Securing microservices

(Demo prep)

- Remove all tokens from Postman
- Clear Authorization values in Postman

Authorization with Scopes

Let’s now explore how to use claims-based authorization to enable more strict rules on the access to the REST API operations of our Catalog microservice.

In Identity repo

1. Update **appsettings.json**:

```
  "IdentityServerSettings": {
    "ApiScopes": [
      {
        "Name": "catalog.fullaccess"
      },
      {
        "Name": "catalog.readaccess"
      },
      {
        "Name": "catalog.writeaccess"
      },
      {
        "Name": "inventory.fullaccess"
      },
      {
        "Name": "IdentityServerApi"
      }
    ],
    "ApiResources": [
      {
        "Name": "Catalog",
        "Scopes": [
          "catalog.fullaccess",
          "catalog.readaccess",
          "catalog.writeaccess"
        ],
        "UserClaims": [
          "role"
        ]
      },
      {
        "Name": "Inventory",
        "DisplayName": "Inventory API",
        "Scopes": [
          "inventory.fullaccess"
        ],
        "UserClaims": [
          "role"
        ]
      }
    ]
```

2. Update **appsettings.Development.json**:

```
"IdentityServerSettings": {
  "Clients": [
    {
      "ClientId": "postman",
      "AllowedGrantTypes": [
        "authorization_code"
      ],
      "RequireClientSecret": false,
      "RedirectUris": [
        "urn:ietf:wg:oauth:2.0:oob"
      ],
      "AllowedScopes": [
        "openid",
        "profile",
        "catalog.fullaccess",
        "catalog.readaccess",
        "catalog.writeaccess",
        "inventory.fullaccess",
        "IdentityServerApi"
      ],
      "AlwaysIncludeUserClaimsInIdToken": true
    }
  ]
}
```

In Catalog repo

3. Add **Policies.cs**:

```
namespace Play.Catalog.Service
{
    public static class Policies
    {
        public const string Read = "read_access";
        public const string Write = "write_access";
    }
}
```

4. Update **Startup.cs**:

```
public void ConfigureServices(IServiceCollection services)
{
   …

    services.AddMongo()
            .AddMongoRepository<Item>("items")
            .AddMassTransitWithRabbitMq()
            .AddJwtBearerAuthentication();

    services.AddAuthorization(options =>
    {
        options.AddPolicy(Policies.Read, policy =>
        {
            policy.RequireRole("Admin");
            policy.RequireClaim("scope", "catalog.readaccess", "catalog.fullaccess");
        });

        options.AddPolicy(Policies.Write, policy =>
        {
            policy.RequireRole("Admin");
            policy.RequireClaim("scope", "catalog.writeaccess", "catalog.fullaccess");
        });
    });
    …
}
```

5. Update **ItemsController**.cs:

```
[ApiController]
[Route("items")]
public class ItemsController : ControllerBase
{
    …
    [HttpGet]
    [Authorize(Policies.Read)]
    public async Task<ActionResult<IEnumerable<ItemDto>>> GetAsync()
    {
     …
    }

    // GET /items/{id}
    [HttpGet("{id}")]
    [Authorize(Policies.Read)]
    public async Task<ActionResult<ItemDto>> GetByIdAsync(Guid id)
    {
     …
    }

    // POST /items
    [HttpPost]
    [Authorize(Policies.Write)]
    public async Task<ActionResult<ItemDto>> PostAsync(CreateItemDto createItemDto)
    {
     …
    }

    // PUT /items/{id}
    [HttpPut("{id}")]
    [Authorize(Policies.Write)]
    public async Task<IActionResult> PutAsync(Guid id, UpdateItemDto updateItemDto)
    {
     …
    }

    // DELETE /items/{id}
    [HttpDelete("{id}")]
    [Authorize(Policies.Write)]
    public async Task<IActionResult> DeleteAsync(Guid id)
    {
     …
    }
}
```

6. Start Identity and Catalog servers

7. Let's verify our READ policy first

8. Request a token for Player1 with catalog.readaccess scope (NAME THE TOKEN)

9. Try a GET /items (403 Forbidden)

10. Show the console logs

11. Request a token for Admin with catalog.readaccess scope (NAME THE TOKEN)

12. Try a GET /items (200 OK)

13. Try POST new item (403 Forbidden) (same catalog.readccess scope)

14. Request a token for Admin with catalog.writeaccess scope (NAME THE TOKEN)

15. Try POST new item (200 OK)

So that’s how you can use more granular claims-based rules to authorize access to your REST API operations.

That’s the end of this module and, at this point, you have a fully protected set of microservices that can only be accessed by properly following the modern set of security patterns and techniques that you have learned in this course, all powered by ASP.NET Core Identity and IdentityServer.