Security

Securing your WCF services

Outline

- **WCF provides three important security features**
  - Confidentiality
  - Integrity
  - Authentication
- **Security is on *by default* in almost all bindings**
  - You configure transport vs. message using the security mode
  - You configure authentication via the client credential type
- **WCF provides numerous authorization options**
  - Impersonation
  - Role-based access control
  - Service authorization behavior

The "CIA" of security

- **Is security important?**
  - Do you have resources that have value to an adversary?
  - If so, then you must expect to be attacked
- **WCF provides basic protections that you need: CIA**

Confidentiality	Integrity	Authentication
• Encrypting messages mitigates eavesdropping attacks	• Signing messages mitigates tampering and replay attacks	• Proof of identity mitigates spoofing and impersonation attacks

Choices you'll need to make

- **The protection level required by your services**
  - Should the data be signed, encrypted, or both?
- **Transport vs. message security on bindings**
  - Can also use a hybrid of the two
- **Authentication, or "Who are you?"**
  - You choose the type of credentials you want the client to use, and WCF will pick an appropriate authentication protocol
- **Authorization, or "What are you allowed to do?"**
  - Impersonate the caller, letting someone else handle authz
  - Provide your own authorization management

Declaring the required protection level

- **The developer of a service doesn't ultimately control how it's exposed**
  - So what if the host application exposes unsecure endpoints?
- **Hence, developers can set the required protection level on contracts**
  - The host will fail if the required protection level isn't met by an endpoint
- **You can set the protection level at different scopes**
  - On a particular message
  - On individual operations and fault contracts
  - On a service contract

ProtectionLevel

- **Simply use the ProtectionLevel property on the appropriate attribute**
  - It comes with three values: None, Sign, and EncryptAndSign

```
[MessageContract(IsWrapped=false,
    ProtectionLevel=ProtectionLevel.EncryptAndSign)]
public class MathRequest
{
   ...
```

Requires the message to be signed & encrypted

Requires all operations to be signed

Requires this operation to be signed & encrypted

```
[ServiceContract(Name="SimpleMath",
    ProtectionLevel=ProtectionLevel.Sign)]
public interface IMath
{
    [OperationContract(Name="add",
        ProtectionLevel=ProtectionLevel.EncryptAndSign]
    MathResponse Add(MathRequest request);
...
```

Configuring security in WCF bindings

Security mode	Client credential type
• Transport • Message • Mixed	• Username • Certificate • Windows • IssuedToken

These two choices determine how security protocols will be implemented

Configuring binding security settings

Windows Integrated Authn

```
<wsHttpBinding>
  <binding name="MyTransportSecurityBinding">
    <security mode="Transport">
      <transport clientCredentialType="Windows"/>
    </security>
  </binding>
</wsHttpBinding>
```

Service supplies X.509 cert
Client supplies User name + pwd

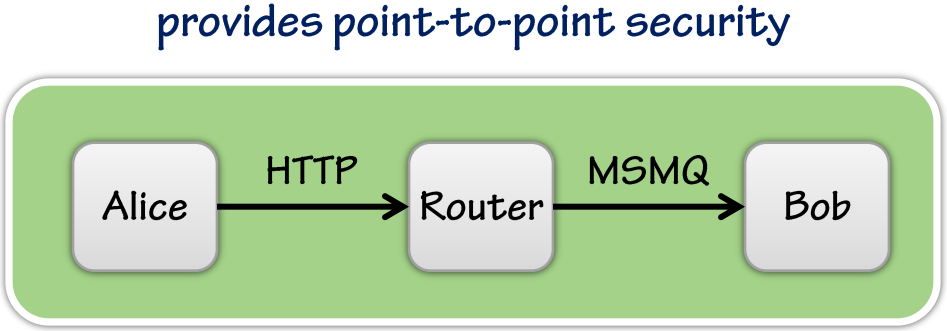
```
<wsHttpBinding>
  <binding name="MyMessageSecurityBinding">
    <security mode="Message">
      <message clientCredentialType="UserName"/>
    </security>
  </binding>
</wsHttpBinding>
```

Service runs SSL
Client supplies SAML token

```
<wsHttpBinding>
  <binding name="MyMixedSecurityBinding">
    <security mode="TransportWithMessageCredential">
       <message clientCredentialType="IssuedToken"/>
    </security>
  </binding>
</wsHttpBinding>
```

Transport security

- **Each transport typically has a built-in security layer that you can use**
  - HTTP using SSL
  - TCP/NP using Kerberos
  - MSMQ using certificates
- **Provides point-to-point security between nodes**

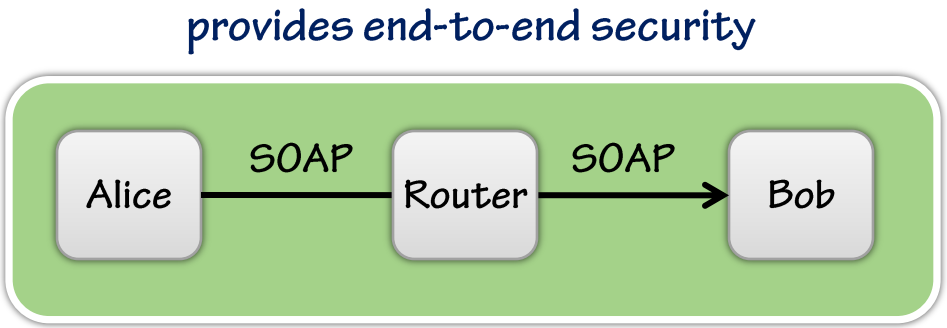


Transport security tradeoffs

Benefits	• Mature, well understood security model • Better performance
Drawbacks	• Constrains the type of client credentials • You get point-to-point authentication, not end-to-end authentication

Message security

- **Message security pushes authentication down into the SOAP headers**
  - Provides same security features as transport security
  - But in a transport-neutral way (pushes security into SOAP messages)
- **Provides an end-to-end security solution across all nodes**



Message security tradeoffs

Benefits

- Supports a wide variety of credentials
- Largely independent of transport
- Supports end-to-end authentication
- Multiple WCF extensibility hooks

Drawbacks

- Newer isn't always better for security
- WS-* isn't as broadly adopted as SSL
- Perf can be significantly worse

Mixed mode

- **TransportWithMessageCredential**
  - Speed and maturity of transport security
  - Flexibility of client credential types embedded in message
- **Transport security typically supplied by SSL**
  - Authenticates service to client via service's certificate
  - Sign & encrypt payload
- **WS-Security header holds client credential**
  - Opens up many options for credential format

Credential formats

- **There's one setting to control credentials: clientCredentialType**
  - Selects the type of credential the client must present
  - Also implicitly dictates the type of credential the server must possess
  - A certificate for the service endpoint is almost always needed

clientCredentialType	Implied service credential type
None	Certificate (optional)
Username	Certificate
Certificate	Certificate
Windows	Windows
IssuedToken	Certificate

Authentication in standard bindings

- **Authn required by default in almost all standard bindings**
  - BasicHttpBinding is an exception

Binding Name	Transport	Message	Default Client Credential
BasicHttpBinding	Supported	Supported	None
WSHttpBinding	Supported	Default	Windows
WSDualHttpBinding	Supported	Default	Windows
NetTcpBinding	Default	Supported	Windows
NetNamedPipesBinding	Default	Supported	Windows
NetMsmqBinding	Default	Supported	Windows

Security call context

- **Every secure WCF operation has a ServiceSecurityContext object**
  - ServiceSecurityContext.Current
  - OperationContext.ServiceSecurityContext
- **The context object provides you with information about the caller**
  - Use PrimaryIdentity or WindowsIdentity to access the IIdentity object
  - IsAnonymous will tell you if it was an anonymous call

Discovering client identity in a service

- The following example discovers the client identity in an operation

```
public void ApproveInvoice(int invoiceId, string comments) {
    OperationContext ctx = OperationContext.Current;
    ServiceSecurityContext sctx = ctx.ServiceSecurityContext;

    // security context can be null if client is anonymous
    if (null == sctx) throwAccessDeniedFault();

    // IIdentity is same interface used elsewhere in .NET
    IIdentity id = sctx.PrimaryIdentity;
    auditOperation("ApproveInvoice", id.Name);

    // ...
}
```

Authorization options

Role-based access control	• Windows groups a simple option (use IPrincipal) • Use an ASP.NET role provider • PrincipalPermission works reasonably well
ServiceAuthorizationBehavior	• Decision based on SOAP action & client identity • Fires earlier than PrincipalPermission • Hoists authz logic out of service implementation
Impersonation	• Only an option w/Windows creds • Use WindowsIdentity.Impersonate or [OperationBehavior]

Impersonation

- **Impersonation is a Windows feature**
  - Must be using Windows authn for this to work
  - Easy to get this working for local resources
  - Trickier for remote resources (requires delegation)
- **Temporarily take on the client's identity**
  - You're passing the authorization problem to a system behind you
  - Great when you're accessing existing secure resources
  - Can eliminate the need for you to implement authz in your app

Impersonation

- **File system will use client's security context to grant permission**
  - File.OpenText throws an exception if client doesn't have read permission

```
public string ReadFile(string fname) {
    WindowsIdentity id = Thread.CurrentPrincipal.Identity
                         as WindowsIdentity;
    if (null == id) throwSoapFault("Windows authn required");
    using (WindowsImpersonationContext wic = id.Impersonate()) {
        try {
            return File.OpenText(fname).ReadToEnd();
        }
        catch (UnauthorizedAccessException) {
            throwSoapFault("Unauthorized");
        }
    }
}
```

Impersonation is a WCF behavior

- Can also use a behavior to achieve the same goal

```
[OperationBehavior(Impersonation=ImpersonationOption.Required)]
public string ReadFile(string fname) {
    try {
        return File.OpenText(fname).ReadToEnd();
    }
    catch (UnauthorizedAccessException) {
        throwSoapFault("Unauthorized");
    }
}
```

Groups, Roles, and Claims

- **Windows groups can be used directly for authorization**
  - Thread.CurrentPrincipal.IsInRole("MyDomain\MyGroup")
  - [PrincipalPermission(…, Role="MyDomain\MyGroup")]
  - Drawback: hard to deploy if domain/machine names are hardcoded
- **Can use AzMan or ASP.NET role provider**
  - Thread.CurrentPrincipal.IsInRole("MyRole")
  - [PrincipalPermission(…, Role="MyRole")]
  - Benefit: each application has its own roles, no name collisions
- **To prepare for federation scenarios, start thinking about claims**
  - Claims are a superset of groups and roles

Federation and claims

- **Federation allows you to rely on identity information from trusted partners or identity providers**
  - No need to provision accounts for users (reduces costs)
- **Security tokens for users consist of a signed set of claims**
  - Each claim is a statement about the user
  - Signature tells you who is making the statements
- **Structure of a claim is very simple**
  - URI indicates type of claim
  - Name of claim
  - Value of claim

WCF claims-based authorization

- **Claims are the most general-purpose authorization mechanism**
  - Supports Windows groups
  - Supports roles
  - Supports federated identity scenarios and CardSpace
- **Each authenticated client presents a set of claims**
  - Service can enumerate claims and make authorization decisions
  - Available via ServiceSecurityContext.AuthorizationContext

Authorization via behaviors

- **Best to do authz as early as possible in the pipeline**
  - Why decrypt the message body if you're just going to reject it?
  - Why unmarshal the message, etc?
- **ServiceAuthorization behavior allows you to do this**
  - You must derive a class from ServiceAuthorizationManager
  - Given SOAP action and security context, return a boolean indicating whether the user may perform the action
- **Compare this to using [PrincipalPermission] or IsInRole**
  - These techniques can only happen after message is unmarshaled
  - Using a behavior also helps you to isolate authorization logic from business logic, which is a good idea

Authorization via behaviors

```
public class MyAuthzManager : ServiceAuthorizationManager
{
    protected override bool CheckAccessCore(OperationContext operationContext)
    {
        string action =
            operationContext.RequestContext.RequestMessage.Headers.Action;
        AuthorizationContext authzCtx =
            operationContext.ServiceSecurityContext.AuthorizationContext;

        // could also look at caller's IPrincipal if you prefer
        return userIsAuthorized(action, authzCtx.ClaimSets);
    }
}
```

```
<behavior name="MyAuthzBehavior">
  <serviceAuthorization serviceAuthorizationManagerType="MyAuthzManager" />
</behavior>
```

Summary

- **WCF provides three important security features**
  - Confidentiality
  - Integrity
  - Authentication
- **Security is on *by default* in almost all bindings**
  - You configure transport vs. message using the security mode
  - You configure authentication via the client credential type
- **WCF provides numerous authorization options**
  - Impersonation
  - Role-based access control via groups, roles, or claims
  - Service authorization behavior

References

- **Security in Windows Communication Foundation**
  - http://msdn.microsoft.com/msdnmag/issues/06/08/SecurityBriefs/
- **Input validation tutorials produced by Pluralsight**
  - http://www.pluralsight.com/inputValidationTutorials
- **Pluralsight's WCF Wiki**
  - http://pluralsight.com/wiki/default.aspx/Aaron/WindowsCommunicationFoundationWiki.html