Role vs Claim vs Permission based Access Control Explained

I will try to explain the Role/Claim/Permission-based Access Control concept in layman's terms. The code snippet I will present here, are pseudocode, may or may not compile.

**What are Roles?**

Roles can be thought of as Job Titles. Like "Sales Manager", "Marketing Manager", "Admin" etc.

**What are the claims?**

Claims can be broader than a Role. You can think about Claim as a TAG. For example, you can tag a person as "Friendly", "Talkative", "European", "Photographer", "Adult-who-is-18-years-old" etc. Technically, a role can be thought of as a claim too.

**Role-Based Access Control**

Very simple. Instead of using words, let's show some examples. Say, you allow visiting some pages on your website by checking Roles. Like this:

[Authorize(Roles="Sales Manager")]

public ActionResult CreateCustomer()

{

return View();

}

[Authorize(Roles="Marketing Manager")]

public ActionResult EditLandingPage()

{

return View();

}

**Claims Based Access Control**

In layman's terms, in Claims Based Access control, you check for claims instead of a role when determining access to a page.

(This is a pseudo code. ClaimsAuthorize is not a built-in class in MVC, rather, you may find some NuGet packages for that or you can write your own)

[ClaimsAuthorize(Claims="Senior-Employee, Award-Winner-Employee, Experienced-On-Sales")]

public ActionResult CreateCustomer()

{

return View();

}

[ClaimsAuthorize(Claims="Trust-worthy-Employee, President")]

public ActionResult DeleteCustomer()

{

return View();

}

[ClaimsAuthorize(Claims="Adult-over-18years")]

public ActionResult ViewImagesOfViolence()

{

return View();

}

Notice that, instead of checking for roles, we are allowing to visit a page based on WHO the user is claiming to be.

**RBAC vs CBAC**

Ok, now, if you ask what is the benefit of Role-based access control or Claim based access control, then, think about this page "ViewImagesOfViolence". Is not it more intuitive to check for a claim "Adult-over-18years" when determining if you should allow the user to visit that page? In a word, using Claims, you can create more segments within your users comparing roles. In an abstract sense, all roles can be claims too, but claims cannot be thought of as roles.

**Permission-based access control**

Instead of checking for Roles or Claims when allowing permissions to view a page, you should rather think about Permission-based access control. Let me show you some pain points.

When you are using Role-based authentication, if you have an action for creating customers and you want that the people who are in the 'Sale' role should be able to do that, then you write code like this:

[Authorize(Roles="Sale")]

public ActionResult CreateCustomer()

{

return View();

}

Later, you realized that, sometimes, people from the 'Marketing' role should be able to create Customers. Then, you update your Action method like that

[Authorize(Roles = "Sale", "Marketing")]

public ActionResult CreateCustomer()

{

return View();

}

Now, you realized that some of the marketing people must not be able to create Customers, but it is not possible to assign a different role for those people who are in Marketing. So, you are forced to allow all marketing people to create Customers.

you spotted another problem, anytime you decide that Marketing people should be allowed to create customers, you have to update all of your MVC Action methods Authorize attribute, compile your application, test, and deploy. Some days later, you decided, not marketing but some other role should be allowed to do the task, so you search in your codebase and delete all 'Marketing' from Authorize attribute and add your new role name in Authorize attribute... Not a healthy solution. At that point, you would realize a need for Permission-Based Access Control.

Permission-Based access control is a way of assigning various permissions to various users or various roles or various claims and checking if a user has permission to execute an action from the code in run time. If you assign permission to a role or a claim, then, you would check what are the roles or claims for that logged-in user. And then, you will check what permissions are available for those roles or claims.

You can define some set of permissions like this :

"CanCreateCustomer", "CanDeleteCustomer", "CanEditCustomer".. etc..

Now, you can decorate your Action Method like this:

[Authorize(Permission="CanCreateCustomer")]

public ActionResult CreateCustomer()

{

return View();

}

Please note, [Authorize(Permission="CanCreateCustomer")] may not be built into the MVC class library, I am just showing as an example in an abstract sense. There can be a NuGet package that will have Permission property in Authorize class.

Now, you can see that, CreateCustomer action method will always need permission 'CanCreateCustomer' and it will never change or hardly change.

**Who will get the permissions?**

You can assign a set of permissions directly to a user. But do not do that. It will be tremendously difficult to manage that. Rather,

You can assign a set of permissions to a Role Or you can assign a set of permissions to a Claim (Recommended).

As I mentioned, roles can be thought of as claims too. So, you can treat the roles as claims. Then, you can create a table of Claims in your database. Then, create another table for holding the relations where each claim can contain multiple permissions.

This security model offers you clean code practice. Moreover, when you write your Action Method, you don't have to think about who can use this method, rather you can always be assured that whoever is using this method will have proper permission given by the Admin. Then, Admin can decide who will be able to do what. Not you as a developer. That's how your business logic is separated from Security logic.

Whenever someone signs in, your application will check whatever permissions are available for that user and that permission set will be available as additional properties of the currently logged-in user, so you don't have to check permission set all the time from the database. The bottom line is, you get more control of your security logic in your application if you apply permission-based access control.

If your application is a very little application where there would be only 2 roles: Customer and Admin and there is no chance that Customers will be able to do anything else other than what they are meant to do in your application, then perhaps, simple Role-based access control will serve the purpose, but as your application grows, you will start to feel the need of permission based access control at some point.