when I am done - Correct**

b. **I should attach an EC2 IAM Role to a personal server - WRONG ANSWER. You can't attach EC2 IAM roles to on premise servers**

8. Granting a user permissions to pass a role to an AWS service

Example:

Imagine that you want to grant a user the ability to pass any of an approved set of roles to the Amazon EC2 service upon launching an instance. You need three elements:

1. An **IAM *permissions policy*** attached to the **role** that determines what the role can do. Scope permissions to only the actions that the role must perform, and to only the resources that the role needs for those actions. You can use AWS managed or customer-created IAM permissions policy.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [ "A list of the permissions the role is allowed to use" ],
        "Resource": [ "A list of the resources the role is allowed to access" ]
    }
}
```

2. A ***trust policy*** for the **role** that allows the service to assume the role.
   For example, you could attach the following trust policy to the role with the `UpdateAssumeRolePolicy` action. This trust policy allows Amazon EC2 to use the role and the permissions attached to the role

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "TrustPolicyStatementThatAllowsEC2ServiceToAssumeTheAttachedRole",
        "Effect": "Allow",
        "Principal": { "Service": "ec2.amazonaws.com" },
       "Action": "sts:AssumeRole"
    }
}
```

3. An **IAM *permissions policy*** attached to the **IAM user** that allows the user to pass only those roles that are approved.
   `iam:PassRole` usually is accompanied by **`iam:GetRole`** so that the user can get the details of the role to be passed. In this example, the user can pass only roles that exist in the specified account with names that begin with `EC2-roles-for-XYZ-`:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::<account-id>:role/EC2-roles-for-XYZ-*"
    }]
}
```

9. What is Service Role

IAM roles that can be assumed by an AWS service are called *service roles*.

Service roles must include a **trust policy**.

Trust policies are resource-based policies that are attached to a role that define which principals can assume the role.

A *service principal* is an identifier that is used to grant permissions to a service.

The identifier for a service principal includes the service name, and is usually in the following format: service-name.amazonaws.com

The service principal is defined by the service. You can find the service principal for some services by opening AWS services that work with IAM

The following example shows a policy that can be attached to a service role.

The policy enables two services, Amazon ECS and Elastic Load Balancing, to assume the role.

The services can then perform any tasks granted by the permissions policy assigned to the role (not shown).

To specify multiple service principals, you **do not specify two** `Service` elements; you can have only one. Instead, you use an array of multiple service principals as the value of a single `Service` element.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "ecs.amazonaws.com",
          "elasticloadbalancing.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

10. Granting an AWS Service to pass a role to an AWS service