Securing Services

Miguel A. Castro
@miguelcastro67

pluralsight
hardcore developer training

WCF Security

- **Huge Topic !**
  - A full day to cover everything

- **Security Modes**
- **Protection Levels**
- **Authentication**
- **Authorization**
- **Identities**
- **Certificates**

- **Believe it or not, I'm gonna touch on all of this**

WCF Security Scenarios

- **Two primary scenarios cover most cases**
- **Intranet Scenario**
  - Clients are inside the firewall
  - TCP Binding
  - Windows-based authentication/authorization
- **Internet Scenario**
  - Clients outside the firewall
  - HTTP Binding
  - Certificate-protected custom authentication/authorization

Identities

- **Client identity passed to service**
- **Various identities available on service**
  - Token/Host Identity
    - Identity of the host process – identity used to access resources *(file system, registry, database, etc.)*
      WindowsIdentity.GetCurrent()
  - Primary Identity
    - Identity passed in from the client – placed on the thread *after* the service constructor is hit
      ServiceSecurityContext.Current.PrimaryIdentity
  - Windows Identity
    - Same as Primary Identity if Windows Authentication used – empty if not
      ServiceSecurityContext.Current.WindowsIdentity
- **All are implementations of the IIdentity interface**

Intranet Security

- **Typically `netTcpBinding` used**
- **Security set to `Transport`**
- **Protection Level**
  - How the message is protected
  - Three choices: None, Signed, Encrypted & Signed
- **Client Credential Type**
  - How the client will authenticate to the service
  - Three choices: None, Certificate, **Windows**
- **All these set on the binding *(both sides)***
- **Binding is inherently secure**
  - No clear text
  - No certificate necessary – yeah!

Authentication

- **Service will authenticate using Windows credentials**
- **By default, credentials sent by proxy are those of the client**
  - Remember, the client on a web app is IIS
- **Alternative windows credentials can be sent in through the proxy**
- **`PrimaryIdentity` property receives that login user**

Authorization

- **Authorization is against the `PrimaryIdentity`**
- **A Windows principal is created with the identity and its associated roles (windows groups)**
  - Principal placed on thread
  - Regular **Thread.CurrentPrincipal**
  - Principal is implementation of the **IPrincipal** interface
- **`PrincipalPermission` attribute can be used on operations**
  - Can demand authorization based on user name or role
- **This demand uses Thread.CurrentPrincipal.IsInRole**
- **Attribute is NOT WCF specific**
  - Been around for a while
  - Can be used for any down-level classes

Demo Time

Intranet Web Application

- Client credentials sent are those of IIS server
- Web app can perform a programmatic [soft] impersonation just before the service call
- Credentials sent are those of the browser client
- Contexts User Identity needs to be cast to `WindowsIdentity` to access `Impersonate`
- Impersonation should be “disposed” after call

Demo Time

Internet Security (Service)

- **Typically `wsHttpBinding` used**
- **Security set to `Message`**
- **Client Credential Type**
  - How the client will authenticate to the service
  - Choices: None, Certificate, Windows, **UserName**
- **Must tell how to negotiate service credentials**
- **All these set on the binding *(both sides)***
- **Binding is inherently unsecure**
  - Clear text
  - Must use certificate
- **Service can use Windows or ASP.NET Providers**

Internet Security (Client)

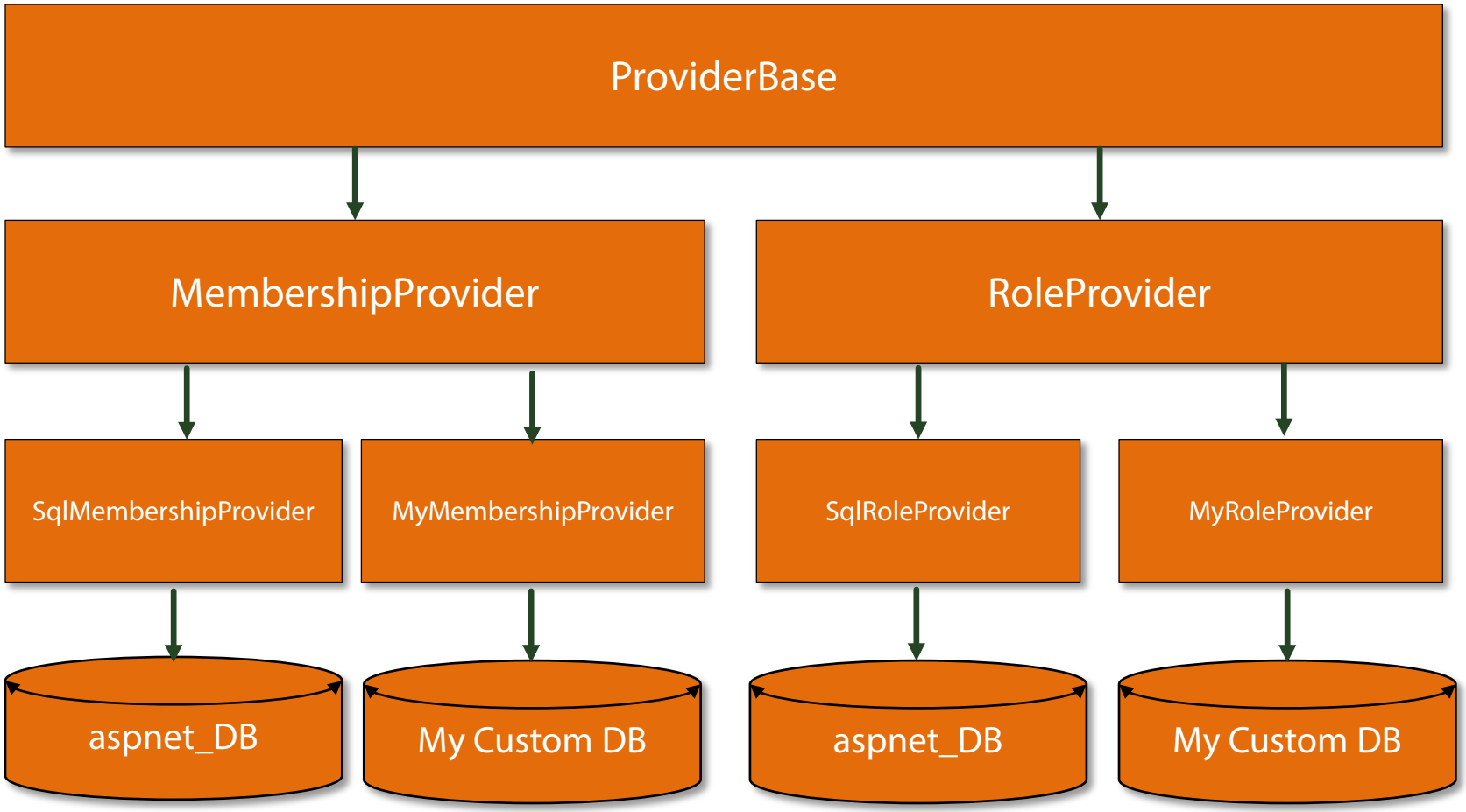
- **Client needs endpoint behavior to define certificate trust level**
- **If `PeerTrust` used:**
  - Client must have copy of certificate (w/o private key) ahead of time
  - Endpoint needs to define certificate name (defaults to cert name same as host name)
- **If `ChainTrust` used:**
  - Client must have the encoded public key (base 64)
  - Available as meta-data of service if ASR is used

Demo Time

ASP.NET Providers

- **Been around since ASP.NET 2.0**
- **Use a strategy pattern to determine authentication and authorization implementations**
- **Microsoft provides built-in providers & DB**
- **You can develop your own**
- **Note that as an ASP.NET Security solution, this model is deprecated (though still works and is useful) and has since been replaced by the new ASP.NET Identity**
  - As of the production of this course (Jan 2015), it is still used by the latest version of WCF

ASP.NET Providers



ASP.NET Providers in WCF

- **Membership provider needs to implement**
  - ValidateUser
- **Role provider needs to implement**
  - UserIsInRole
- **No automatic caching mechanism for roles**
  - Unlike ASP.NET WebForms and MVC
- **Providers defined in `<system.web>`**
- **Service config needs to point to providers**

Demo Time