Authorization

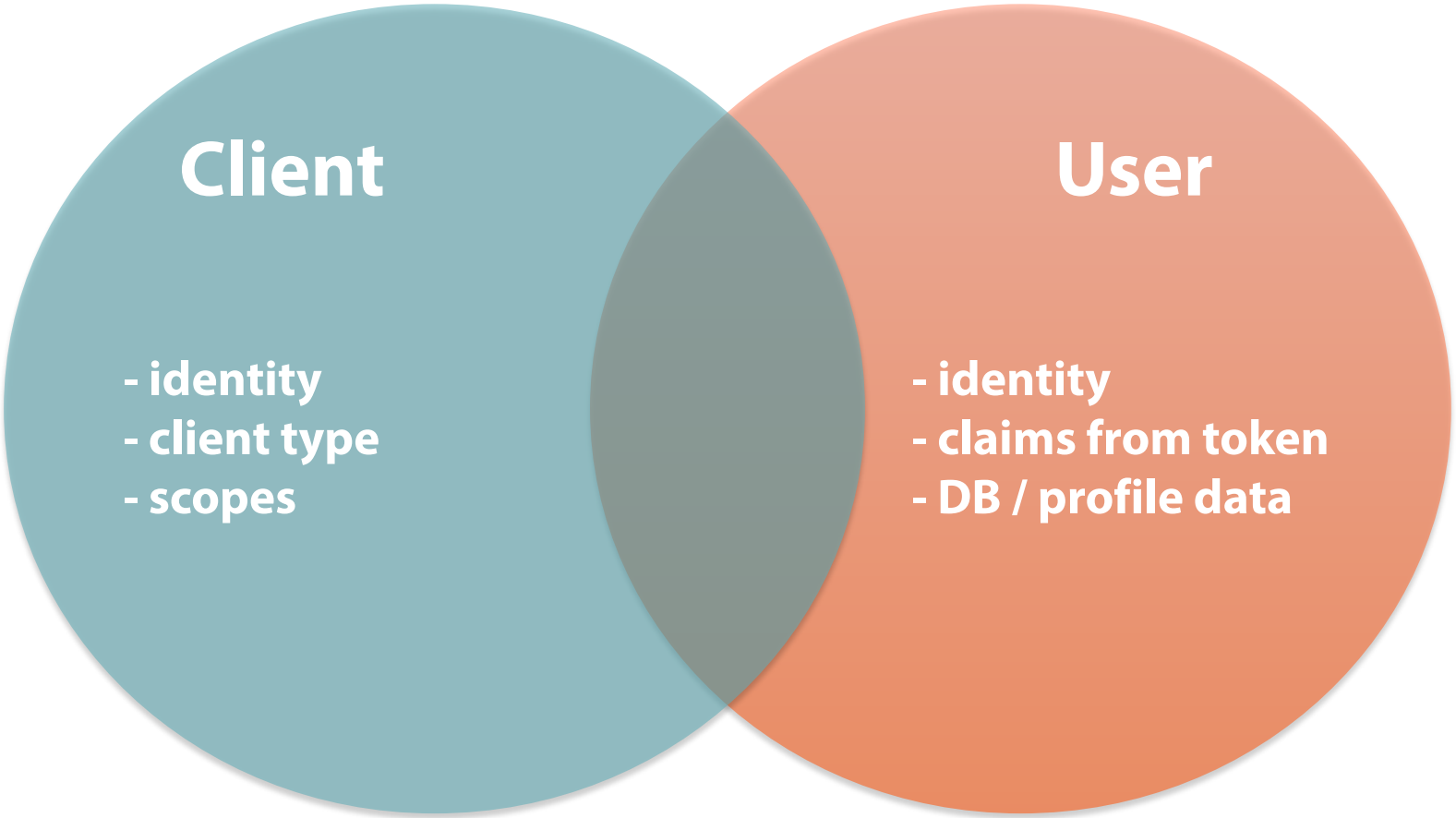
Dominick Baier
http://leastprivilege.com
@leastprivilege

pluralsight
hardcore dev and IT training

Agenda

- Client vs user Authorization
- Authorization filters and attributes
- Custom authorization logic

Authorization



Options

- **Every piece of the pipeline could abort the current request and return a 401**
  - „as early as possible, as late as needed“
- **Typical places (from coarse to fine grained)**
  - OWIN middleware
  - global authorization filter
  - (declarative) controller authorization filter
  - (declarative) action authorization filter
  - (imperative) code „inside“ action/business logic

AuthorizeAttribute

```
[Authorize(Roles = "SomeRole")]
public class StandardAttributesController : ApiController
{
    public IHttpActionResult Get() { … }

    [AllowAnonymous]
    public IHttpActionResult Post() { … }

    [Authorize(Roles = "SomeAdditionalRole")]
    public IHttpActionResult Put() { … }

    [OverrideAuthorization]
    [Authorize(Roles = "SomeOtherRole, OrSomeOtherRole")]
    public IHttpActionResult Delete() { … }
}
```

AuthorizeAttribute Internals

```
[AttributeUsage(AttributeTargets.Method | AttributeTargets.Class,
  Inherited = true, AllowMultiple = true)]
public class AuthorizeAttribute : AuthorizationFilterAttribute
{
  public override void OnAuthorization(HttpActionContext actionContext)
  {
    if (SkipAuthorization(actionContext))
    {
        return;
    }

    if (!IsAuthorized(actionContext))
    {
        HandleUnauthorizedRequest(actionContext);
    }
  }
}
```

Custom Authorization Filter / Attribute

```
public class CustomAuthorizationAttribute : AuthorizeAttribute
{
    protected override bool IsAuthorized(HttpActionContext actionContext)
    {
        // retrieve principal and check authZ
        var principal = actionContext.RequestContext.Principal
          as ClaimsPrincipal;

        return outcome;
    }

    protected override void HandleUnauthorizedRequest(
      HttpActionContext actionContext)
    {
        actionContext.Response =
          actionContext.Request.CreateErrorResponse(
            HttpStatusCode.Unauthorized, "unauthorized");
    }
}
```

Custom Authorization Attributes

- User

```
[ResourceActionAuthorize("add", "customer")]
public HttpResponseMessage Post(Customer c)
{ }
```

- Client

```
[ScopeAuthorize("add")]
public HttpResponseMessage Post(Customer c)
{ }
```

https://github.com/thinktecture/Thinktecture.IdentityModel

Imperative Authorization Logic

```
public IHttpActionResult Post(Customer customer)
{
    if (!Request.CheckAccess(
        "add", "customer", customer.Region))
    {
        return Unauthorized(new AuthenticationHeaderValue(
            "Bearer", "Permission required."));
    }

    …
}
```

Summary

- **Authorization is very application specific**
  - clients, users, roles, permissions, multi-tenancy, row level…
- **Make an authorization decision at the earliest possible point in time where you have all necessary data**
  - coarse vs fine grained authorization
- **Web API has a dedicated authorization stage**
  - filter and attribute
- **Avoid mixing authorization logic with business or facade logic**
  - resource/action based approach does a clean separation