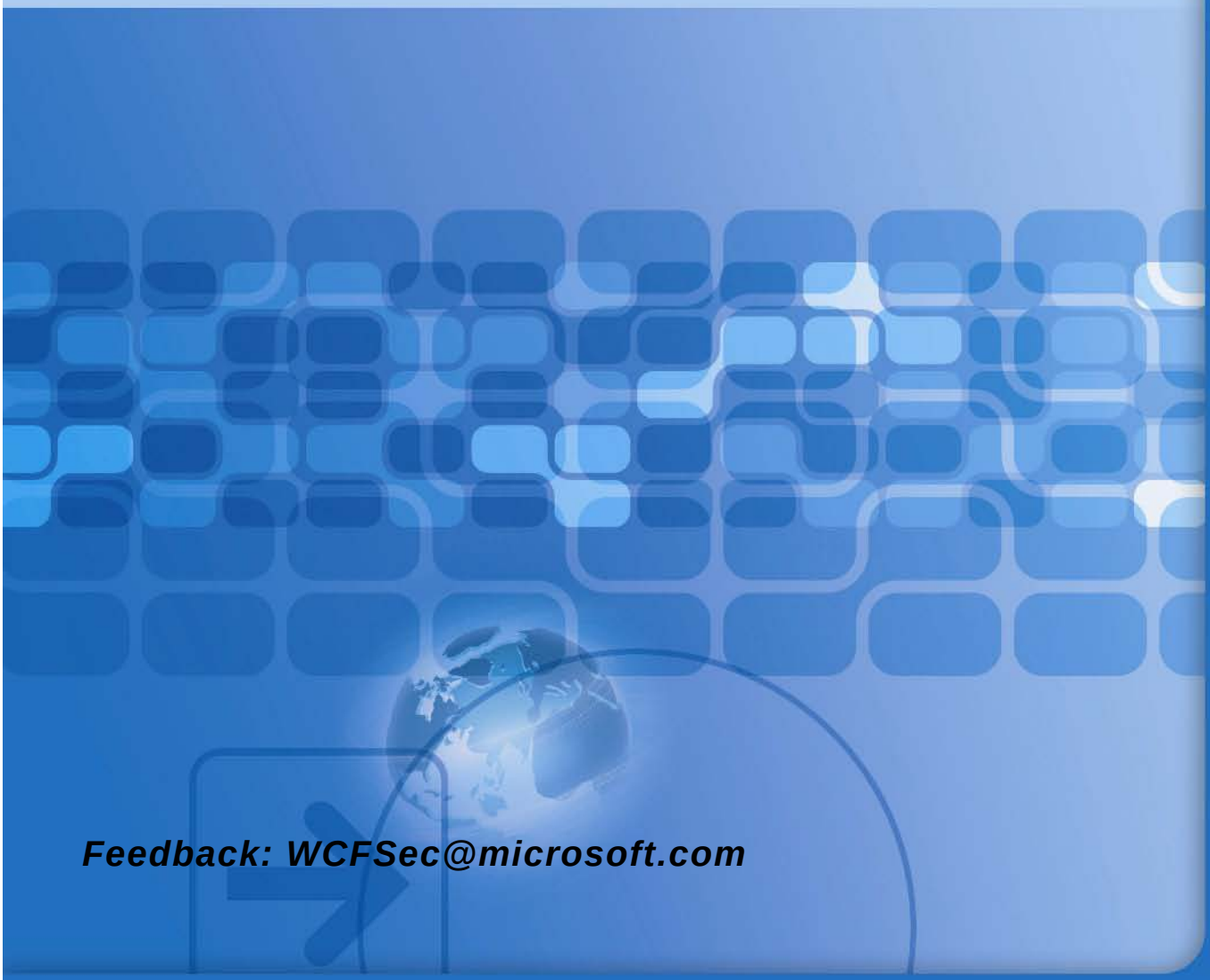
Microsoft

Improving Web Services Security

Scenarios and Implementation Guidance for WCF



Feedback: WCFSec@microsoft.com

patterns & practices

Improving Web Services Security

Scenarios and Implementation Guidance for WCF

patterns & practices

J.D. Meier
Carlos Farre
Jason Taylor
Prashant Bansode
Steve Gregersen
Madhu Sundararajan
Rob Boucher

Foreword by Nicholas Allen

The computer industry has come to a realization – based on many years of slowly learning from painful experiences – that computer networks are hostile environments. Nevertheless, computer users demand as part of their basic expectations that applications take advantage of the ubiquitous and continuously available connectivity at their disposal to deliver a rich connected experience.

It is now your task to design and assemble the loosely coupled service components that you have available in a way that blunts threats and thwarts attacks on the user's precious assets. Your applications must withstand the hazards of living in a hostile networked environment. To make that possible, you must understand the risks that your applications face and you must be certain that the remedies you put in place properly mitigate the dangers of those risks.

As someone who has been through several rounds of security and threat modeling for Windows Communication Foundation, I can say without hesitation that knowledge and experience are your greatest assets for designing secure Web service applications. The trick is to gain as much of that knowledge as possible from the painful experiences of other people rather than painful experiences of your own.

J.D. Meier and team have done a fantastic job of assembling and digesting countless practical experiences into a convenient and centralized resource. Practitioners of service-oriented development with WCF will want to use this guide as both a means of learning about the fundamentals of Web service security and a reference for getting specific, step-by-step instructions for dozens of the most common security problems. I enjoy that this guide collects together several different approaches for learning about and implementing security solutions. By combining a variety of formats – scenarios, how-to articles, and guidelines are only a sample of the offered modes – solutions are both reinforced and made more easily discoverable through different entry points.

The reason that I'm so excited to see *Improving Web Services Security: Scenarios and Implementation Guidance for WCF* is that having a secure system has become such a deep and pervasive requirement that security has to be treated as part and parcel of functionality. Having the Guide to make WCF security understandable and accessible adds value to the WCF platform by improving its usability as a whole. I highly recommend this book to anyone involved in the development, deployment, or management of WCF applications. This book has something of value for you whether it is read end to end or consumed tactically in parts to solve a specific problem. Security is too intrinsically important to pass up this aid to your success.

Nicholas Allen
Program Manager, Windows Communication Foundation
May 2008

Foreword by Rockford Lhotka

Looking into the future, it is clear that Windows Communication Foundation (WCF) is one of the core pillars of the Microsoft .NET Framework. As the logical successor to ASMX Web services, Web service extensions, Remoting, Microsoft Message Queuing (MSMQ), and Enterprise Services, WCF is the single API for any cross-process or cross-network communication needs in .NET. This is true for both service-oriented and n-tier client/server scenarios, as WCF effectively supports both models.

While Visual Studio continues to improve its tool support for WCF, the reality is that WCF is a very large and complex technology. Tooling alone can't simplify all the options enough to make the use of WCF truly easy. It is critical that developers using WCF understand the various security configuration options, how they interact with the available bindings, and the ramifications of their choices.

Although understanding the options and their consequences is critical, one must ultimately implement the decisions. Typically, this is done through configuration of WCF, which is perhaps the hardest and most complex part of any WCF project. Even with the configuration tools available, configuring WCF for even relatively simple security models can be a very painstaking and time-consuming task.
This is why the guidance you are about to read is so exciting! It opens with a section covering the security concerns you'll need to address when building service-oriented systems. The discussion then moves on to coverage of the concepts and reasoning behind the available security options in WCF, and how choices made here can impact your options elsewhere in WCF. Armed with that background, you can then read the sections covering specific scenarios for both Internet and intranet application models. Finally come what I view as the jewels of this guidance: the detailed how-to walkthroughs for configuring WCF as needed to meet your security requirements.

Nowhere else will you find such unified content describing the concepts, reasoning, and practical application of security in Windows Communication Foundation.

Rockford Lhotka
Principal Technology Evangelist, Magenic
May 2008

Introduction

This guide shows you how to improve security for your WCF services. It also shows you how to effectively design your authentication, authorization, and communication strategies for Microsoft® Windows Communication Foundation.

The information in this guide is based on practices learned from customer feedback and product support, as well as experience gained in the field and while implementing real solutions. The guidance is task-based and presented in the following parts:

- **Part I – Security Fundamentals for Web Services** gives you a quick overview of fundamental security concepts as they relate to services, service-oriented design, and Service-Oriented Architecture (SOA).
- **Part II – WCF Security Fundamentals** gives you a firm foundation in key WCF security concepts, with special attention on authentication, authorization, and secure communication, as well as WCF binding configurations.
- **Part III – Intranet Application Scenarios** shows you a set of end-to-end intranet application scenarios that you can use to jump-start your application architecture designs, with a focus on authentication, authorization, and communication for your intranet from a WCF perspective.
- **Part IV – Internet Application Scenarios** shows you a set of end-to-end Internet application scenarios that you can use to jump-start your application architecture design for the Internet from a WCF perspective.

WCF / Services Security

Many factors and decisions combine to improve security in WCF services and applications. This guide focuses on the following:

- Authentication, authorization, and communication design for your services
- Solution patterns for common distributed application scenarios using WCF
- Principles, patterns, and practices for improving key security aspects in services

The following diagram illustrates a common solution pattern for WCF intranet scenarios:

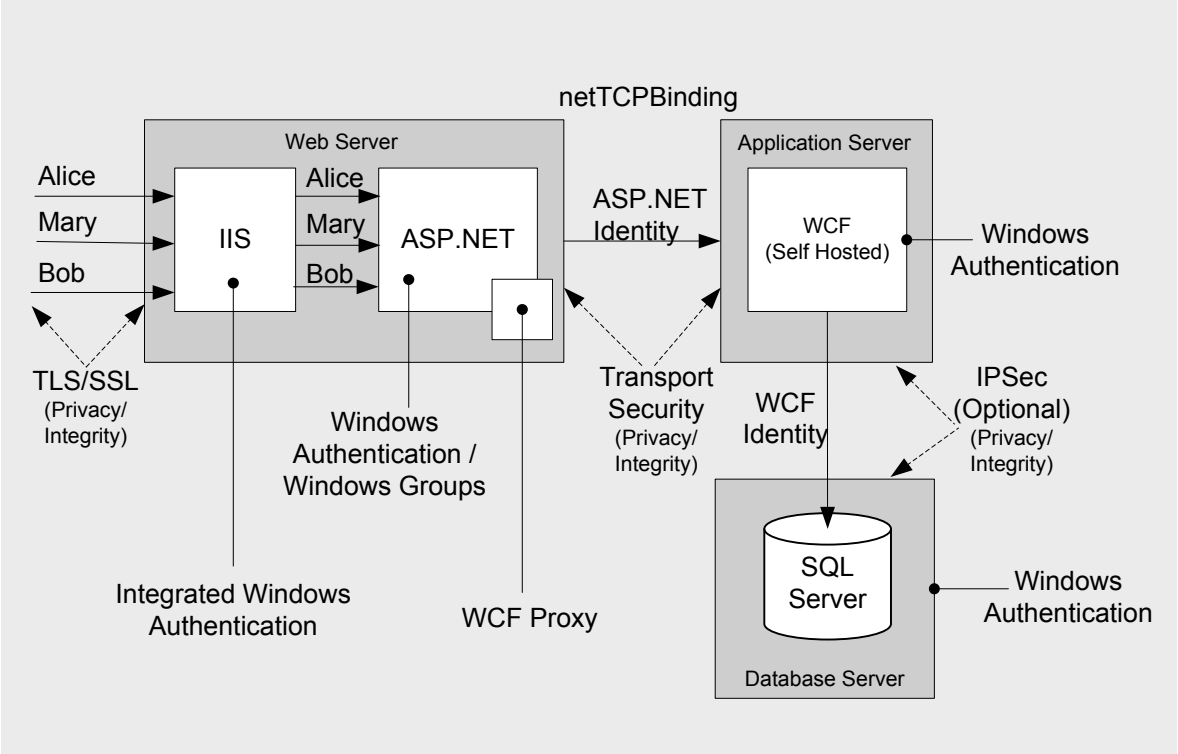


Figure 1. Example WCF Implementation Solution Pattern

Scope of This Guide

This guide is focused on key security aspects of WCF. The guide addresses security across the three primary physical tiers: the client, remote application server, and database server. Clients include Microsoft Windows Forms, ASP.NET, and WCF.

Out of Scope

The following are outside the scope for this guide:

- Federation
- Claims authorization

Why We Wrote This Guide

From our own experience with WCF, and through conversations with customers and Microsoft employees who work in the field, we determined that there was significant demand for a guide that would show how to use WCF in the real world. While there is information in the product documentation, in blog posts, and in forums, there has been no single place to find proven practices for the effective use of WCF in the context of line-of-business (LOB) applications under real-world constraints.

Who Should Read This Guide

This guide is targeted at individuals involved in building applications with WCF. The following are examples of roles that would benefit from this guidance:

- A development team that wants to adopt WCF
- A software architect or developer looking to get the most out of WCF, with regard to designing his or her application security
- Interested parties investigating the use of WCF who don't know how well it would work for their particular deployment scenarios and constraints
- Individuals tasked with learning WCF security practices

How to Use This Guide

Use the first part of the guide to gain a firm foundation in key security concepts and WCF. Next, use the application scenarios to evaluate potential designs for your own scenario. The application scenarios are skeletal, end-to-end examples of how you might design your authentication, authorization, and communication from a security perspective. Use the appendix of "Guidelines," "Practices," "How To" articles, and "Questions and Answers" to dive into implementation details. This separation allows you to understand the topics first and then explore the details as you see fit.

Organization of This Guide

You can read this guide from end to end, or you can read only the chapters you need for your job.

Parts

This guide is divided into four parts:

- Part I – Security Fundamentals for Web Services
- Part II – WCF Security Fundamentals
- Part III – Intranet Application Scenarios
- Part IV – Internet Application Scenarios

Part I – Security Fundamentals for Web Services

- Chapter 01 – Security Fundamentals for Web Services
- Chapter 02 – Threats and Countermeasures for Web Services
- Chapter 03 – Security Design Guidelines for Web Services

Part II – WCF Security Fundamentals

- Chapter 04 – WCF Security Fundamentals
- Chapter 05 – Authentication, Authorization, and Identities in WCF
- Chapter 06 – Impersonation and Delegation in WCF
- Chapter 07 – Message and Transport Security
- Chapter 08 – Bindings

Part III – Intranet Application Scenarios

- Chapter 09 – Intranet – Web to Remote WCF Using Transport Security (Original Caller, TCP)
- Chapter 10 – Intranet – Web to Remote WCF Using Transport Security (Trusted Subsystem, HTTP)
- Chapter 11 – Intranet – Web to Remote WCF Using Transport Security (Trusted Subsystem TCP)
- Chapter 12 – Intranet – Windows Forms to Remote WCF Using Transport Security (Original Caller, TCP)

Part IV – Internet Application Scenarios

- Chapter 13 – Internet – WCF and ASMX Client to Remote WCF Using Transport Security (Trusted Subsystem, HTTP)
- Chapter 14 – Internet – Web to Remote WCF Using Transport Security (Trusted Subsystem, TCP)
- Chapter 15 – Internet – Windows Forms Client to Remote WCF Using Message Security (Original Caller, HTTP)

Checklist

- WCF Security Checklist

Guidelines

- WCF Security Guidelines

Practices

- WCF Security Practices at a Glance

Questions and Answers

- WCF Security Questions and Answers (Q&A)

"How To" Articles

- How To – Audit and Log Security Events in WCF Calling from Windows Forms
- How To – Create and Install Temporary Certificates in WCF for Message Security During Development
- How To – Create and Install Temporary Certificates in WCF for Transport Security During Development
- How To – Create and Install Temporary Client Certificates in WCF During Development
- How To – Host WCF in a Windows Service Using TCP
- How To – Impersonate the Original Caller in WCF Calling from a Web Application
- How To – Impersonate the Original Caller in WCF Calling from Windows Forms
- How To – Perform Input Validation in WCF
- How To – Perform Message Validation with Schema Validation in WCF

- How To – Use basicHttpBinding with Windows Authentication and TransportCredentialOnly in WCF from Windows Forms
- How To – Use Certificate Authentication and Message Security in WCF calling from Windows Forms
- How To – Use Certificate Authentication and Transport Security in WCF Calling from Windows Forms
- How To – Use Delegation for Flowing the Original Caller Credentials to the Back-end in WCF Calling from Windows Forms
- How To – Use Health Monitoring to Instrument a WCF Service for Security
- How To – Use netTcpBinding with Windows Authentication and Message Security in WCF from Windows Forms
- How To – Use netTcpBinding with Windows Authentication and Transport Security in WCF from Windows Forms
- How To – Use Protocol Transition for Impersonating and Delegating the Original Caller in WCF
- How To – Use the SQL Server Role Provider with Username Authentication in WCF Calling from Windows Forms
- How To – Use SQL Server Role Provider with Windows Authentication in WCF Calling from Windows Forms
- How To – Use Username Authentication with Custom Authentication and Message Security in WCF Calling from Windows Forms
- How To – Use Username Authentication with the SQL Server Membership Provider and Message Security in WCF Calling from Windows Forms
- How To – Use Username Authentication with Transport Security in WCF Calling from Windows Forms
- How To – Use wsHttpBinding with Username Authentication and TransportWithMessageCredential in WCF Calling from Windows Forms
- How To – Use wsHttpBinding with Windows Authentication and Message Security in WCF Calling from Windows Forms
- How To – Use wsHttpBinding with Windows Authentication and Transport Security in WCF Calling from Windows Forms

Resources

- WCF Security Resources

Feedback and Support

We have made every effort to ensure the accuracy of this guide and its companion content.

Feedback on the Guide

We've provided a short questionaire on the Internet that would only take 5 to 10 minutes max to fill out. Copy these questions to an email message and send the answers to *WCFSec@microsoft.com*.

We are also particularly interested in feedback regarding the following:

- Technical issues specific to recommendations
- Usefulness and usability issues

Any input can be sent in e-mail to *WCFSec@microsoft.com*.

Technical Support

Technical support for the Microsoft products and technologies referenced in this guide is provided by Microsoft Product Support Services (PSS). For product support information, please visit the Microsoft Product Support Web site at http://support.microsoft.com.

Community Support

Microsoft MSDN® Newsgroups:

Forum	Address
Windows Communication Foundation ("Indigo")	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=118&SiteID=1
Architecture General	http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=228&SiteID=1

The Team Who Brought You This Guide

This guide was created by the following team members:

- **J.D. Meier**
- **Carlos Farre**
- **Jason Taylor**
- **Prashant Bansode**
- **Steve Gregersen**
- **Madhu Sundararajan**
- **Rob Boucher**

Contributors and Reviewers

- **External Contributors / Reviewers**: Andy Eunson; Anil John; Anu Rajendra; Brandon Bohling; Chaitanya Bijwe; Daniel Root; David P. Romig, Sr.; Dennis Rea; Kevin Lam; Michele Leroux Bustamante; Parameswaran Vaideeswaran; Rockford Lhotka; Rudolph Araujo; Santosh Bejugam
- **Microsoft Contributors / Reviewers**: Alik Levin; Brandon Blazer; Brent Schmaltz; Curt Smith; David Bradley; Dmitri Ossipov; Jan Alexander; Jason Hogg; Jason Pang; John Steer; Marc Goodner; Mark Fussell; Martin Gudgin; Martin Petersen-Frey; Mike de Libero; Mohammad Al-Sabt; Nobuyuki Akama; Ralph Squillace; Richard Lewis; Rick Saling; Rohit Sharma; Scott Mason; Sidd Shenoy; Sidney Higa; Stuart Kwan; Suwat Chitphakdibodin; T.R. Vishwanath; Todd Kutzke; Todd West; Vijay Gajjala; Vittorio Bertocci; Wenlong Dong; Yann Christensen; Yavor Georgiev

Tell Us About Your Success

If this guide helps you, we would like to know. Tell us by writing a short summary of the problems you faced and how this guide helped you out. Submit your summary to: MyStory@Microsoft.com .

Solutions at a Glance

Summary

This chapter provides an at-a-glance roadmap into the various solutions presented in this guide. Use this chapter to find fast answers to common WCF security-related problems. This roadmap summarizes the solutions presented in this guide and provides links to appropriate material so that you can easily find the information you need and solutions to specific problems.

Security Engineering

- **How to identify and evaluate threats**
  Use threat modeling to systematically identify threats rather than applying security in a haphazard manner. Next, rate the threats based on the risk of an attack or occurrence of a security compromise and the potential damage that could result. This allows you to tackle threats in the appropriate order.

  For more information about creating a threat model and evaluating threat risks, see "Threat Modeling Web Applications" at http://msdn.microsoft.com/en-us/library/ms978516.aspx .

- **How to create secure designs**
  Use tried and tested design principles. Focus on the critical areas where the correct approach is essential and where mistakes are often made. This guide refers to these as *application vulnerability categories*. They include input validation, authentication, authorization, configuration management, sensitive data protection, session management, cryptography, parameter manipulation, exception management, and auditing and logging considerations. Pay serious attention to deployment issues including topologies, network infrastructure, security policies, and procedures.

  You can use the end-to-end application scenarios in this guide to help identify candidate authentication and authorization strategies.

- **How to perform an design inspections**
  Review your application's design in relation to the target deployment environment and associated security policies. Consider the restrictions imposed by the underlying infrastructure layer security, including perimeter networks, firewalls, remote application servers, and so on. Use application vulnerability categories to help partition your application, and analyze the approach taken for each area.

  You can use the guidelines in this guide to create customized guidelines for your teams.

- **How to perform security code inspections**
  You can use the following general technique for performing security inspections:

  1. Identify security code review objectives. Establish goals and constraints for the review.

2. Perform a preliminary scan. Use static analysis to find an initial set of security issues and to improve your understanding of where you will be most likely to find security issues when you review the code more fully.
3. Review the code for security issues. Review the code thoroughly with the goal of finding security vulnerabilities that are common to many applications. You can use the results of step 2 to focus your analysis.
4. Review for security issues unique to the architecture. Complete a final analysis by looking for security issues that relate to the unique architecture of your application. This step is most important if you have implemented a custom security mechanism or any feature designed specifically to mitigate a known security threat.

For more information on performing code inspections, see "How To: Perform a Code Review for Managed Code ("Baseline Activity")" at http://msdn.microsoft.com/en-us/library/ms998364.aspx.

- **How to perform security deployment inspections**
  Inspect your service's run-time behavior and configuration. This includes your service's accounts, ports, and protocols.

Message and Transport Security

- **How to choose between message and transport security**
  The transport-level security model is simple, well understood, and adequate for many (primarily intranet-based) scenarios, in which the transport mechanisms and endpoint configuration can be tightly controlled.

  The main issues with transport-level security are:
  - Security becomes tightly coupled to, and dependent on, the underlying platform, transport mechanism, and security service provider (NTLM, Kerberos, and so on).
  - Security is applied on a point-to-point basis, with no provision for multiple hops and routing through intermediate application nodes.

  Message-level security:
  - Can be independent from the underlying transport.
  - Enables a heterogeneous security architecture.
  - Provides end-to-end security and accommodates message routing through intermediate application nodes.
  - Supports multiple encryption technologies.
  - Supports non-repudiation.

Authentication / Authorization

- **How to design an effective authentication and authorization strategy**
  Use the following pattern to work through your authentication and authorization strategies:
  1. Identify your user stores.
  2. Identify your role stores.

3. Identify resources you need to access and operations you need to perform.
4. Identify which identities need to access the resources and perform the operations.
5. Choose your authentication and authorization strategies.

- **How to authenticate users for intranet applications**
  The most common scenarios for intranet applications include any of the following patterns:
  - Username authentication with the SQL Server membership provider
  - Windows authentication with Active Directory
  - Username authentication with a custom store
  - Certificate authentication with Windows

- **How to authenticate users for Internet applications**
  The most common scenarios for Internet applications include any of the following patterns:
  - Username authentication with the SQL Server membership provider
  - Basic authentication with Active Directory
  - Username authentication with a custom store
  - Certificate authentication with Windows

- **How to authorize callers to perform operations and access resources**
  Consider the following options:
  - If you are using Windows authentication, use **WindowsTokenRoleProvider** for role authorization using Windows groups.
  - If you are using Windows authentication, use **SqlRoleProvider** for role authorization.
  - If you are using Windows authentication, use AzMan policy store in an XML file, in Active Directory, or in Active Directory Application Mode (ADAM). Consider using **AuthorizationStoreRoleProvider** for role authorization.
  - If you are using username authentication with **SqlMembershipProvider**, use **SqlRoleProvider** for role authorization.
  - If you are using username authentication mapped to Windows, use **WindowsTokenRoleProvider** for role authorization using Windows groups.
  - If you are using username authentication mapped to Windows, use AzMan policy store in an XML file, in Active Directory, or in Active Directory Application Mode (ADAM). Consider using **AuthorizationStoreRoleProvider** for role authorization.
  - If you are using certificate authentication with certificates mapped to Windows accounts, use **WindowsTokenRoleProvider** for role authorization using Windows groups.
  - If you are using certificate authentication with certificates mapped to Windows accounts, use AzMan policy store in an XML file, in Active Directory, or in Active Directory Application Mode (ADAM). Consider using **AuthorizationStoreRoleProvider** for role authorization.

- **How to choose effective strategies for authorization**
  You can use the following resource access strategies:
  - **Role-based**. Map users to roles and check whether a role can perform the requested operation.

- **Identity-based**. Authorize users based on their identity.
- **Claims-based**. Grant or deny access to the operation or resources based on the client's claims.
- **Resource-based**. Protect resources using access control lists (ACLs).

- **How to choose between trusted subsystem and impersonation/delegation**
  With the trusted subsystem model, you use the process identity to access downstream network resources such as databases. With impersonation/delegation, you use impersonation and use the original caller's identity to access the database. The trusted subsystem model offers better scalability because your application benefits from efficient connection pooling. You also minimize back-end ACL management. Only the trusted identity can access the database. Your end users have no direct access. In the trusted subsystem model, the service is granted broad access to back-end resources. As a result, a compromised service could potentially make it easier for an attacker to gain broad access to back-end resources. Keeping the service account's credentials protected is essential. With impersonation/delegation, you benefit from operating system auditing because you can track which users have attempted to access specific resources. You can also enforce granular access controls in the database, and individual user accounts can be restricted independently of one another in the database.

- **How to choose between resource-based and role-based authorization**
  Your authorization strategy may also be influenced by your choice of authentication type. Consider the following:

  Resource-based authorization considerations:
  - If you are using certificate authentication, you will need to map certificates to Windows groups.
  - If you are using username authentication, you will need to perform protocol transition.
  - Windows authentication will work with resource-based authorization by default.
  - Basic authentication will work with resource-based authorization by default.
    Note: You need to impersonate for resource-based authorization.

  Role-based authorization considerations:
  - If you are using certificate authentication, you will need to map certificates to Windows groups.
  - If you are using username authentication with Windows groups, you will need to perform protocol transition.
  - Username authentication will work with ASP.NET roles by default.
  - Windows authentication will work with Windows groups by default.
  - Basic authentication will work with Windows groups by default.

Patterns

- **How to leverage Web services security patterns**

Familiarize yourself with the following patterns, then evaluate and apply the patterns when they make sense for your particular scenario:

- Brokered Authentication
- Brokered Authentication: Kerberos
- Brokered Authentication: X509 PKI
- Brokered Authentication: STS
- Data Confidentiality
- Data Origin Authentication
- Direct Authentication
- Exception Shielding
- Message Replay Detection
- Message Validator
- Perimeter Service Router
- Protocol Transition with Constrained Delegation
- Trusted Subsystem

For information on the patterns above, see the patterns & practices "Web Services Security" guide at http://msdn.microsoft.com/en-us/library/aa480545.aspx.

Auditing and Logging

- **How to enable auditing in WCF**
  You can enable auditing in the configuration file.

- **How to instrument your WCF service**
  You can use ASP.NET Health Monitoring.

- **How to improve your auditing in WCF**
  Audit for the following:
  - **User management events**. Instrument your application and monitor user-management events such as password resets, password changes, account lockout, user registration, and authentication events. Doing this helps you to detect and react to potentially suspicious behavior. It also enables you to gather operations data; for example, to track who is accessing your application and when user account passwords need to be reset.
  - **Unusual or suspicious activity**. Instrument your application and monitor events that might indicate unusual or suspicious activity. This enables you to detect and react to potential problems as early as possible. Unusual activity could include replays of old authentication tickets or too many login attempts over a specific period of time.
  - **Significant business operations**. Track significant business operations. For example, instrument your application to record access to particularly sensitive methods and business logic.

Bindings

- **How to choose the right WCF binding**
  Consider the following scenarios:
  - If you need to support clients over the Internet, consider using **wsHttpBinding**.
  - If you need to expose your WCF service to legacy clients such as an ASMX Web service, use **basicHttpBinding**.
  - If you need to support WCF clients within an intranet, consider using **netTcpBinding**.
  - If you need to support WCF clients on the same machine, consider using **netNamedPipeBinding**.
  - If you need to support disconnected queued calls, use **netMsmqBinding**.
  - If you need to support bidirectional communication between the WCF client and WCF service, use **wsDualHttpBinding** or **netTcpBinding**.

- **How to create a custom binding**
  To create a custom binding, in the WCF configuration file, select a set of binding elements that are supposed to be constructed in a specific order. Those binding elements refer to transaction, reliable message, security, encoding formats, and transport protocol.

- **How to support multiple authentication and authorization strategies**
  Use multiple bindings to support multiple authentication and authorization strategies. For instance, you could use **basicHttpBinding** with username authentication to support legacy ASMX clients, and **wsHttpBinding** with Windows authentication to support newer WCF-enabled clients.

Exception Management

- **How to handle exceptions in WCF**
  Use fault contracts to handle exceptions in WCF. By using the **FaultContract** attribute in a service contract, you can specify the possible faults that can occur in your WCF service. This prevents you from exposing any other exception details to the clients.
  - Apply the **FaultContract** attribute directly on a contract operation, specifying the exception type that can be thrown as shown in the following example:

```
[OperationContract]
[FaultContract(typeof(DivideByZeroException))]
double Divide(double number1,double number2);
```

Impersonation / Delegation

- **How to impersonate at the service level**
  You can impersonate the entire service by setting the **impersonateCallerForAllOperations** attribute to "true" in the WCF configuration file. If you are impersonating all operations in

the service, the **Impersonation** property of the **OperationBehaviorAttribute** applied to each operation will override. Therefore if the property on the operation is set to something other than **Allowed** or **Required**, impersonation will be turned off for that operation.

- **How to impersonate at the operation level**
  You can impersonate declaratively by applying the **OperationBehavior** attribute on any operation that requires client impersonation. Use impersonation selectively and only on the operations that need it, since by nature impersonation increases the potential attack surface of your application.

- **How to flow the original caller to the back end (double hop)**
  If your WCF service runs under the Network Service account, configure your computer account in Active Directory to be trusted for delegation. If your application runs under a custom domain account, you must register a service principal name (SPN) in Active Directory in order to associate the domain account with the HTTP service on your WCF server. You then configure your domain account in Active Directory to be trusted for delegation.
  Impersonate the original caller imperatively or declaratively – before you access the back-end resource, the original caller will be delegated to be authenticated and authorized at the back end.

Message Validation

- **How to perform parameter validation**
  Use parameter inspectors to validate for length, range, format, and type. You can validate parameters on both the client and the service. The server should not trust client-side validation, but you can use it to reduce round-trips for incorrect input. The following are the key steps you need to perform:
  1. Write a class that implements a parameter inspector.
  2. Write class that implements endpoint behavior.
  3. Write a class that implements a behavior element.
  4. Add the behavior element as an extensibility point in the WCF configuration file.
  5. Create an endpoint behavior that uses the behavior element as an extensibility point.
  6. Configure the endpoint to use the endpoint behavior.

- **How to perform message validation**
  Use schemas and regular expressions to validate for length, range, format, and type. Schemas are preferred for validating complex types (classes and message contracts). For performance reasons, you will want to load the schema from the cache (in the Message Inspector). You can validate incoming and outgoing messages on the server side as well as incoming and outgoing messages on the client side. The server should not trust client-side validation, but you can use it to reduce round-trips for incorrect input.
  The following are the key steps you need to perform:
  1. Write a class that implements Message inspector.
  2. Write a class that implements endpoint behavior.

3. Write a class that implements a behavior element.
4. Add the behavior element as an extensibility point in the WCF configuration file.
5. Create an endpoint behavior that uses the behavior element as an extensibility point.
6. Configure the endpoint to use the endpoint behavior.

Fast Track – A Guide for Getting Started and Applying the Guidance

Summary

This “fast track” chapter highlights the basic approach taken by this guide to help you design and develop WCF applications with security in mind. Use this chapter to understand the basic approach, security engineering activities, key scenarios, the security frame, and best practices for the development of secure WCF applications with security.

Goal and Scope

This guide shows you how to design and build secure Web services with WCF.

It includes:

- End-to-end application scenarios
- Guidelines
- Step-by-step How To articles

The Approach

The keys to building secure services include:

- **Identify your security objectives**. This includes identifying your particular security requirements.
- **Know your threats**. Know which threats are relevant for your scenarios and context. Threat modeling is an effective technique for helping you identify relevant threats and vulnerabilities. Your objectives will help you prioritize your threats and vulnerabilities. Based on the threat model, developers address vulnerabilities, and testers verify that the developers closed the issues.
- **Apply proven principles, patterns, and practices**. By using proven principles, patterns, and practices, you can eliminate classes of security problems. You can also leverage lessons learned. Patterns are effectively reusable solutions and typically encapsulate underlying principles. While principles, patterns, and practices are a good starting point, you should never blindly adopt them — you need to evaluate whether they make sense for your specific scenario.
- **Apply effective security engineering throughout the application life cycle**. It is important to consider security throughout your application life cycle. You should start by setting your security objectives. Threat modeling will help shape your design and make key trade-offs. Security design, code, and deployment inspections, along with testing, will improve your overall security posture.

patterns & practices Security Engineering

The Microsoft patterns & practices Security Engineering approach includes specific security-related activities that help you meet your application security objectives.

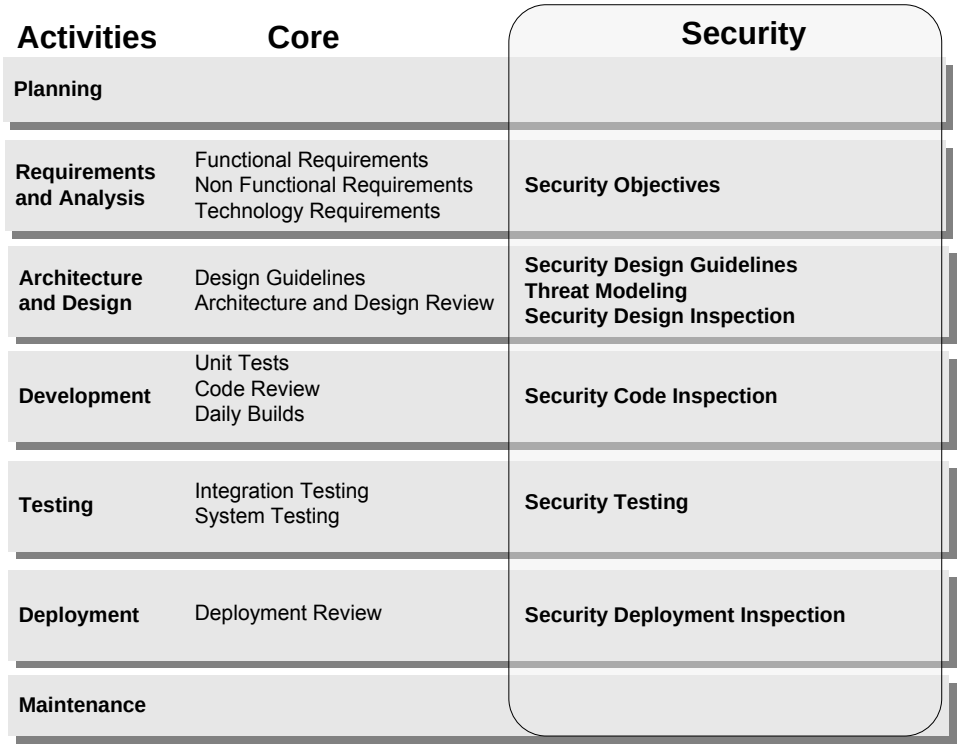


Figure 1. Security Engineering Activities

Summary of Key Security Engineering Activities

The Microsoft patterns & practices Security Engineering approach extends these proven core activities to create security-specific activities. These activities include:

- **Security objectives**. Setting objectives helps you scope and prioritize your work by setting boundaries and constraints. Setting security objectives helps you identify where to start, how to proceed, and when you are done.
- **Threat modeling**. *Threat modeling* is an engineering technique that can help you identify threats, attacks, vulnerabilities, and countermeasures that could affect your application. You can use threat modeling to shape your application's design, meet your company's security objectives, and reduce risk.
- **Security design guidelines**. Creating design guidelines is a common practice at the start of an application project, in order to guide development and share knowledge across the team. Effective design guidelines for security organize security principles, practices, and patterns by actionable categories.
- **Security design inspection**. Security design inspections are an effective way to identify problems in your application design. By using pattern-based categories and a question-driven approach, you simplify evaluating your design against root-cause security issues.

- **Security code inspection**. Many security defects are found during code reviews. Analyzing code for security defects includes knowing what to look for and how to look for it. Security code inspections optimize inspecting code for common security issues.
- **Security testing**. Use a risk-based approach and use the output from the threat modeling activity to help establish the scope of your testing activities and define your test plans.
- **Security deployment inspection**. When you deploy your application during your build process or staging process, you have an opportunity to evaluate your application's run-time characteristics in the context of your infrastructure. Deployment reviews for security focus on evaluating your security design and the configuration of your application, host, and network.

For more information on security engineering see, "patterns & practices Security Engineering Explained" at http://msdn.microsoft.com/en-us/library/ms998382.aspx#

End-to-End Scenarios

Intranet

The following figure is an example of a common WCF intranet scenario. Note the use of the TCP protocol. WCF is hosted by the Windows service, and Windows authentication is used to authenticate users inside the Windows domain.

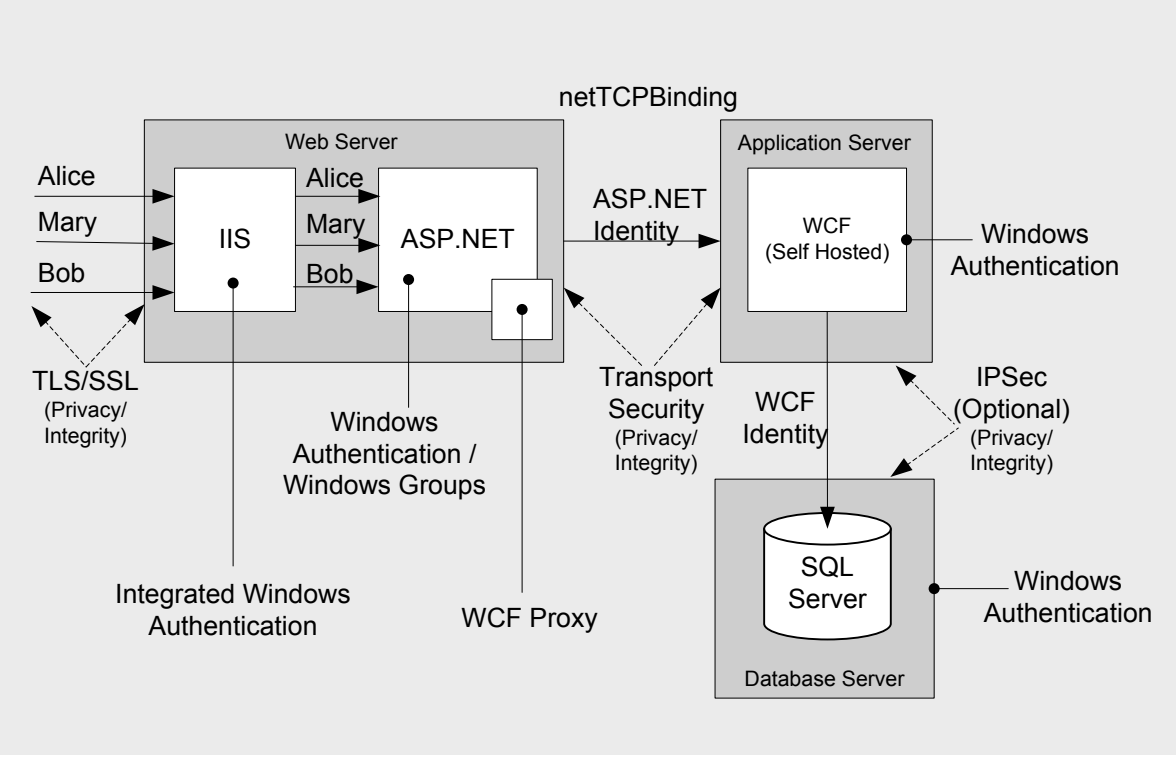


Figure 2. A Common WCF Intranet Scenario

Internet

The following figure is an example of a common WCF Internet scenario. Note the use of the HTTP protocol. WCF is hosted in Internet Information Services (IIS), and Username authentication is used to authenticate users.

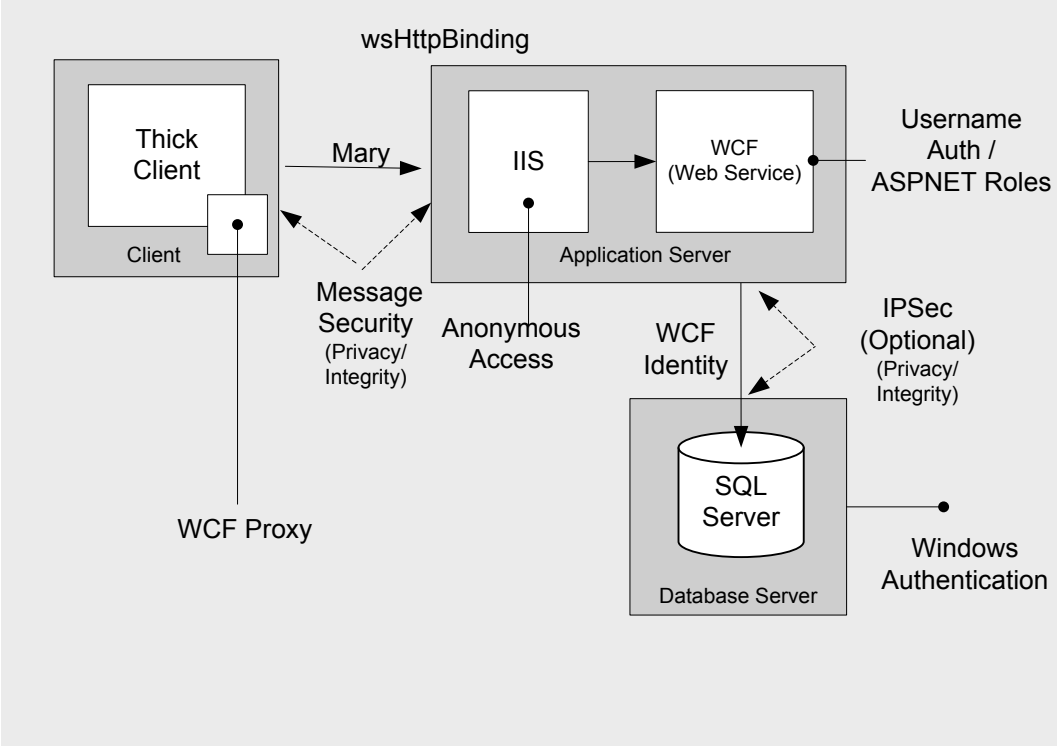


Figure 3. A Common WCF Internet Scenario

Web Services Security Frame

The following key security concepts provide a frame for thinking about security when designing and architecting services. This helps you turn core security features such as authentication, authorization, auditing, confidentiality, integrity, and availability into action.

Category	Description
Auditing and Logging	*Auditing and logging* refers to how security-related events are recorded, monitored, and audited.
Authentication	*Authentication* is the process in which an entity proves the identity of another entity, typically through credentials, such as a username and password.
Authorization	*Authorization* is how your service provides access controls for resources and operations.
Configuration Management	*Configuration management* refers to how your service handles database connections, administration, and other configuration

	settings.
Exception Management	*Exception management* refers to how you handle exceptions within your application, including fault contracts.
Impersonation/Delegation	*Impersonation and delegation* refers to how your service impersonates users and passes identity information downstream for authorization purposes.
Message Encryption	*Message encryption* refers to protecting a message by converting the contents to cipher text by using cryptographic methods.
Message Replay Detection	*Message replay detection* refers to identifying and rejecting messages that are resubmitted.
Message Signing	*Message signing* refers to signing a message with a digital signature using cryptographic methods, to confirm the source of the message and detect if the contents have been tampered with (i.e., authentication and integrity of the message).
Message Validation	*Message validation* refers to how you verify the message payload against a schema, as well as message size, content, and character sets. This includes how your service filters, scrubs, and rejects input and output before additional processing. Input and output includes input from clients consuming the service as well as file-system input, in addition to input from network resources, such as databases. Output typically includes the return values from your service or disk/database writes, among others.
Sensitive Data	*Sensitive data* is user and application data whose integrity and confidentiality need to be protected. This includes how you protect sensitive data from being stolen from memory, from configuration files, or when transmitted over the network.
Session Management	A *session* refers to a series of related interactions between a client and your service.

Threats and Attacks to Your Web Services

The following table highlights some of the common threats and attacks against Web services.

Category	Description
Auditing and Logging	• Tampering with log files • Ineffectual or nonexistent audit processes
Authentication	• Network eavesdropping • Brute force attacks • Dictionary attacks • Cookie replay attacks • Credential theft
Authorization	• Elevation of privilege • Disclosure of confidential data

	• Data tampering • Luring attacks
Configuration Management	• Unauthorized access to administration interfaces • Unauthorized access to configuration stores • Retrieval of clear text • Configuration secrets • No individual accountability
Exception Management	• System or application details are revealed • Denial of service (DoS)
Impersonation/Delegation	Elevation of privilege
Message Encryption	Information disclosure
Message Replay Detection	Horizontal and vertical privilege escalation
Message Signing	Data tampering
Message Validation	• Buffer overflows • Cross-site scripting • SQL injection • Canonicalization attacks
Sensitive Data	• Accessing of sensitive data in storage • Network eavesdropping • Information disclosure
Session Management	• Session hijacking • Session replay • Man-in-the-middle attacks

Guidelines for Your Web Services

The following table summarizes effective guidelines to improve the security of your Web services.

Category	Description
Auditing and Logging	• Identify malign or malicious behavior. • Know your baseline (e.g., what does good traffic look like?). • Instrument to expose behavior that can be watched. (The big mistake here is typically that application instrumentation is completely missing. • Create a process to watch the logs and an escalation path for significant issues.
Authentication	• Use strong password policies. • Do not store credentials on the client side. • Do not store credentials in clear text on the server side. • Encrypt communication channels to secure authentication tokens. • Use secure protocols such as Secure HTTP (HTTPS) to secure

	authentication tokens.
Authorization	• Use least-privileged accounts. • Consider granularity of access. • Enforce separation of privileges. • Use role-based access control.
Configuration Management	• Use least-privileged service accounts. • Do not store credentials in plaintext format. • Use strong authentication and authorization on administrative interfaces. • Do not use the Local Security Authority (LSA). • Avoid storing sensitive information in the Web space or in configuration files, especially in clear text.
Exception Management	• Use structured exception handling (try-catch-finally). • Only catch and wrap exceptions if the operation adds value/information. • Do not reveal sensitive system or application information. • Do not log private data (passwords, etc.). • Use the **finally** block to perform cleanup. • Be cognizant of exception filters.
Impersonation/Delegation	• Use constrained delegation. • Do not hard-code credentials in your code and preferably not in the configuration files. • Use IIS application domains or Windows service accounts for the host. • Encrypt credentials; if you do, put them in configuration files.
Message Encryption	Use strong algorithms with appropriate cipher modes, key management, key length, etc.
Message Replay Detection	• Enable replay detection within WCF. • Use nonces and unique tokens to detect replay or unauthorized requests.
Message Signing	• Use strong algorithms with appropriate padding modes, key management, key length, etc. • Avoid use of self-signed certificates.
Message Validation	• Use schema validation. • Offload schema validation to an XML accelerator if possible. • Use parameter validation.
Sensitive Data	• Do not store secrets in software. • Enforce separation of privileges. • Encrypt sensitive data over the network. • Secure the channel. • Avoid key management. • Cycle your keys.

Session Management	• Partition services by anonymous, identified, and authenticated users;. • Reduce session timeouts. • Avoid storing sensitive data in session stores. • Secure the channel to the session store. • Authenticate and authorize access to the session store.

Web Services Security Patterns

The following table summarizes Web services security patterns and provides links to more information.

Pattern	Description	Reference
Authentication		
Direct Authentication	The Web service acts as an authentication service to validate credentials from the client. The credentials, which include proof-of-possession that is based on shared secrets, are verified against an identity store.	http://msdn.microsoft.com/en-us/library/aa480566.aspx
Brokered Authentication	The Web service validates the credentials presented by the client, without the need for a direct relationship between the two parties. An authentication broker that both parties trust independently issues a security token to the client. The client can then present credentials, including the security token, to the Web service.	http://msdn2.microsoft.com/en-us/library/aa480560.aspx
Brokered Authentication: Kerberos	Use the Kerberos protocol to broker authentication between clients and Web services.	http://msdn2.microsoft.com/en-us/library/aa480562.aspx
Brokered Authentication: X509 PKI	Use brokered authentication with X.509 certificates issued by a certificate authority (CA) in a public key infrastructure (PKI) in order to verify the credentials presented by the requesting application.	http://msdn2.microsoft.com/en-us/library/aa480565.aspx
Brokered Authentication: Security Token	Use brokered authentication with a security token issued by an STS. The STS is trusted by both the client and the Web service to	http://msdn2.microsoft.com/en-us/library/aa480563.aspx

Service (STS)	provide interoperable security tokens.	
Authorization		
Protocol Transition with Constrained Delegation	Use the Kerberos protocol extensions in Windows Server. The extensions require the user ID but not the password. You still need to establish trust between the client application and the Web service; however, the application is not required to store or send passwords.	http://msdn.microsoft.com/en-us/library/aa480585.aspx
Trusted Subsystem	The Web service acts as a trusted subsystem to access additional resources. It uses its own credentials instead of the user's credentials to access the resource.	http://msdn2.microsoft.com/en-us/library/aa480587.aspx
Exception Management		
Exception Shielding	Sanitize unsafe exceptions by replacing them with exceptions that are safe by design. Return only those exceptions to the client that have been sanitized or exceptions that are safe by design. Exceptions that are safe by design do not contain sensitive information in the exception message, and they do not contain a detailed stack trace, either of which might reveal sensitive information about the Web service's inner workings.	http://msdn2.microsoft.com/en-us/library/aa480591.aspx
Message Encryption		
Data Confidentiality	Use encryption to protect sensitive data that is contained in a message. Unencrypted data, which is known as *plaintext*, is converted to encrypted data, which is known as *cipher text*. Data is encrypted with an algorithm and a cryptographic key. Cipher text is then converted back to plaintext at its destination.	http://msdn.microsoft.com/en-us/library/aa480570.aspx
Message Replay Detection		
Message Replay Detection	Cache an identifier for incoming messages, and use message replay detection to identify and reject messages that match an entry in the replay detection cache.	http://msdn2.microsoft.com/en-us/library/aa480598.aspx
Message Signing		
Data Origin Authentication	Use data origin authentication, which enables the recipient to verify that	http://msdn2.microsoft.com/en-us/library/aa480571.aspx

	messages have not been tampered with in transit (data integrity) and that they originate from the expected sender (authenticity).	
Message Validation		
Message Validator	The message validation logic enforces a well-defined policy that specifies which parts of a request message are required for the service to successfully process it. It validates the XML message payloads against an XML schema (XSD) to ensure that they are well-formed and consistent with what the service expects to process. The validation logic also measures the messages against certain criteria by examining the message size, the message content, and the character sets that are used. Any message that does not meet the criteria is rejected.	http://msdn2.microsoft.com/en-us/library/aa480600.aspx
Deployment		
Perimeter Service Router	Design a Web service intermediary that acts as a perimeter service router. The perimeter service router provides an external interface on the perimeter network for internal Web services. It accepts messages from external applications and routes them to the appropriate Web service on the private network.	http://msdn2.microsoft.com/en-us/library/aa480606.aspx

Bindings in WCF

The following table summarizes common bindings in WCF.

Binding	Description
basicHttpBinding	Configures and exposes endpoints that are able to communicate with ASP.NET Web Services (ASMX)–based Web services and clients and other services that conform to the WS-I Basic Profile 1.1 specification. By default, it has security disabled.
wsHttpBinding	Defines a secure, reliable, interoperable binding suitable for non-duplex service contracts. The binding implements the following specifications: WS-Reliable Messaging for reliability, and WS-Security for message security and authentication. The

	transport is HTTP, and message encoding is text/XML encoding. By default, it provides message security with Windows authentication.
ws2007HttpBinding	Defines a secure, reliable, interoperable binding suitable for non-duplex service contracts. The binding implements the following specifications: WS-Reliable Messaging for reliability, and WS-Security for message security and authentication. The transport is HTTP, and message encoding is text/XML encoding. The **ws2007HttpBinding** provides binding similar to **wsHttpBinding** but uses the standard for OASIS (Organization for the Advancement of Structured Information Standards). By default, it provides message security with Windows authentication.
netTcpBinding	Specifies a secure, reliable, optimized binding suitable for cross-machine communication. By default, it generates a run-time communication stack with transport security and Windows authentication as default security settings. It uses TCP protocol for message delivery, and binary message encoding.
netNamedPipeBinding	Defines a binding that is secure, reliable, optimized for cross-process communication on the same machine. By default, it generates a run-time communication stack with WS-ReliableMessaging for reliability, transport security for transfer security, named pipes for message delivery, and binary message encoding. It is not secured by default.
netMsmqBinding	Defines a queued binding suitable for cross-machine communication.
wsFederationHttpBinding	Defines a binding that supports federated security. It helps implement federation, which is the ability to flow and share identities across multiple enterprises or trust domains for authentication and authorization. WCF implements federation over message and mixed mode security but not over transport security. Services configured with this binding must use the HTTP protocol as transport.
ws2007FederationHttpBinding	Defines a binding that derives from **wsFederationHttpBinding** and supports federated security. It helps implement federation, which is the ability to flow and share identities across multiple enterprises or trust domains for authentication and authorization. WCF implements federation over message and mixed mode security but not over transport security. Services configured with this binding must use the HTTP protocol as transport. The **ws2007FederationHttpBinding** provides binding similar to

	ws2007FederationHttpBinding but uses the OASIS standard.
wsDualHttpBinding	Defines a secure, reliable, and interoperable binding that is suitable for duplex service contracts or communication through Simple Object Access Protocol (SOAP) intermediaries.
customBinding	Allows you to create a custom binding with full control over the message stack.

Transport Security

When using transport security, the user credentials and claims are passed by using the transport layer. In other words, user credentials are transport-dependent, which allows fewer authentications options compared to message security. Each transport protocol (TCP, IPC, MSMQ, or HTTP) has its own mechanism for passing credentials and handling message protection. The most common approach is to use Secure Sockets Layer (SSL) for encrypting and signing the contents of the packets sent over HTTPS.

Transport security is used to provide point-to-point security between the two endpoints (service and client). If there are intermediary systems between client and the service, each intermediate point must forward the message over a new SSL connection.

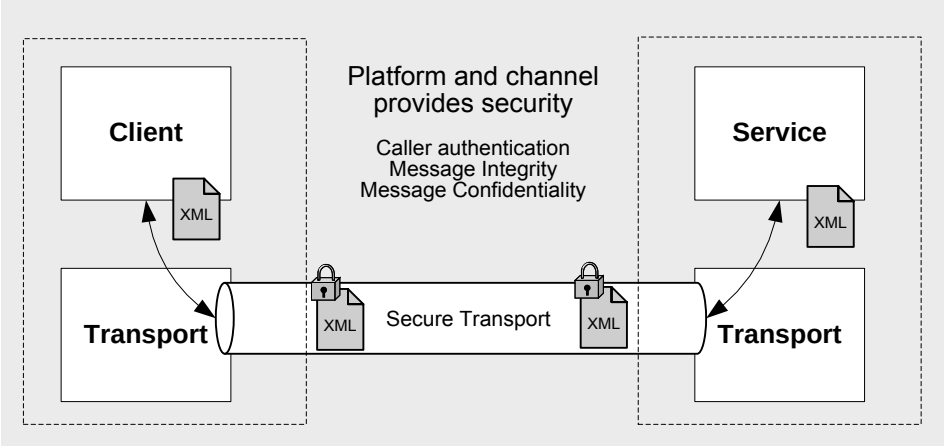


Figure 4. Transport Security

Use transport security for the following scenarios:

- You are sending a message directly from your application to a WCF service, and the message will not be routed through intermediate systems.
- You have both the service and the client in an intranet.

Using transport security offers the following advantages:

- It provides interoperability, meaning that communicating parties do not need to understand the WS-Security specifications.
- It may result in better performance.

- Hardware accelerators can be used to further improve performance.

Using transport security has the following disadvantages:

- Security is applied on a point-to-point basis, with no provision for multiple hops or routing through intermediate application nodes.
- It supports a limited set of credentials and claims compared to message security.
- It is transport-dependent upon the underlying platform, transport mechanism, and security service provider, such as NTLM or Kerberos.

Message Security

When using message security, the user credentials and claims are encapsulated in every message by using the WS-Security specification to secure messages. This option gives the most flexibility from an authentication perspective. You can use any type of security credentials you want, largely independent of transport, as long as both the client and service agree.

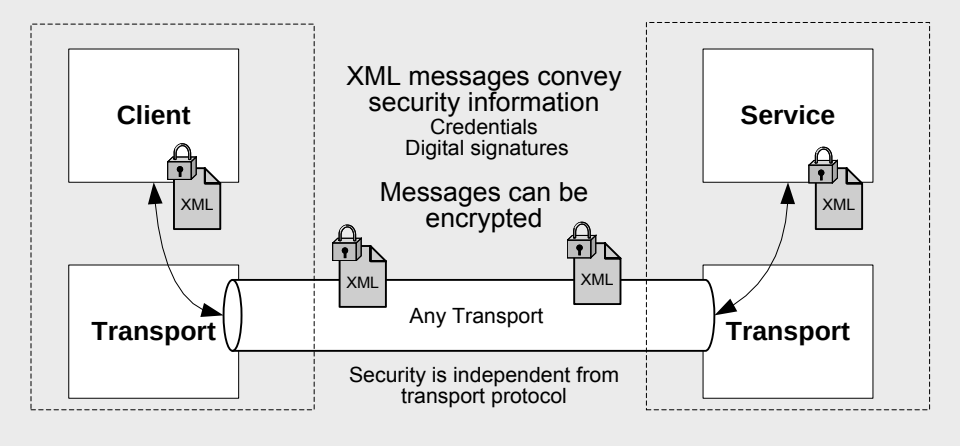


Figure 5. Message Security

Use message security for the following scenarios:

- You are sending a message to a WCF service, and the message is likely to be forwarded to other WCF services or may be routed through intermediate systems.
- Your WCF clients are accessing the WCF service over the Internet.

Using message security offers following advantages:

- It provides end-to-end security; because message security directly encrypts and signs the message, having intermediaries does not break the security.
- It allows partial or selective message encryption and signing, thus improving overall application performance.
- Message security is transport-independent and thus can be used with any transport protocol.

- It supports a wide set of credentials and claims, including issue tokens, which enable federated security.

Using message security has following disadvantages:

- This option may reduce performance compared to transport security because each individual message is encrypted and signed.
- It does not support interoperability with older ASMX clients since it requires both the client and service to support WS-Security specifications.

Authentication

Transport Security

The follow authentication options are available when using transport security mode:

- **None**. When using this option, the WCF service does not authenticate the callers. This is not the recommended option from security perspective — avoid using this option wherever possible.
- **Basic**. This option is available with the HTTP protocol only. The client is authenticated by using the username and password against Active Directory. The client credentials are transported by using Base64 encode string, which is literally like clear string and therefore is not the most secure option. The service is authenticated by the SSL certificate used for secure communication.
- **NTLM**. This option is available with the HTTP protocol only. The client is authenticated by using a challenge-response scheme against Windows accounts. The NTLM option is well suited for a workgroup environment. NTLM authentication is more secure than either Digest or Basic authentication. The service is authenticated by using the Windows credentials of the process identity, or by using an SSL certificate if you are using the HTTP protocol.
- **Windows**. The Windows option tells the WCF service to use Kerberos when in a domain or NTLM when deployed in a workgroup environment. This option uses a Windows token presented by the caller to authenticate against Active Directory. This is the most secure option compared to Basic, Digest, or NTLM authentication. The service is authenticated by using the Windows credentials of the process identity or an SSL certificate if you are using the HTTP protocol.
- **Certificate**. When using this option, the caller presents an X.509 client certificate that the WCF service either validates with peer trust or trusts based on the issuer of the certificate. This option should be used when Windows authentication is not possible, as in the case of business-to-business (B2B) scenarios. The service is authenticated with the service certificate or by using an SSL certificate if you are using the HTTP protocol.

Message Security

The follow authentication options are available when using message security mode:

- **None**. When using this option, the WCF service does not authenticate the callers. This is not the recommended option from security perspective — avoid using this option wherever possible.
- **Windows**. When using this option, the WCF service uses Kerberos when in a domain or NTLM when deployed in workgroup environment. This option uses the Windows token presented by the caller to authenticate against Active Directory. The service is authenticated by using the Windows credentials of the process identity.
- **Username**. When using this option, the caller provides a username and password to the service. The service can then authenticate against Windows, use a membership providers such as **SqlMembershipProvider**, or use a custom validator to validate against the custom store. You should choose this option only when Windows authentication is not possible. The service is authenticated with a service certificate.
- **Certificate**. When using this option, the caller presents an X.509 client certificate. The WCF service then looks up the certificate information on the host side and either validates it (peer trust) or trusts the issuer (chain trust) of the client certificate. This option should be used when Windows authentication is not possible, or in case of B2B scenarios. Service is authenticated with the service certificate.
- **Issue token**. When using this option, the client and service depend on STS to issue tokens that the client and service trusts. CardSpace is a typical example of STS.

Authorization Options in WCF

WCF supports three basic authorization approaches:

- **Role-based**. Access to WCF operations is secured based on the role membership of the caller. Roles are used to partition your application's user base into sets of users that share the same security privileges within the application; for example, Senior Managers, Managers, and Employees .Users are mapped to roles, and if the user is authorized to perform the requested operation, the application uses fixed identities with which to access resources. These identities are trusted by the respective resource managers; for example, databases, the file system. and so on.
- **Identity-based**. WCF supports an Identity Model feature, which is an extension of role-based authorization. Identity Model enables you to manage claims and policies in order to authorize clients. With this approach, you can verify claims contained within the authenticated users' credentials. These claims can be compared with the set of authorization policies for the WCF service. Depending on the claims provided by the client, the service can either grant or deny access to the operation or resources. Identity Model is useful for fine-grained authorization and is most beneficial when using issue token authentication.
- **Resource-based**. Individual resources are secured by using Windows access control lists (ACLs). The WCF service impersonates the caller prior to accessing resources, which allows the operating system to perform standard access checks. All resource access is performed by using the original caller's security context. This impersonation approach severely impacts

application scalability, because it means that connection pooling cannot be used effectively within the application's middle tier.

In enterprise-level applications where scalability is essential, a role-based or identity based approach to authorization represents the best choice. For small-scale intranet applications that serve per-user content from resources (such as files) that can be secured with Windows ACLs, a resource-based approach may be appropriate.

PART I

Security Fundamentals for Web Services

In This Part:

- **Security Fundamentals for Web Services**
- **Threats and Countermeasures for Web Services**
- **Security Design Guidelines for Web Services**

Chapter 1 – Security Fundamentals for Web Services

Objectives

- Understand the key security requirements.
- Understand the difference between threats, attacks, vulnerabilities, and countermeasures.
- Understand the key distinctions for Service-Oriented Architecture (SOA).
- Understand the Web Services Security Frame.
- Understand the key principles and patterns for building secure services.

Overview

Building secure services includes knowing the threats you face, making effective trade-offs, and integrating security throughout your software development life cycle. One of the most effective ways to deal with security is to leverage proven principles, patterns, and practices. The key is to know which principles, patterns, and practices are effective for your particular situation. Techniques such as threat modeling and security inspections can you help you shape your software to meet your specific security objectives.

What Do We Mean by Security?

Security is fundamentally about protecting assets. Assets may be tangible items, such as operations or your customer database — or they may be less tangible, such as your company's reputation.

It is important to recognize that security is a path, not a destination. As you analyze your infrastructure and applications, you identify potential threats and understand that each threat presents a degree of risk. Security is about risk management and implementing effective countermeasures. One of the most important concepts in security is that effective security is a combination of people, process, and technology.

The Foundations of Security

Security relies on the following elements:

- **Authentication**. Authentication addresses the question: who are you? It is the process of uniquely identifying the clients of your applications and services. These might be end users, other services, processes, or computers. In security parlance, authenticated clients are referred to as *principals*.
- **Authorization**. Authorization addresses the question: what can you do? It is the process that governs the resources and operations that the authenticated client is permitted to access. Resources include files, databases, tables, rows, and so on, together with system-level resources such as registry keys and configuration data. Operations include performing transactions such as purchasing a product, transferring money from one account to another, or increasing a customer's credit rating.

- **Auditing**. Effective auditing and logging is the key to non-repudiation. Non-repudiation guarantees that a user cannot deny performing an operation or initiating a transaction. For example, in an e-commerce system, non-repudiation mechanisms are required to make sure that a consumer cannot deny ordering 100 copies of a particular book.
- **Confidentiality**. Confidentiality, also referred to as privacy, is the process of making sure that data remains private and confidential, and that it cannot be viewed by unauthorized users or eavesdroppers who monitor the flow of traffic across a network. Encryption is frequently used to enforce confidentiality. Access control lists (ACLs) are another means of enforcing confidentiality.
- **Integrity**. Integrity is the guarantee that data is protected from accidental or deliberate (malicious) modification. Like privacy, integrity is a key concern, particularly for data passed across networks. Integrity for data in transit is typically provided by using hashing techniques and message authentication codes.
- **Availability**. From a security perspective, availability means that systems remain available for legitimate users. The goal for many attackers with denial of service (DoS) attacks is to crash an application or to make sure that the application is sufficiently overwhelmed so that other users cannot access it.

Threats, Vulnerabilities, and Attacks Defined

When thinking about security, it is helpful to think in terms of assets, threats, vulnerabilities, and attacks.

- **Asset**. An *asset* is something related to your application that is worth protecting. Sensitive data, intellectual property, and access to critical operations are all assets. For example, user credit card numbers are an asset worth protecting in your application.
- **Threat**. A *threat* is any potential occurrence, malicious or otherwise, that could harm an asset. In other words, a threat is any bad thing that can happen to your assets.
- **Vulnerability**. A *vulnerability* is a weakness that makes a threat possible. This may be because of poor design, configuration mistakes, or inappropriate and insecure coding techniques. Weak input validation is an example of an application layer vulnerability, which can result in input attacks.
- **Attack**. An *attack* is an action that exploits vulnerability or enacts a threat. Examples of attacks include sending malicious input to an application, or flooding a network in an attempt to deny service.

To summarize, a threat is a potential event that can adversely affect an asset, whereas a successful attack exploits vulnerabilities in your system.

What Is a Service?

A *service* is a public interface that provides access to a unit of functionality. Services literally provide some programmatic ‘service’ to the caller who consumes them. Services are loosely coupled and can be combined from within a client or from within other services to provide more complex functionality. Services are distributable and can be accessed from a remote

machine as well as from the local machine on which they are running. Services are message-oriented, meaning that service interfaces are defined by a Web Services Description Language (WSDL) file and operations are called using XML-based message schemas that are passed over a transport. Services support a heterogeneous environment by focusing interoperability at the message/interface definition. If components can understand the message and interface definition, they can use the service regardless of their base technology.

Common Services Scenarios

Services are flexible by nature and can be used in a wide variety of scenarios and combinations. The following are key scenarios that we will return to many times over the course of this guide:

- **Service exposed over the Internet.** This scenario describes a service that is consumed by Web applications or smart client applications over the Internet. Authentication and authorization decisions have to be made based upon Internet trust boundaries and credentials options. For example, username authentication is more likely in the Internet scenario than the intranet scenario. This scenario includes business-to-business as well as consumer-focused services. For example, a Web site that allows scheduling of your family's doctor visits could be included in this scenario.
- **Service exposed over an intranet.** This scenario describes a service that is consumed by Web applications or smart client applications over an intranet. Authentication and authorization decisions have to be made based upon intranet trust boundaries and credentials options. For example, an Active Directory user store is more likely in the intranet scenario than in the Internet scenario. An enterprise Web-mail application could be included in this scenario.
- **Service exposed on the local machine.** This scenario describes a service that is consumed by an application on the local machine. Transport and message protection decisions must be based on local machine trust boundaries and users.
- **Mixed scenario.** This scenario describes a service that is consumed by multiple applications over the Internet, an intranet, and/or the local machine. For example, a line-of-business (LOB) application that is consumed internally by a thick client application and over the Internet by a Web application could be included in this scenario.

Service-Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) is an architecture of loosely coupled components that can be distributed across platform, technology, and physical topologies. Service components can be combined to provide a business process, or to provide more complex services for a client application. Services are the preferred communication technique across application boundaries, including platform, deployment, and trust boundaries.

The key attributes of SOA are:

- **Interoperable.** Components can be interoperable across platform and technology boundaries.

- **Componentized.** Services are exposed as autonomous components that can be versioned and managed independently.
- **Composable.** Services can be composed by an application to perform more complex operations or to enact a business process.
- **Message-based interfaces.** Interfaces are defined by message contracts and schemas. Operation calls and parameters are passed in XML message envelopes.
- **Distributable.** Service components can be consumed from the same machine or distributed to remote machines. The service interface and logic is independent of the transport and protocol used to access the service.
- **Discoverable.** Services publish their metadata as WSDL so that client applications can discover the interfaces and schemas and generate a client-side proxy to consume the service.

SOA Tenants

You can further define the SOA attributes based on a set of four SOA tenets. Microsoft architect Don Box was the first to provide this set of design tenets that govern SOA:

- **Boundaries are explicit**. Operations are called over well-defined boundaries, passing explicitly defined messages.
- **Services are autonomous.** Each service is maintained, developed, deployed, and versioned autonomously.
- **Services share schema and contract, not class.** Services share contracts and schemas to communicate.
- **Compatibility is based upon policy.** *Policy* in this case means definition of transport, protocol, security, etc.

Service Orientation vs. Object Orientation

Services are the preferred communication technique to use across application boundaries, including platform, deployment, and trust boundaries. If you are building a distributed application, consider using a service-oriented approach. Although object orientation provides a pure view of what a system should look like and is effective for producing logical models, an object-based approach can fail to consider real-world factors, such as physical distribution, trust boundaries, and network communication, as well as nonfunctional requirements, such as performance and security.

Object orientation	Service orientation
Assumes a homogeneous platform and execution environment.	Assumes a heterogeneous platform and execution environment.
Shares types, not schemas.	Shares schemas, not types.
Assumes cheap, transparent communication.	Assumes variable cost, explicit communication.
Objects are linked: object identity and lifetime are maintained by the infrastructure.	Services are autonomous: security and failure isolation are a necessity.

Typically requires synchronized deployment of both client and server.	Allows continuous, separate deployment of client and server.
Is easy to conceptualize and thus provides a natural model to follow.	Builds on ideas from component software and distributed objects. Dominant theme is to manage/reduce sharing between services.
Provides no explicit guidelines for state management and ownership.	Owns and maintains state or uses the reference state.
Assumes a predictable sequence, timeframe, and outcome of invocations.	Assumes message-oriented, potentially asynchronous, and long-running communications.
Goal is to transparently use functions and types remotely.	Goal is to provide inter-service isolation and wire interoperability based on standards.

Application Boundaries

Common application boundaries include platform, deployment, trust, and evolution boundaries. (*Evolution* refers to whether or not you develop and upgrade applications together.) When evaluating architecture and design decisions that affect your application boundaries, consider the following:

- Objects and remote procedure calls (RPC) are appropriate within boundaries.
- Services are appropriate across and within boundaries.

Enterprise SOA vs. Application SOA

Because SOA is an architectural style, it can be helpful to think of SOA in terms of two different scopes. For the purposes of this guide, we factor SOA in terms of individual application scenarios and larger enterprise scenarios.

Enterprise SOA

At the enterprise level, you consider SOA from the standpoint of your enterprise architecture. This is where your enterprise architects come into play. The following are key SOA considerations at the enterprise level:

- How can you compose application services to create a business process?
- What is your portfolio of capabilities that are candidates for services?
- How can you push common application features to a common services infrastructure?
- How can you provide common services across your portfolio of applications?
- How can you connect your heterogeneous systems through common services?
- How can you provide a common security infrastructure for your services?

Application SOA

At the application level, you consider SOA from the standpoint of your application's architecture and architectural style. This is where your application architects come into play. The following are key SOA considerations at the application level:

- How can you design, create, and consume services within your application?
- How can you leverage service-oriented approaches within your application's architecture?
- How can you design for effective message-oriented communication?

This guides focuses on application-level SOA.

SOA Security in Practice

Effective SOA security in practice includes the following measures:

- Coordinating people, process, and technology.
- Integrating and leveraging various levels of standards (general security standards, XML security standards, Web services security standards).
- Integrating and leveraging various user stores and role stores.
- Making trade-offs between user experience, technical, and business perspectives.

WS-Security Standards / Web Services Security Concepts

The WS-* architecture is a set of standards-based protocols designed to secure Web service communication. The WS-* security standards include:

- **WS-Policy**. WS-Policy allows Web services to define policy requirements for endpoints. These requirements include privacy rules, encryption rules, and security tokens.
- **WS-Security**. WS-Security allows Web services to apply security to Simple Object Access Protocol (SOAP) messages through encryption and integrity checks on all or part of the message.
- **WS-Trust**. WS-Trust allows Web services to use security tokens to establish trust in a brokered security environment.
- **WS-SecureConversation**. WS-SecureConversation builds on top of WS-Policy, WS-Security, and WS-Trust to enable secure communications between client and service.
- **WS-ReliableMessaging**. WS-ReliableMessaging allows Web services and clients to trust that when a message is sent, it will be delivered to the intended party.
- **WS-AtomicTransactions**. WS-AtomicTransactions allows transaction-based Web services in which transactions can be rolled back in the event of a failure.

How Do You Build Secure Services?

The keys to building secure services include:

- **Identify your security objectives**. This includes identifying your security requirements.

- **Know your threats**. Know which threats are relevant for your particular scenarios and context. Threat modeling is an effective technique for helping you identify relevant threats and vulnerabilities. Your objectives will help you prioritize your threats and vulnerabilities. Using the threat model, developers address vulnerabilities, and testers verify that the developers closed the issues.
- **Apply proven principles, patterns, and practices**. Principles, patterns, and practices are a good starting point for building secure services. By using proven principles, patterns, and practices, you can eliminate classes of security problems. You can also leverage lessons learned. Patterns are effectively reusable solutions and typically encapsulate underlying principles. While principles, patterns, and practices are a good starting point, you should never blindly adopt them — you need to evaluate whether they make sense for your scenario.
- **Apply effective security engineering throughout the application life cycle**. You should consider security throughout your application life cycle. You should start with security objectives. Threat modeling will help you shape your design and make key trade-offs. Security design, code, and deployment inspections, along with testing, will improve your overall security posture.

Additional Resources

- For more information on security engineering, see “patterns & practices Security Engineering Explained” at http://msdn.microsoft.com/en-us/library/ms998382.aspx .
- For more information on threat modeling, see “Threat Modeling Web Applications” at http://msdn.microsoft.com/en-us/library/ms978516.aspx.

patterns & practices Security Engineering

The Microsoft patterns & practices Security Engineering approach includes specific security-related activities that help you meet your application security objectives.

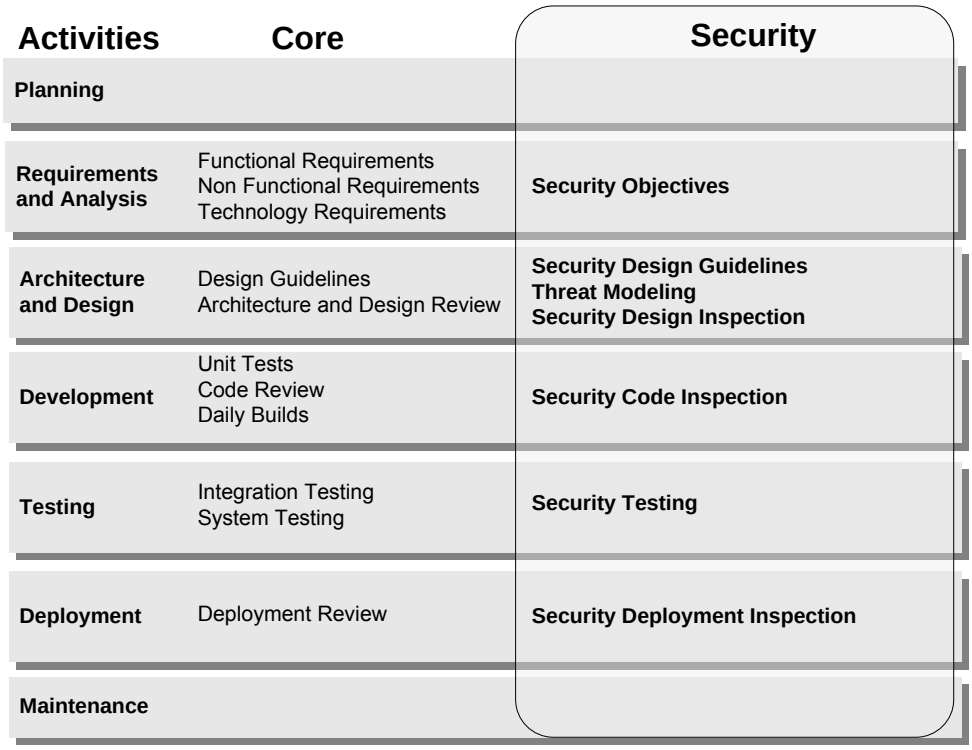


Figure 1 - Key Security Engineering Activities

Summary of Key Security Engineering Activities

This patterns & practices Security Engineering approach extends these proven core activities to create security-specific activities. These activities include:

- **Security objectives**. Setting objectives helps you scope and prioritize your work by setting boundaries and constraints. Setting security objectives helps you identify where to start, how to proceed, and when you are done.
- **Threat modeling**. Threat modeling is an engineering technique that can help you identify threats, attacks, vulnerabilities, and countermeasures that could affect your application. You can use threat modeling to shape your application's design, meet your company's security objectives, and reduce risk.
- **Security design guidelines**. Creating design guidelines is a common practice at the start of an application project to guide development and share knowledge across the team. Effective design guidelines for security organize security principles, practices, and patterns by actionable categories.
- **Security design inspection**. Security design inspections are an effective way to identify problems in your application design. By using pattern-based categories and a question-driven approach, you simplify evaluating your design against root-cause security issues.
- **Security code inspection**. Many security defects are found during code reviews. Analyzing code for security defects includes knowing what to look for and how to look for it. Security code inspections optimize inspecting code for common security issues.

- **Security testing**. Use a risk-based approach and use the output from the threat-modeling activity to help establish the scope of your testing activities and define your test plans.
- **Security deployment inspection**. When you deploy your application during your build process or staging process, you have an opportunity to evaluate run-time characteristics of your application in the context of your infrastructure. Deployment reviews for security focus on evaluating your security design and configuration of your application, host, and network.

For more information on security engineering, see "patterns & practices Security Engineering Explained" at http://msdn.microsoft.com/en-us/library/ms998382.aspx#

Web Services Security Principles

Recommendations made throughout this guide are based on a core set of security principles. These principles have proven effective across many different technologies and scenarios, including Web services in SOA. Use the following list to apply a principle-based approach to Web service security when building your WCF application.

Principle	Concepts
Apply defense in depth	Use multiple gatekeepers to keep attackers at bay. Defense in depth means you do not rely on a single layer of security, or you consider that one of your layers may be bypassed or compromised.
Check at the gate	Authenticate and authorize callers early — at the first gate.
Compartmentalize	Isolate and contain a problem. Apply the principle of separation of concerns. If an attacker takes over your application, what resources can he or she access? Can an attacker access network resources? How are you restricting potential damage? Firewalls, least-privileged accounts, and least-privileged code are examples of compartmentalizing.
Create secure defaults	Is the default account set up with least privilege? Is the default account disabled by default and then explicitly enabled when required? Does the configuration use a password in plaintext? When an error occurs, does sensitive information leak back to the client, to potentially be used against the system?
Do not trust user input	Keep user input out of the control path. Your application's user input is the attacker's primary weapon when targeting your application. Assume that all input is malicious until proven otherwise, and apply a defense-in-depth strategy to input validation, taking particular precautions to make sure that input is validated whenever a trust boundary in your application is crossed. You need to validate input at both entry points and exit points in your application.

Establish trust boundaries	Trust boundaries indicate where trust levels change. You can think of trust from the perspective of confidentiality and integrity. For example, a change in access control levels in your application, where a specific role or privilege level is required to access a resource or operation, would be a change in trust level. Another example would be at an entry point in your application where you might not fully trust the data passed to the entry point. Identify trust boundaries from a data flow perspective. For each subsystem, consider whether the upstream data flow or user input is trusted, and if it is not, consider how the data flow and input can be authenticated and authorized. Knowing which entry points exist between trust boundaries allows you to focus your threat identification on these key entry points. For example, you are likely to have to perform more validation on data passed through an entry point at a trust boundary.
Fail securely	If an application fails, do not leave sensitive data accessible. Return friendly errors to end users that do not expose internal system details. Do not include details that may help attackers exploit vulnerabilities in your application.
Reduce your attack surface	If you do not use it, remove it or disable it. Reduce the surface area of attack by disabling or removing unused services, protocols, and functionality. Does your server need all those services and ports? Does your application need all those features?
Secure the weakest link	Is there vulnerability at the network layer that an attacker can exploit? What about the host? Is your application secure? Any weak link in the chain is an opportunity for breached security.
Use least privilege	By running processes using accounts with minimal privileges and access rights, you significantly reduce the capabilities of an attacker if the attacker manages to compromise security and run code.

Web Services Security Frame

The following key security concepts provide a frame for thinking about security when designing and architecting services. This helps you turn core security features such as authentication, authorization, auditing, confidentiality, integrity, and availability into action.

Category	Description
Auditing and Logging	Auditing and logging refers to how security-related events are recorded, monitored, and audited.
Authentication	Authentication is the process where an entity proves the identity of another entity, typically through credentials, such as a username and password.

Authorization	Authorization is how your service provides access controls for resources and operations.
Configuration Management	Configuration management refers to how your service handles database connections, administration, and other configuration settings.
Exception Management	Exception management refers to how you handle exceptions within your application, including fault contracts.
Impersonation/Delegation	Impersonation and delegation refers to how your service impersonates users and passes identity information downstream for authorization purposes.
Message Encryption	Message encryption refers to protecting a message by converting the contents to cipher-text using cryptographic methods.
Message Replay Detection	Message replay detection refers to identifying and rejecting messages that are resubmitted.
Message Signing	Message signing refers to signing a message with a digital signature using cryptographic methods, to confirm the source of the message and detect if the contents have been tampered with (i.e., authentication and integrity of the message).
Message Validation	Message validation refers to how you verify the message payload against a schema, as well as message size, content, and character sets. This includes how your service filters, scrubs, and rejects input and output before additional processing. Input and output includes input from clients consuming the service as well as file-system input, in addition to input from network resources, such as databases. Output typically includes the return values from your service or disk/database writes, among others.
Sensitive Data	Sensitive data is user and application data whose integrity and confidentiality need to be protected. This includes how you protect sensitive data from being stolen from memory, from configuration files, or when transmitted over the network.
Session Management	A session refers to a series of related interactions between a client and your service.

Using the Web Services Security Frame

The Web Services Security Frame serves as a foundation for the rest of this guide. Guidelines, checklists, and other guidance are all organized around the categories represented in this frame. You can use this frame to help wrap your mind around WCF security and better organize the key decisions you need to make when considering security for your application. Through practice and experience, we have learned that the frame is most useful when combined with a question-driven approach to security. This approach will help you transition from security understanding to actionable steps you can take to improve the security stance of your application.

Category	Key questions
Auditing and Logging	• What events are important for the security of your application? • In the event of an attack, what trail of evidence would you want left behind for your investigation? • What user management or sensitive business operations do you want to track?
Authentication	• What credentials will your users present to your service? • From what types of clients and locations (Internet versus intranet) will they be calling? • How do you want to store user account information? • Do you want to map authentication to pre-existing Windows accounts in your domain?
Authorization	• What roles will be defined for your service? • What operations in your service should require explicit authorization? • Do you want to authorize the original caller in your service, before your service, or in the business layers called by your service? • Do you need to use the original caller to access resources on the back end? • Where do you want to store role information? • Do you already have roles defined, such as Windows groups, that you want your service to interact with?
Configuration Management	• Under what security context does your application run? • Which databases does it connect to and under what security context? • How is your application administered? • What settings are sensitive and should be secured?
Exception Management	• When a method call in your application fails, what does your application do? • How much do you reveal? • Do you return friendly error information to end users? • Do you pass valuable exception information back to the caller? • Does your application fail gracefully?
Impersonation/Delegation	• What tiers and layers of your application need access to the original caller's identity and credentials? • Do you need to flow the original caller to back-end resources? • Do you need to authorize the original caller at the service level or in a downstream component? • Do you need to access the database using the original caller's

	security context? • Which operations in your service will need to use impersonation to flow the original caller's identity?
Message Encryption	• Is there sensitive data transmitted in your messages that needs to be protected from exposure to an attacker?
Message Replay Detection	• How do you protect a service from an attacker who replays an intercepted message?
Message Signing	• Is it important that the message source can be verified and that the contents have not been modified?
Message Validation	• How will you validate incoming SOAP messages on your service? • How will you validate input parameters on your service? • How will you validate information that is returned to your client? • How will you validate data that comes from other sources such as your database or the file system? • How will you make your outbound data safe?
Sensitive Data	• How does your application handle sensitive data? (Sensitive data refers to any data that must be protected either in memory, over the network, or in persistent stores, and how your application handles that data.) • How are you keeping secrets (confidentiality)? • How are you tamper-proofing your data or libraries (integrity)? • How are you providing seeds for random values that must be cryptographically strong? (Cryptography refers to how your application enforces confidentiality and integrity.)
Session Management	• How does your application handle and protect client sessions?

Web Services Security Patterns

The following Web services security patterns from the Microsoft patterns & practices *Web Services Security* guide (http://msdn.microsoft.com/en-us/library/aa480545.aspx) are helpful for addressing various security concerns, such as authentication, authorization, etc.

- Brokered Authentication
- Brokered Authentication: Kerberos
- Brokered Authentication: X.509 PKI
- Brokered Authentication: STS
- Data Confidentiality
- Data Origin Authentication
- Direct Authentication
- Exception Shielding

- Message Replay Detection
- Message Validator
- Perimeter Service Router
- Protocol Transition with Constrained Delegation
- Trusted Subsystem

Web Services Security Patterns Organized by the Web Services Security Frame

The following table summarizes the Web Services Security patterns organized by the Web Services Security Frame.

Pattern	Description	Reference
Authentication		
Direct Authentication	The Web service acts as an authentication service to validate credentials from the client. The credentials, which include proof-of-possession that is based on shared secrets, are verified against an identity store.	http://msdn.microsoft.com/en-us/library/aa480566.aspx
Brokered Authentication	The Web service validates the credentials presented by the client, without the need for a direct relationship between the two parties. An authentication broker that both parties trust independently issues a security token to the client. The client can then present credentials, including the security token, to the Web service.	http://msdn2.microsoft.com/en-us/library/aa480560.aspx
Brokered Authentication: Kerberos	Use the Kerberos protocol to broker authentication between clients and Web services.	http://msdn2.microsoft.com/en-us/library/aa480562.aspx
Brokered Authentication: X.509 PKI	Use brokered authentication with X.509 certificates issued by a certificate authority (CA) in a public key infrastructure (PKI) to verify the credentials presented by the requesting application.	http://msdn2.microsoft.com/en-us/library/aa480565.aspx
Brokered Authentication: STS	Use brokered authentication with a security token issued by a Security Token Service (STS). The STS is trusted by both the client and the Web service to provide interoperable security tokens.	http://msdn2.microsoft.com/en-us/library/aa480563.aspx
Authorization		

Protocol Transition with Constrained Delegation	Use the Kerberos protocol extensions in Microsoft Windows Server®. The extensions require the user ID but not the password. You still need to establish trust between the client application and the Web service; however, the application is not required to store or send passwords.	http://msdn.microsoft.com/en-us/library/aa480585.aspx
Trusted Subsystem	The Web service acts as a trusted subsystem to access additional resources. It uses its own credentials instead of the user's credentials to access the resource.	http://msdn2.microsoft.com/en-us/library/aa480587.aspx
Exception Management		
Exception Shielding	Sanitize unsafe exceptions by replacing them with exceptions that are safe by design. Return only those exceptions to the client that have been sanitized, or exceptions that are safe by design. Exceptions that are safe by design do not contain sensitive information in the exception message, and they do not contain a detailed stack trace, either of which might reveal sensitive information about the Web service's inner workings.	http://msdn2.microsoft.com/en-us/library/aa480591.aspx
Message Encryption		
Data Confidentiality	Use encryption to protect sensitive data that is contained in a message. Unencrypted data, which is known as *plaintext*, is converted to encrypted data, which is known as *ciphertext*. Data is encrypted with an algorithm and a cryptographic key. Ciphertext is then converted back to plaintext at its destination.	http://msdn.microsoft.com/en-us/library/aa480570.aspx
Message Replay Detection		
Message Replay Detection	Cache an identifier for incoming messages, and use message replay detection to identify and reject messages that match an entry in the replay detection cache.	http://msdn2.microsoft.com/en-us/library/aa480598.aspx
Message Signing		
Data Origin Authentication	Use data origin authentication, which enables the recipient to verify that messages have not been tampered with in	http://msdn2.microsoft.com/en-us/library/aa480571.aspx

	transit (data integrity) and that they originate from the expected sender (authenticity).	
Message Validation		
Message Validator	The message validation logic enforces a well-defined policy that specifies which parts of a request message are required for the service to successfully process it. It validates the XML message payloads against an XML schema (XSD) to ensure that they are well-formed and consistent with what the service expects to process. The validation logic also measures the messages against certain criteria by examining the message size, the message content, and the character sets that are used. Any message that does not meet the criteria is rejected.	http://msdn2.microsoft.com/en-us/library/aa480600.aspx
Deployment		
Perimeter Service Router	Design a Web service intermediary that acts as a perimeter service router. The perimeter service router provides an external interface on the perimeter network for internal Web services. It accepts messages from external applications and routes them to the appropriate Web service on the private network.	http://msdn2.microsoft.com/en-us/library/aa480606.aspx

Summary

The foundations of WCF security include authentication, authorization, auditing, confidentiality, integrity, and availability. When you think about security in your service, you should first understand the distinctions between threats, attacks, vulnerabilities, and countermeasures. To build secure services, you will identify your security objectives; identify your threats and vulnerabilities; apply principles, patterns, and practices; and use security engineering techniques throughout your application life cycle. By using the Web Services Security Frame, you can better organize and use your security knowledge.

Additional Resources

For more information, see the following resources:

- For more information on applying security throughout the life cycle, see “patterns & practices Security Engineering Explained” at http://msdn.microsoft.com/en-us/library/ms998382.aspx .
- For more information on how to perform effective threat modeling, see “patterns & practices Threat Modeling Web Applications” at http://msdn.microsoft.com/en-us/library/ms978516.aspx .
- For more information on Web Services Security patterns, see “patterns & practices Web Services Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0” at http://msdn.microsoft.com/en-us/library/aa480545.aspx .

Chapter 2 – Threats and Countermeasures for Web Services

Objectives

- Understand the difference between a threat, an attack, a vulnerability, and a countermeasure
- Understand common vulnerabilities in Web services.
- Understand how to implement effective countermeasures for dealing with common Web services threats and attacks.

Overview

It is important to think like an attacker when designing and implementing your service. Putting yourself in the attacker's mindset will make you more effective at designing mitigations for vulnerabilities and coding defensively.

Threats, Attacks, Vulnerabilities, and Countermeasures

This chapter analyzes security for Web services from the perspectives of threats, vulnerabilities, attacks, and countermeasures. These terms are defined as follows:

- **Asset**. A resource of value such as the data in a database, data on the file system, or a system resource.
- **Threat**. A potential occurrence – malicious or otherwise – that can harm an asset.
- **Vulnerability**. A weakness that makes a threat possible.
- **Attack.** An action taken to exploit vulnerability and realize a threat.
- **Countermeasure**. A safeguard that addresses a threat and mitigates risk.

Web Services Security Frame

The following key security concepts provide a frame for thinking about security when designing and architecting services. Understanding these concepts helps you put core security features such as authentication, authorization, auditing, confidentiality, integrity, and availability in action.

Category	Description
Auditing and logging	*Auditing and logging* refers to how security-related events are recorded, monitored, and audited.
Authentication	*Authentication* is the process in which an entity proves the identity of another entity, typically through credentials, such as a username and password.
Authorization	*Authorization* is the means by which your service provides access controls for resources and operations.

Configuration Management	*Configuration management* refers to how your service handles database connections, administration, and other configuration settings.
Exception management	*Exception management* refers to how you handle exceptions within your application, including fault contracts.
Impersonation/delegation	*Impersonation and delegation* refers to how your service impersonates users and passes identity information downstream for authorization purposes.
Message encryption	*Message encryption* refers to protecting a message by converting the contents to cipher-text using cryptographic methods.
Message replay detection	*Message replay detection* refers to identifying and rejecting messages that are resubmitted.
Message signing	*Message signing* refers to signing a message with a digital signature using cryptographic methods, to confirm the source of the message and detect if the contents have been tampered with (i.e., authentication and integrity of the message).
Message validation	*Message validation* refers to how you verify the message payload against a schema, as well as message size, content, and character sets. This includes how your service filters, scrubs, and rejects input and output before additional processing. Input and output includes input from clients consuming the service as well as file-system input, in addition to input from network resources, such as databases. Output typically includes the return values from your service or disk/database writes among others.
Sensitive data	*Sensitive data* refers to user and application data whose integrity and confidentiality you need to protect. You need to protect sensitive data from being stolen from memory or configuration files, or when it is transmitted over the network.
Session management	A *session* refers to a series of related interactions between a client and your service.

Auditing and Logging

Auditing and logging is used to monitor and record important activity, such as transactions or user management events, on both the client and the service. Logged information should be recorded and stored to enable efficient auditing of events in the case of an attack or a suspected attack.

Threats and attacks include:

- **Repudiation**. An attacker denies performing an operation, exploits an application without trace, or covers his or her tracks.

- **Denial of service (DoS)**. An attacker overwhelms logs with excessive entries or very large log entries.
- **Disclosure of confidential information.** An attacker gathers sensitive information from log files.

Vulnerabilities include:

- Failing to audit failed logons.
- Failing to secure log files.
- Storing sensitive information in log files.
- Failing to audit across application tiers.
- Failure to throttle log files.

Countermeasures include:

- Identify malicious behavior.
- Know your baseline (know what good traffic looks like).
- Use application instrumentation to expose behavior that can be monitored.
- Throttle logging.
- Strip sensitive data before logging.

Authentication

Authentication is the mechanism by which your clients can establish their identity with your service, using a set of credentials that prove that identity. A username is an example of an identity, while a password is an example of a credential.

Threats and attacks include:

- **Network eavesdropping**. An attacker steals identity and/or credentials off the network by reading network traffic not intended for them.
- **Brute force attacks**. An attacker guesses identity and/or credentials through the use of brute force.
- **Dictionary attacks.** An attacker guesses identity and/or credentials through the use of common terms in a dictionary designed for that purpose.
- **Cookie replay attacks.** An attacker gains access to an authenticated session through the reuse of a stolen cookie containing session information.
- **Credential theft.** An attacker gains access to credentials through data theft; for instance, phishing or social engineering.

Vulnerabilities:

- Using weak passwords.
- Storing clear text credentials in configuration files.
- Passing clear text credentials over the network.
- Permitting prolonged session lifetime.

- Mixing personalization with authentication.
- Using weak authentication mechanisms (e.g., using basic authentication over an untrusted network).

Countermeasures include:

- Use strong password policies.
- Do not store credentials in an insecure manner.
- Use authentication mechanisms that do not require clear text credentials to be passed over the network.
- Encrypt communication channels to secure authentication tokens.
- Use Secure HTTP (HTTPS) only with forms authentication cookies.
- Separate anonymous from authenticated pages.
- Using cryptographic random number generators to generate session IDs.

Authorization

Authorization is the mechanism by which you control the operations and resources an authenticated client can access. Authorization controls may be enforced at the class level, the method level, or in fine-grained business logic if needed.

Threats and attacks include:

- **Elevation of privilege.** An attacker enters a system as a lower-level user, but is able to obtain higher-level access.
- **Disclosure of confidential data.** An attacker accesses confidential information because of authorization failure on a resource or operation.
- **Data tampering.** An attacker modifies sensitive data because of authorization failure on a resource or operation.
- **Luring attacks**. An attacker lures a higher-privileged user into taking an action on their behalf. This is not an authorization failure but rather a failure of the system to properly inform the user.
- **Token stealing**. An attacker steals the credentials or token of another user in order to gain authorization to resources or operations they would not otherwise be able to access.

Vulnerabilities include:

- Relying on a single gatekeeper (e.g., relying on client-side validation only).
- Failing to lock down system resources against application identities.
- Failing to limit database access to specified stored procedures.
- Using inadequate separation of privileges.
- Connection pooling.
- Permitting overprivileged accounts.

Countermeasures include:

- Use least-privileged accounts.
- Tie authentication to authorization on the same tier.
- Consider granularity of access.
- Enforce separation of privileges.
- Use multiple gatekeepers.
- Secure system resources against system identities.

Configuration Management

Security settings, authentication, authorization, logging, and other parameters can usually be set in configuration files. Improper configuration can lead to security vulnerabilities, as can a lack of protection on the files themselves, which can lead to improper modification or theft of connection strings and other sensitive information.

Threats and attacks include:

- **Unauthorized access to configuration stores.** An attacker gains access to configuration files and is able to modify binding settings, etc.
- **Retrieval of clear text configuration secrets.** An attacker gains access to configuration files and is able to retrieve sensitive information such as database connection strings.

Vulnerabilities include:

- Using insecure custom administration interfaces.
- Failing to secure configuration files on the server.
- Storing sensitive information in the clear text.
- Having too many administrators.
- Using overprivileged process accounts and service accounts.

Countermeasures include:

- Use access control lists (ACLs).
- Encrypt sensitive sections of configuration files.
- Use secure settings for various operations of Web services using configuration files.

Exception Management

Exception management is the means by which you expose and consume exception information within your service and send it back to your clients. In most cases, exceptions should be shielded from the client entirely and handled with a sanitized error message. In addition, fault contracts should be negotiated beforehand so that the client and the Web service agree on a course of action when a failure occurs.

Threats and attacks include:

- **Information disclosure.** Sensitive system or application details are revealed through exception information.
- **Denial of service**. An attacker uses error conditions to stop your service or place it in an unrecoverable error state.
- **Elevation of privilege.** Your service encounters an error and fails to an insecure state; for instance, failing to revert impersonation.

Vulnerabilities include:

- Failure to use structured exception handling (try/catch).
- Revealing too much information to the client.
- Failure to specify fault contracts with the client.
- Failure to use a global exception handler.

Countermeasures include:

- Use structured exception handling (by using try/catch blocks).
- Catch and wrap exceptions only if the operation adds value/information.
- Do not reveal sensitive system or application information.
- Implement a global exception handler.
- Do not log private data such as passwords.

Impersonation/Delegation

Impersonation and delegation are techniques used to flow the original caller to back-end resources. *Impersonation* is used to access resources on the same machine where the service code is running. *Delegation* is used to access network resources on other machines.

Threats and attacks include:

- **Elevation of privilege.** An attacker is able to run in the context of a higher-privileged user.
- **Disclosure of confidential information**. An attacker gains access to data that should only be available to another user.

Vulnerabilities include:

- Failure to revert to a lower privilege after using impersonation.
- Improper use of global impersonation across the entire service.

Countermeasures include:

- Use **Using** statement to automatically revert impersonation.
- Granularly impersonate only those operations that need it.

Message Encryption

Message encryption is used to protect sensitive data in-transport over the network. Encryption does not protect the integrity of the data, but only its confidentiality. Message encryption can be provided by either message security or transport security. Message security encrypts each message individually, while transport security encrypts the entire communication channel (e.g., with SSL).

Threats and attacks include:

- **Failure to encrypt messages**. An attacker is able to read message content off the network because it is not encrypted.
- **Theft of encryption keys.** An attacker is able to decrypt sensitive data because he or she has the keys.
- **Man-in-the-middle attack.** An attacker can read and then modify messages between the client and the service.

Vulnerabilities include:

- Not encrypting messages.
- Using custom cryptography.
- Distributing keys insecurely.
- Managing or storing keys insecurely.

Countermeasures include:

- Use message security or transport security to encrypt your messages.
- Use proven platform-provided cryptography.
- Periodically change your keys.

Message Replay Detection

Message replay detection is a feature that allows your code to detect some instances in which an attacker is trying to replay messages in order to steal a session from one of your clients.

Threats and attacks include:

- **Session replay.** An attacker steals messages off the network and replays them in order to steal a user's session.

Vulnerabilities include:

- Failure to use a mechanism to detect message replays.

Countermeasures include:

- Use any platform-provided replay detection features.

- Consider creating custom code if the platform does not provide a detection mechanism.

Message Signing

Message signing is used to protect the integrity of messages in transit over the network and to provide proof of the original sender. Signing does not protect the confidentiality of the data, but only its integrity and confidence in the original sender. Message signing can be provided by either message security or transport security. Message security signs each message individually, while transport security protects the entire communication channel (e.g., with SSL).

Threats and attacks include:

- **Data tampering**. An attacker modifies the data in a message in order to attack the client or the service.

Vulnerabilities include:

- Not using either message or transport security.

Countermeasures include:

- Turn on message or transport security.

Message Validation

Message validation is used to protect your service from malformed messages and message parameters. Message schemas can be used to validate incoming messages, while custom validators can be used to validate parameter data before your service consumes it.

Threats and attacks include:

- **Canonicalization attacks.** Canonicalization attacks can occur anytime validation is performed on a different form of the input than that which is used for later processing. For instance, a validation check may be performed on an encoded string, which is later decoded and used as a file path or URL.
- **Cross-site scripting.** Cross-site scripting can occur if you fail to encode user input before echoing back to a client that will render it as HTML.
- **SQL injection.** Failure to validate input can result in SQL injection if the input is used to construct a SQL statement, or if it will modify the construction of a SQL statement in some way.
- **XPath injection.** XPath injection can result if the input sent to the Web service is used to influence or construct an XPath statement. The input can also introduce unintended results if the XPath statement is used by the Web service as part of some larger operation, such as applying an XQuery or an XSLT transformation to an XML document.
- **XML bomb.** XML bomb attacks occur when specific, small XML messages are parsed by a service resulting in data that feeds on itself and grows exponentially. An attacker sends an

XML bomb with the intent of overwhelming a Web service's XML parser, thus resulting in a denial of service (DoS) attack.

Vulnerabilities include:

- Using non-validated input used to generate SQL queries.
- Relying only on client-side validation.
- Using input file names, URLs, or usernames for security decisions.
- Using application-only filters for malicious input.
- Looking for known bad patterns of input.
- Trusting data read from databases, file shares, and other network resources.
- Failing to validate input from all sources including cookies, Simple Object Access Protocol (SOAP) headers, SOAP parameters, databases, and network resources.

Countermeasures include:

- Do not trust client input.
- Validate input: length, range, format, and type.
- Validate XML streams.
- Constrain, reject, and sanitize input.
- Encode output.
- Restrict the size, length, and depth of parsed XML messages.

Sensitive Data

Sensitive data refers to any confidential information that your service processes or transmits. Protection of sensitive data includes protecting the information over the network, in configuration files, in local memory or file storage, or in databases and log files. Sensitive information includes user identity and credentials as well as any personally identifiable information such as social security number.

A more complete definition of sensitive data is:

- Information that either contains personally identifiable information or can be used to derive personally identifiable information that should not be shared with users.
- Information that a user provides that they would not want shared with other users of the application.
- Information that comes from an external trusted source that is not designed to be shared with users.

Threats and attacks include:

- **Memory dumping.** An attacker is able to read sensitive data out of memory or from local files.
- **Network eavesdropping.** An attacker listens to and intercepts unencrypted sensitive data off the network.

- **Configuration file sniffing**. An attacker steals sensitive information, such as connection strings, out of configuration files.

Vulnerabilities include:

- Storing secrets when you do not need to.
- Storing secrets in code.
- Storing secrets in clear text in files, registry, or configuration.
- Passing sensitive data in clear text over networks.

Countermeasures include:

- Do not store secrets in software.
- Encrypt sensitive data over the network.
- Secure the channel.
- Encrypt sensitive data in configuration files.

Session Management

Sessions are the means by which an application maintains stateful communication with a client over time. This is usually supported through the use of a session ID, token, or cookie. If a session is supported in such a way that credentials are not required for every interaction, an attacker could potentially steal the session and act on the original user's behalf.

Threats and attacks include:

- **Session hijacking.** An attacker steals the session ID of another user in order to gain access to resources or operations they would not otherwise be able to access.
- **Session replay.** An attacker steals messages off the network and replays them in order to steal a user's session.
- **Man-in-the-middle attack.** An attacker can read and then modify messages between the client and the service.
- **Inability to log out successfully.** An application leaves a communication channel open rather than completely closing the connection and destroying any server objects in memory relating to the session.
- **Cross-site request forgery.** Cross-site request forgery (CSRF) is where an attacker tricks a user into performing an action on a site where the user actually has a legitimate authorized account.
- **Session fixation.** An attacker uses CSRF to set another person's session identifier and thus hijack the session after the attacker tricks a user into initiating it**.**
- **Load balancing and session affinity**. When sessions are transferred from one server to balance traffic among the various servers, an attacker can hijack the session during the handoff.

Vulnerabilities include:

- Passing session IDs over unencrypted channels.
- Permitting prolonged session lifetime.
- Having insecure session state stores.
- Placing session identifiers in query strings.

Countermeasures include:

- Partition the site by anonymous, identified, and authenticated users.
- Reduce session timeouts.
- Avoid storing sensitive data in session stores.
- Secure the channel to the session store.
- Authenticate and authorize access to the session store.

Threats and Attacks Explained

The following explanations briefly describe some of the threats and attacks mentioned above:

- **Brute force attacks.** Attacks that use the raw computer processing power to try different permutations of any variable that could expose a security hole. For example, if an attacker knew that access required an 8-character username and a 10-character password, the attacker could iterate through every possible combination (256 multiplied by itself 18 times) in order to attempt to gain access to a system. No intelligence is used to shape or filter likely combinations.
- **Buffer overflows**. The maximum size of a given variable (string or otherwise) is exceeded, forcing unintended program processing. In this case, the attacker uses this behavior to cause insertion and execution of code in such a way that the attacker gains control of the program in which the buffer overflow occurs. Depending on the program's privileges, the seriousness of the security breach will vary.
- **Canonicalization attacks.** There are multiple ways to access the same object and an attacker uses a method to bypass any security measures instituted on the primary intended methods of access. Often, the unintended methods of access can be less secure deprecated methods.
- **Cookie manipulation. Throu**gh various methods, an attacker will alter the cookies stored in the browser. Attackers will then use the cookie to fraudulently authenticate themselves to a service or Web site.
- **Cookie replay attacks**. Reusing a previously valid cookie to deceive the server into believing that a previously authenticated session is still in progress and valid.
- **Credential theft.** Stealing the verification part of an authentication pair (identity + credentials = authentication). Passwords are a common credential.
- **Cross-site scripting**. An attacker is able to inject executable code (script) into a stream of data that will be rendered in a browser. The code will be executed in the context of the user's current session and will gain privileges to the site and information that it would not otherwise have.

- **Connection pooling.** The practice of creating and then reusing a connection resource as a performance optimization. In a security context, this can result in either the client or server using a connection previously used by a highly privileged user being used for a lower-privileged user or purpose. This can potentially expose vulnerabilities if the connection is not reauthorized when used by a new identity.
- **Data tampering**. An attacker violates the integrity of data by modifying it in local memory, in a data-store, or on the network. Modification of this data could provide the attacker with access to a service through a number of the different methods listed in this document.
- **Denial of service**. Denial of service (DoS) is the process of making a system or application unavailable. For example, a DoS attack might be accomplished by bombarding a server with requests to consume all available system resources, or by passing the server malformed input data that can crash an application process.
- **Dictionary attack**. Use of a list of likely access methods (usernames, passwords, coding methods) to try and gain access to a system. This approach is more focused and intelligent than the "brute force" attack method, so as to increase the likelihood of success in a shorter amount of time.
- **Disclosure of sensitive/confidential data.** Sensitive data is exposed in some unintended way to users who do not have the proper privileges to see it. This can often be done through parameterized error messages, where an attacker will force an error and the program will pass sensitive information up through the layers of the program without filtering it. This can be personally identifiable information (PII) or system data.
- **Elevation of privilege**. EA user with limited privileges assumes the identity of a privileged user to gain privileged access to an application. For example, an attacker with limited privileges might elevate his or her privilege level to compromise and take control of a highly privileged and trusted process or account.
- **Encryption.** The process of taking sensitive data and changing it in such a way that it is unrecognizable to anyone but those who know how to decode it. Different encryption methods have different strengths based on how easy it is for an attacker to obtain the original information through whatever methods are available.
- **Information disclosure**. Unwanted exposure of private data. For example, a user views the contents of a table or file that he or she is not authorized to open, or monitors data passed in plaintext over a network. Some examples of information disclosure vulnerabilities include the use of hidden form fields, comments embedded in Web pages that contain database connection strings and connection details, and weak exception handling that can lead to internal system-level details being revealed to the client. Any of this information can be very useful to the attacker.
- **Luring attacks**. An attacker lures a higher-privileged user into taking an action on his or her behalf. This is not an authorization failure but rather a failure of the system to properly inform the user.
- **Man-in-the-middle attacks**. A person intercepts both the client and server communications and then acts as an intermediary between the two without each ever knowing. This gives the "middle man" the ability to read and potentially modify messages from either party in order to implement another type of attack listed here.

- **Network eavesdropping**. Listening to network packets and reassembling the messages being sent back and forth between one or more parties on the network. While not an attack itself, network eavesdropping can easily intercept information for use in specific attacks listed in this document.
- **Password cracking**. If the attacker cannot establish an anonymous connection with the server, he or she will try to establish an authenticated connection. For this, the attacker must know a valid username and password combination. If you use default account names, you are giving the attacker a head start. Then the attacker only has to crack the account's password. The use of blank or weak passwords makes the attacker's job even easier.
- **Repudiation**. The ability of users (legitimate or otherwise) to deny that they performed specific actions or transactions. Without adequate auditing, repudiation attacks are difficult to prove.
- **Session hijacking**. Also known as man-in-the-middle attacks, session hijacking deceives a server or a client into accepting the upstream host as the actual legitimate host. Instead, the upstream host is an attacker's host that is manipulating the network so the attacker's host appears to be the desired destination.
- **Session replay**. An attacker steals messages off of the network and replays them in order to steal a user's session.
- **Session fixation.** An attacker sets (fixates) another person's session identifier artificially. The attacker must know that a particular Web service accepts any session ID that is set externally; for example, the attacker sets up a URL such as http://unsecurewebservice.com/?sessionID=1234567. The attacker then sends this URL to a valid user, who clicks on it. At this point, a valid session with the ID 1234567 is created on the server. Because the attacker determines this ID, he or she can now hijack the session, which has been authenticated using the valid user's credentials.
- **Spoofing**. An attempt to gain access to a system by using a false identity. This can be accomplished by using stolen user credentials or a false IP address. After the attacker successfully gains access as a legitimate user or host, elevation of privileges or abuse using authorization can begin.
- **SQL injection**. Failure to validate input in cases where the input is used to construct a SQL statement or will modify the construction of a SQL statement in some way. If the attacker can influence the creation of a SQL statement, he or she can gain access to the database with privileges otherwise unavailable and use this in order to steal or modify information or destroy data.
- **Throttling.** The process of limiting resource usage to keep a particular process from bogging down and/or crashing a system. Relevant as a countermeasure in DoS attacks, where an attacker attempts to crash the system by overloading it with input.

Chapter 3 – Security Design Guidelines for Web Services

Contents

Overview

Designing a Web service with security in mind presents developers and architects with an interesting set of challenges. Some are unique to service-oriented architecture and some are similar to the challenges that face enterprise Web application development teams.

A Web service is most commonly implemented as a wrapper – that is, as an interface between a client consuming the service and back-end business logic components doing the actual work. A Web service acts as a trust boundary in your application architecture. By its nature, a Web service acts as a gateway between trusted business components and less trusted or untrusted client components. For this reason, it is impossible to think about the security of a Web service without also thinking about authentication, authorization, protection of sensitive data on the network, and handling potentially malicious input. Each of these areas represents key decisions you will need to make in order to maintain the security of your application.

By following security best practices in the design of your Web service, you can use proven practices to improve your decision-making capabilities and make a cascading positive impact on the overall security of your application. Use the following design guidelines to reduce wasted effort trying to solve security problems for which there are already best practices in place to improve the security of your service.

Security Architecture and Design Issues for Web Services

During the design phase, it is important to think like an attacker and consider potential vulnerabilities that can impact your service. A clear understanding of attacks and vulnerabilities will put you in the right mindset to mitigate potential problems and create a design that is resistant to malicious attack. The following table outlines key problem areas for each category in the Web service security frame.

Vulnerability category	Potential problem due to bad design
Auditing and logging	• Failure to observe signs of intrusion • Inability to prove a user's actions • Difficulties in problem diagnosis
Authentication	• Identity spoofing • Password cracking • Elevation of privileges • Unauthorized access
Authorization	• Access to confidential or restricted data, • Tampering • Execution of unauthorized operations
Configuration management	• Unauthorized access to administration interfaces • Unauthorized ability to update configuration data • Unauthorized access to user accounts and account profiles
Exception management	• Denial of service (DoS) attacks • Disclosure of sensitive system level details • Elevation of privilege.
Message encryption	• Sniffing of confidential data off the network • Stealing users' credentials or session information
Message replay detection	• Replaying user messages to gain unauthorized access to resources or data
Message signing	• Tampering with messages on the network without detection. Failure to mutually authenticate allows attacker to send messages as if they were a legitimate user.
Message validation	• Messages containing malicious input. • Cross-site scripting or SQL injection attacks on the service or clients that rely on the service.
Sensitive data	• Confidential information disclosure and data tampering.
Session management	• Session hijacking and/or identity spoofing due to Capture of session ID.

Deployment Considerations

During the application design phase, you should review your corporate security policies and procedures together with the infrastructure on which your application is to be deployed. Frequently, the target environment is rigid, and your application design must reflect its restrictions. Sometimes design tradeoffs are required; for example, because of protocol or port restrictions or specific deployment topologies. Identify constraints early in the design phase in order to avoid surprises later, and involve members of the network and infrastructure teams to help with this process.

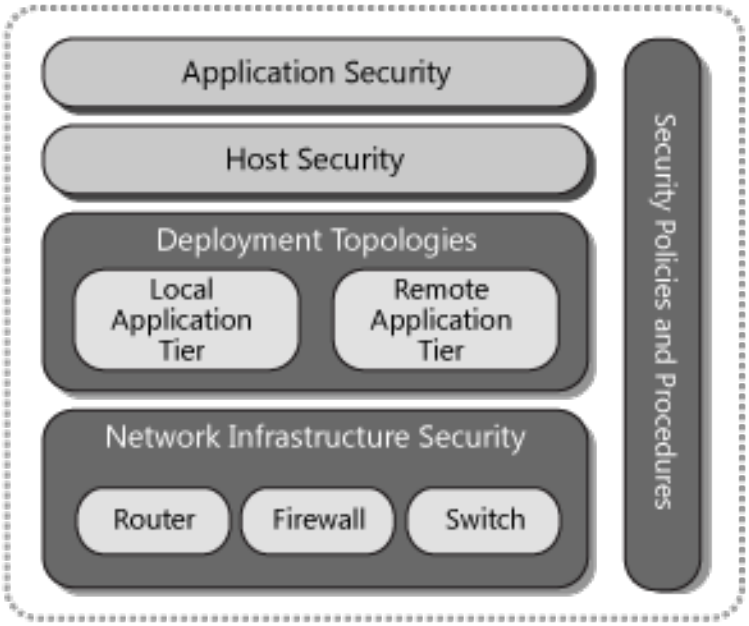


Fig. 1 – Deployment Considerations

Consider the following guidelines before deploying your Web service:

- **Identify security policies and procedures**. A security policy determines what your applications are allowed to do and what the users of the application are permitted to do. More importantly, a security policy defines restrictions to determine what applications and users are not allowed to do. When designing your applications, identify and work within the framework defined by your corporate security policy to make sure you do not breach any policy that might prevent the application from being deployed.
- **Understand network infrastructure components**. Make sure you understand the network structure provided by your target environment, as well as the baseline security requirements of the network in terms of filtering rules, port restrictions, supported protocols, and so on.
- **Identify how firewalls and firewall policies are likely to affect your application's design and deployment**. If present, firewalls separating the Internet-facing applications from the internal network, as well asadditional firewalls in front of the database, can affect your possible communication ports. Consequently, the firewall configuration can affect authentication options from the Web server to remote application and database servers. For example, Windows authentication requires additional ports.
- **Identify protocols, ports, and services**. At the design stage, consider what protocols, ports, and services are allowed to access internal resources from the Web servers in the perimeter network. Also identify the protocols and ports that the application design requires, and analyze the potential threats that can occur from opening new ports or using new protocols.

- **Communicate assumptions**. Communicate and record any assumptions made about network and application-layer security and which component will handle what task. This prevents security controls from being overlooked when both the development and network teams assume that the other team is addressing the issue. Pay attention to the security defenses that your application relies on the network to provide. Consider the implications of a change in network configuration. For example, how much security would you lose if you implement a specific network change?
- **Analyze deployment topologies**. Your application's deployment topology, and whether you have a remote application tier, are key considerations that must be incorporated into your design. If you have a remote application tier, you need to consider how to secure the network between servers in order to address the network eavesdropping threat and provide privacy and integrity for sensitive data.
- **Consider identity flow**. Also consider identity flow and identify the accounts that will be used for network authentication when your application connects to remote servers. A common approach is to use a least-privileged process account and create a duplicate (mirrored) account on the remote server with the same password. Alternatively, you might use a domain process account, which provides easier administration but is more problematic to secure because of the difficulty of limiting the account's use throughout the network. An intervening firewall or separate domains without trust relationships often makes the local account approach the only viable option.
- **Understand intranet, extranet, and Internet considerations**. Intranet, extranet, and Internet application scenarios each present design challenges. Questions that you should consider include: How will you flow caller identity through multiple application tiers to back-end resources? Where will you perform authentication? Can you trust authentication at the front end and then use a trusted connection to access back-end resources? In extranet scenarios, you also must consider whether you trust partner accounts.

Additional Resources

For more information, see "Perimeter Service Router" at http://msdn2.microsoft.com/en-us/library/aa480606.aspx .

Auditing and Logging

Auditing and logging are used to monitor and record important activities, such as transactions or user management events, on both the client and the service. Ensure that your logging design allows for the effective auditing of security-critical operations such as user management events or important business operations such as financial transactions. Be careful not to log sensitive information because the access rights to your log files may be different from access rights to protected operations in your service. Protect your log files so that an attacker cannot access or tamper with your logs.

Consider the following guidelines:

- **Audit and log access across application tiers**.
- **Back up and analyze log files regularly**.

- **Consider identity flow**.
- **Do not log sensitive information**.
- **Instrument for significant business operations**.
- **Instrument for unusual activity**.
- **Instrument for user management events**.
- **Know your baseline**.
- **Log key events**.
- **Protect and audit log files**.
- **Use log throttling**.

Each of these guidelines is briefly described in the following sections.

Audit and Log Access Across Application Tiers

Audit and log access across the tiers of your application for the purpose of non-repudiation. Use a combination of application-level logging and platform auditing features.

Back Up and Analyze Log Files Regularly

There is no point in logging activity if the log files are never analyzed. Log files should be removed from production servers on a regular basis. The frequency of removal depends on your application's level of activity. Your design should consider the way that log files will be retrieved and moved to offline servers for analysis. Any additional protocols and ports opened on the Web server for this purpose must be securely locked down.

Consider Identity Flow

Consider how your application will flow caller identity across multiple application tiers. You have two basic choices:

- You can flow the caller's identity at the operating system level by using the Kerberos protocol delegation. This allows you to use operating system–level auditing. The drawback to this approach is that it affects scalability because it means there can be no effective database connection pooling at the middle tier.
- Alternatively, you can flow the caller's identity at the application level and use trusted identities to access back-end resources. With this approach, you have to trust the middle tier, which brings a potential repudiation risk. You should generate audit trails in the middle tier that can be correlated with back-end audit trails.

Do Not Log Sensitive Information

Do not include sensitive information in your log entries. The access rights for your log files may be different than the access rights for sensitive operations and data in your service. Strip out sensitive data such as passwords, credit card numbers, or personally identifiable information (PII) before logging an error or an event to your log files.

Instrument for Significant Business Operations

Track significant business operations. For example, instrument your application to record access to particularly sensitive methods and business logic.

Instrument for Unusual Activity

Instrument your application and monitor events that might indicate unusual or suspicious activity. This enables you to detect and react to potential problems as early as possible. Unusual activity might be indicated by:

- Replays of old authentication tickets.
- Too many login attempts over a specific period of time.

Instrument for User Management Events

Instrument your application and monitor user management events such as password resets, password changes, account lockout, user registration, and authentication events. Doing this helps you to detect and react to potentially suspicious behavior. It also enables you to gather operations data; for example, to track who is accessing your application and when user account passwords need to be reset.

Know Your Baseline

Before deploying your application, audit your log files so you know what normal application behavior looks like. Knowing your baseline can help you identify an attack in progress early on and limit damage to your system.

Log Key Events

The types of events that should be logged include successful and failed logon attempts, modification of data, retrieval of data, network communications, and administrative functions such as the enabling or disabling of logging. Logs should include the time of the event, the location of the event (including the machine name), the identity of the current user, the identity of the process initiating the event, and a detailed description of the event.

Protect and Audit Log Files

Protect and audit and log files using Windows access control lists (ACLs), and restrict access to the log files. If you log events to Microsoft SQL Server® or to some custom event sink, use appropriate access controls to limit access to the event data. For example, grant write access to the account or accounts used by your application, grant full control to administrators, and grant read-only access to operators.

This makes it more difficult for attackers to tamper with log files in order to cover their tracks. Minimize the number of individuals who can manipulate the log files. Authorize access only to highly trusted accounts such as administrators.

Additional Considerations

Also keep in mind the following additional considerations:

- **Log application events on a separate, protected server**. This helps to ensure that attackers cannot tamper with logs.
- **Assign appropriate permissions to the log files**. Logs should be written by a process with write permission only. Logs should be read by users with administrative access.
- **Log application events in sufficient detail**. Provide sufficient detail to permit reconstruction of system activity.
- **Use performance counters for high-volume, per-request events**. This helps to minimize the impact on performance.

Use Log Throttling

Use log throttling to limit the number of logs as well as the size of the log entries that a single user can generate. Log throttling can protect your application from a denial of service (DoS) attack that may overwhelm your logging infrastructure and negatively impact the availability of your service.

Authentication

Authentication is the mechanism by which your clients can establish their identity with your service using a set of credentials that prove that identity. Protect your user's credentials when they are sent over the network, as well as when they are stored on the client or the server. Do not store the user's password directly but instead store a salted hash. If you use cookies or some other token to store authentication information for the client, protect that cookie as strongly as you would protect the original credentials.

Consider the following guidelines:

- **Be able to disable accounts.**
- **Do not send passwords over the wire in plaintext.**
- **Do not store passwords in user stores.**
- **Protect authentication cookies.**
- **Require strong passwords.**
- **Support password expiration periods.**
- **Use account lockout policies for end-user accounts.**

Be Able to Disable Accounts

If the system is compromised, being able to deliberately invalidate credentials or disable accounts can prevent additional attacks.

Do Not Send Passwords over the Wire in Plaintext

Plaintext passwords sent over a network are vulnerable to eavesdropping. To address this threat, secure the communication channel; for example, by using Secure Sockets layer (SSL) to encrypt the traffic.

Do Not Store Passwords in User Stores

If you must verify passwords, it is not necessary to actually store the passwords. Instead, store a one-way hash value and then recompute the hash using the user-supplied passwords. To mitigate the threat of dictionary attacks against the user store, use strong passwords and incorporate a random salt value with the password.

Protect Authentication Cookies

A stolen authentication cookie is a stolen logon. Protect authentication tickets using encryption and secure communication channels. Also limit the time interval in which an authentication ticket remains valid, to counter the spoofing threat that can result from replay attacks, where an attacker captures the cookie and uses it to gain illicit access to your site. Reducing the cookie timeout does not prevent replay attacks but does limit the amount of time the attacker has to access the site using the stolen cookie.

Require Strong Passwords

Do not make it easy for attackers to crack passwords. There are many guidelines available, but a general practice is to require a minimum of eight characters and a mixture of uppercase and lowercase characters, numbers, and special characters. Whether you are using the platform to enforce these requirements for you, or you are developing your own validation, this step is necessary to counter dictionary attacks where an attacker tries to crack a password through systematic trial and error. Use regular expressions to help with strong password validation.

Support Password Expiration Periods

Passwords should not be static and should be changed as part of routine password maintenance through password expiration periods. Consider providing this type of facility during application design.

Use Account Lockout Policies for End-user Accounts

Disable end-user accounts or write events to a log after a set number of failed logon attempts. If you are using Windows authentication, such as NTLM or the Kerberos protocol, these policies can be configured and applied automatically by the operating system. With Forms authentication, these policies are the responsibility of the application and must be incorporated into the application design. Be careful to ensure that account lockout policies cannot be abused in DoS attacks.

Additional Resources

For information on key authentication patterns, see the following resources:

- "Direct Authentication" at http://msdn.microsoft.com/en-us/library/aa480566.aspx .
- "Brokered Authentication" at http://msdn.microsoft.com/en-us/library/aa480560.aspx .
- "Brokered Authentication: Kerberos" at http://msdn.microsoft.com/en-us/library/aa480562.aspx .
- "Brokered Authentication: X.509 PKI" at http://msdn.microsoft.com/en-us/library/aa480565.aspx .
- "Brokered Authentication: Security Token Service (STS)" at http://msdn.microsoft.com/en-us/library/aa480563.aspx .

Authorization

Authorization is the mechanism by which you control the operations and resources an authenticated client can access. Where possible, authenticate your users on the same application tier where you authorize your users. Run your application in a least-privileged account and use impersonation to increase privileges only when necessary and for the shortest time possible. Use ACLs to restrict the system resources that your application and its users can access.

Consider the following guidelines:

- **Tie authentication to authorization on the same tier.**
- **Consider authorization granularity.**
- **Know your authorization options.**
- **Enforce separation of privileges.**
- **Restrict user access to system-level resources.**
- **Use least-privileged accounts.**
- **Use multiple gatekeepers.**

Tie Authentication to Authorization on the Same Tier

Where possible, authenticate your users on the same application tier where you authorize your users. The further you separate the time of check (authentication) from the time of use (authorization), the larger window of opportunity you give an attacker to subvert your authorization mechanism.

Consider Authorization Granularity

There are three common authorization models, each with varying degrees of granularity and scalability:

- **The most granular approach relies on impersonation.** Resource access occurs using the security context of the caller. Windows ACLs on the secured resources (typically files or tables, or both) determine whether the caller is allowed to access the resource. If your application provides access primarily to user-specific resources, this approach may be valid. It has the added advantage that operating system–level auditing can be performed across the tiers of your application, because the original caller's security context flows at the operating system level and is used for resource access. However, the approach suffers from

poor application scalability because effective connection pooling for database access is not possible. As a result, this approach is most frequently found in limited scale intranet-based applications.

- **The least granular but most scalable approach uses the application's process identity for resource access**. This model is referred to as the *trusted subsystem* or sometimes as the *trusted server model*. Although this approach supports database connection pooling, it means that the permissions granted to the application's identity in the database are common, irrespective of the identity of the original caller. The primary authorization is performed in the application's logical middle tier using roles, which group together users who share the same privileges in the application. Access to classes and methods is restricted based on the role membership of the caller. To support the retrieval of per-user data, a common approach is to include an identity column in the database tables and to use query parameters to restrict the retrieved data. For example, you may pass the original caller's identity to the database at the application (not operating system) level through stored procedure parameters.
- **The third option is to use a limited set of identities for resource access based on the role membership of the caller**. This is really a hybrid of the two models described earlier. Callers are mapped to roles in the application's logical middle tier, and access to classes and methods is restricted based on role membership. Downstream resource access is performed using a restricted set of identities determined by the current caller's role membership.

Know Your Authorization Options

Know your authorization options and choose the most appropriate one for your scenario. First decide if you want to use resource-based or role-based authorization. Resource-based authorization uses ACLs on the resource to authorize the original caller. Role-based authorization allows you to authorize access to service operations or resources based on the group a user is in.

- If you choose to use role-based authorization, you can store your roles in Windows groups or in ASP.NET roles.
- If you are using Active Directory, consider using Windows groups based on ease of maintenance and the fact that you maintain both roles and credentials in the Active Directory store. If you are not using Active Directory, consider using ASP.NET roles and the ASP.NET role provider.

Restrict User Access to System-level Resources

System-level resources include files, folders, registry keys, Active Directory objects, database objects, event logs, and so on. Use ACLs to restrict which users can access what resources and the types of operations that they can perform. Pay particular attention to anonymous Internet user accounts; lock these down with ACLs on resources that explicitly deny access to anonymous users.

Use Least-privileged Accounts

You might need to create a custom service account to isolate your application from other applications on the same server, or to be able to audit each application separately.

Use Multiple Gatekeepers

On the server side, you can use IP Security Protocol (IPSec) policies to provide host restrictions to restrict server-to-server communication. For example, an IPSec policy might restrict any host apart from a nominated Web server from connecting to a database server. Internet Information Services (IIS) provides Web permissions and Internet Protocol/ Domain Name System (IP/DNS) restrictions. IIS Web permissions apply to all resources requested over HTTP regardless of the user. The permissions do not provide protection if an attacker manages to log on to the server. For this, NTFS permissions allow you to specify per-user ACLs. Finally, ASP.NET provides URL authorization and File authorization together with principal permission demands. By combining these gatekeepers, you can develop an effective authorization strategy.

Additional Resources

For more information, see “Trusted Subsystem” at http://msdn.microsoft.com/en-us/library/aa480587.aspx

Configuration Management

Security settings, authentication, authorization, logging, and other parameters can be set in configuration files. Encrypt configuration sections that contain sensitive data such as connection strings to your SQL database. Protect access to your configuration settings so that an attacker cannot modify security settings for your service.

Consider the following guidelines:

- **Consider your key storage location.**
- **Encrypt sensitive sections of configuration files.**
- **Use ACLs to protect your configuration files.**
- **Use secure settings for various operations of Web services.**

Consider Your Key Storage Location

If you need to store keys, choose platform features over creating your own mechanism. The Data Protection API (DPAPI)– and RSA-protected configuration providers used to encrypt sensitive data in configuration files can use either machine stores or user stores for key storage. You can either store the key in the machine store and create an ACL for your specific application identity, or store the key in a user store. In the latter case, you need to load the user account’s profile to access the key.

Use machine-level key storage when:

- Your application runs on its own dedicated server with no other applications.

- You have multiple applications on the same server that run using the same identity, and you want those applications to be able to share sensitive information and the same encryption key.

Use user-level key storage if you run your application in a shared hosting environment and you want to ensure that your application's sensitive data is not accessible to other applications on the server. In this scenario, each application should have a separate identity so that they all have their own individual and private key stores.

Encrypt Sensitive Sections of Configuration Files

Configuration files may contain sensitive information, such as connection strings to your database. Encrypt sensitive information in your configuration files using the DPAPI provider with the machine-key store. You can use the aspnet_regiis command-line tool to encrypt sections of your configuration file.

Use ACLs to Protect Your Configuration Files

Use ACLs to lock your configuration files down and restrict inappropriate access. Modifications to your configuration settings, especially binding options, can have a major impact on the security of your service.

Use Secure Settings for Various Operations of Web Services

Set your configuration options to take advantage of features such as message and transport security, which protect the communication channel between your client and your service.

Exception Management

Exception management is the means by which you expose and consume exception information within your service and send it back to your clients. Be careful not to reveal internal application details to your clients as this information could assist an attacker trying to exploit your service. Catch and handle exceptions so that error conditions do not lead to a service crash and a DoS condition for your clients. Fail to a secure state so that an error condition does not result in your application running at higher privilege or accessing resources insecurely.

Consider the following guidelines:

- **Catch exceptions.**
- **Do not log private data such as passwords.**
- **Do not reveal sensitive system or application information.**
- **Log detailed error messages.**

Catch Exceptions

Use structured exception handling and catch exception conditions with try/catch blocks. Doing so avoids leaving your application in an inconsistent state that may lead to information disclosure. It also helps protect your application from DoS attacks. Decide how to propagate

exceptions internally in your application and give special consideration to what occurs at the application boundary. Catch and wrap exceptions only where it adds value or will provide additional information relevant to the exception.

Do Not Log Private Data Such as Passwords

Exception handlers often will result in an error log entry. Be careful not to log sensitive information such as passwords, credit card numbers, or privately identifiable information (PII). This information may make it easier to decipher error logs; however, sensitive data is not secure in log files and could be accessed by users who would not normally have access to this information.

Do Not Reveal Sensitive System or Application Information

In the event of a failure, do not expose information that could lead to information disclosure. For example, do not expose stack trace details that include function names and line numbers in the case of debug builds (which should not be used on production servers). Instead, return generic error messages to the client.

Log Detailed Error Messages

Send detailed error messages to the error log. Send minimal information to the consumer of your service or application, such as a generic error message and custom error log ID that subsequently can be mapped to a detailed message in the event logs. Make sure that you do not log passwords or other sensitive data.

Additional Resources

For more information on how to handle exceptions, see “Exception Shielding” at http://msdn.microsoft.com/en-us/library/aa480591.aspx .

Message Protection

Message protection covers the mechanisms used to protect sensitive data in transit over the network from unauthorized access or modification. Use message or transport security to protect your messages in transit. Do not try to create your own cryptographic routines; use the platform-provided cryptography instead.

Consider the following guidelines:

- **Use message security or transport security to encrypt and sign your messages.**
- **Use platform-provided cryptography.**
- **Use platform features for key management.**
- **Periodically change your keys.**

Use Message Security or Transport Security to Encrypt and Sign Your Messages

Use message security or transport security to encrypt your messages on the network. Message security encrypts each individual message to protect sensitive data. Transport security secures the end-to-end network connection to protect the network traffic. Message encryption protects the contents of your message from being stolen and read. Message signing protects the integrity of your message and guarantees the authenticity of the sender.

Use Platform-Provided Cryptography

Cryptography is notoriously difficult to develop. The Windows crypto APIs have been proven to be effective. These APIs are implementations of algorithms derived from years of academic research and study. Some developers believe that a less well-known algorithm can provide more security, but this is not true. Cryptographic algorithms are mathematically proven; therefore, the more scrutiny they receive, the better. An obscure algorithm will not protect a flawed cryptographic implementation from a determined attacker.

- For hashing, use SHA1. For integrity checking, use HMACSHA1 or a digital signature mechanism.
- Consider using the **XMLEncryption** mechanisms when you need to encrypt different parts of a document under different keys, or if you only want to encrypt small sections of a document.
- Use X.509 and S/MIME encryption if you are using an internal or external public key infrastructure (PKI) based on digital certificates.

Use Platform Features for Key Management

Use platform features where possible to avoid managing keys yourself. For example, by using DPAPI, the encryption key is derived from an account's password, so Windows handles this for you.

Periodically Change Your Keys

You should change your encryption keys from time to time because a static secret is more likely to be discovered over time. Did you write it down somewhere? Did the administrator with access to the secrets change positions in your company or leave the company? Are you using the same session key to encrypt communication for a long time? Also, do not overuse keys.

Additional Resources

For more information, see the following resources:

- "Data Confidentiality" at http://msdn.microsoft.com/en-us/library/aa480570.aspx .
- "Data Origin Authentication" at http://msdn2.microsoft.com/en-us/library/aa480571.aspx .

Message Validation

Message validation is used to protect your service from malformed messages and message parameters. Message schemas can be used to validate incoming messages, and custom

validators can be used to validate parameter data before your service consumes it. Do not trust input from any source that the client can influence, such as cookies, headers, IP address, or the content of messages sent to your service. Do trust input from a database, the file system, or anything else outside the trust boundary of your service. Use message schemas and data validators to check for format, range, length, and type. Do not rely on client-side validation; make all security decisions based on server-side validation.

Consider the following guidelines:

- **Do not trust input.**
- **Verify the message payload against a schema.**
- **Verify the message size, content, and character sets.**
- **Filter, scrub, and reject input and output before additional processing.**

Do Not Trust Input

An attacker passing malicious input can attempt SQL injection, cross-site scripting, and other injection attacks that aim to exploit your application's vulnerabilities. Check for known good data and constrain input by validating it for type, length, format, and range.

Verify the Message Payload Against a Schema

If you need to validate parameters, message contracts, or data contracts passed to operations, use schemas to validate the incoming message. Schemas provide a wide range of input validation without the need for custom code or validation routines.

Verify the Message Size, Content, and Character Sets

Validate incoming messages to ensure that they match your expectations regarding size, content, and character encoding. If a message is much larger than expected or contains encoding other than what your service expects, you may be under attack. Reject the message and log the occurrence so that auditing can determine if an attack is underway.

Filter, Scrub, and Reject Input and Output Before Additional Processing

Filter and reject input before allowing the data to be processed by downstream components. Because malicious input may target the routines that process your input, it is important to detect and reject malformed input early before additional processing occurs. Scrub your output before sending to the client as it may include potentially dangerous input from sources such as the file system or your database that is outside of your service trust boundary.

Additional Resources

For more information, see "Message Validator" at http://msdn2.microsoft.com/en-us/library/aa480600.aspx .

Sensitive Data

Sensitive data refers to confidential information that your service processes, transmits, or stores. Protect sensitive data on the network, in configuration files, in local memory or file storage, and in databases and log files. Ensure that you are aware of all sensitive information your service transmits or processes. Sensitive data includes user identity and credentials as well as any personally identifiable information (PII) such as social security number.

Consider the following guidelines:

- **Do not store database connections, passwords, or keys in plaintext.**
- **Do not store secrets if you can avoid it.**
- **Do not store secrets in code.**
- **Encrypt sensitive data in configuration files.**
- **Encrypt sensitive data over the network.**
- **Retrieve sensitive data on demand.**

Do Not Store Database Connections, Passwords, or Keys in Plaintext

Avoid storing secrets such as database connection strings, passwords, and keys in plaintext. Use encryption and store encrypted strings.

Do Not Store Secrets if You Can Avoid It

Storing secrets in software in a completely secure fashion is not possible. An administrator, who has physical access to the server, can access the data. For example, it is not necessary to store a secret when all you need to do is verify whether a user knows the secret. In this case, you can store a hash value that represents the secret and compute the hash using the user-supplied value to verify whether the user knows the secret.

Do Not Store Secrets in Code

Do not hard-code secrets in code. Even if the source code is not exposed on the Web server, it is possible to extract string constants from compiled executable files. A configuration vulnerability may allow an attacker to retrieve the executable.

Encrypt Sensitive Data in Configuration Files

Configuration files may contain sensitive information, such as connection strings to your database. Encrypt sensitive information in your configuration files by using the DPAPI provider with the machine-key store. You can use the aspnet_regiis command-line tool to encrypt sections of your configuration file.

Encrypt Sensitive Data over the Network

Consider where items of sensitive data, such as credentials and application-specific data, are transmitted over a network link. If you need to send sensitive data between the Web server and browser, consider using SSL. If you need to protect server-to-server communication, such as between your Web server and database, consider IPSec or SSL.

Retrieve Sensitive Data on Demand

The preferred approach is to retrieve sensitive data on demand when it is needed instead of persisting or caching it in memory. For example, retrieve the encrypted secret when it is needed, decrypt it, use it, and then clear the memory (variable) used to hold the plaintext secret.

Session Management

Sessions are the means by which an application maintains stateful communication with a client over time. Protect your session tokens or identifiers so that an attacker cannot gain access and steal a user's session. Reduce the timeouts on your sessions to lower the chances of an attacker being able to steal a session after a user has finished using your application.

Consider the following guidelines:

- **Authenticate and authorize access to the session store.**
- **Avoid storing sensitive data in session stores.**
- **Reduce session timeouts.**
- **Secure the channel to the session store.**

Authenticate and Authorize Access to the Session Store

Authenticate and authorize access to your session store. The session store contains identifiers that maintain session state for your users. This information can be used by an attacker to hijack user sessions and take actions on their behalf. Use authentication and authorization to restrict direct access to this store so that only your service can get at the information stored within.

Avoid Storing Sensitive Data in Session Stores

Avoid storing sensitive information, such as user credentials, in your session store. The permissions required to access your session store may be different than the permissions necessary to access sensitive data or operations in your service. The session store should contain the bare minimum of information to track a session ID and maintain the session state for your users.

Reduce Session Timeouts

Reduce the lifetime of sessions to mitigate the risk of session hijacking and replay attacks. The shorter the session, the less time an attacker has to capture a session cookie and use it to access your application.

Secure the Channel to the Session Store

Consider how session state is to be stored. For optimum performance, you can store session state in the Web application's process address space. However, this approach has limited scalability and implications in Web farm scenarios, where requests from the same user cannot be guaranteed to be handled by the same server. In this scenario, an out-of-process state store on a dedicated state server or a persistent state store in a shared database is required. You

should secure the network link from the Web application to state store by using IPSec or SSL to mitigate the risk of eavesdropping.

PART II

WCF Security Fundamentals

In This Part:

- **WCF Security Fundamentals**
- **Authentication, Authorization and Identities in WCF**
- **Impersonation and Delegation in WCF**
- **Message and Transport Security in WCF**
- **WCF Bindings Fundamentals**

Chapter 4 – WCF Security Fundamentals

Objectives

- Understand basic security-related concepts in WCF.
- Understand what bindings and behaviors are and how they are used in WCF.
- Understand the differences between transport security and message security.
- Understand authorization and roles in the context of WCF.
- Understand impersonation and delegation in the context of WCF.
- Understand, at a high level, what auditing is and what options are available for auditing in WCF.

Overview

Securing your WCF service requires knowledge of the WCF security features related to auditing and logging, authentication, authorization, confidentiality, and integrity. Use behaviors and bindings to configure security for your WCF service. Bindings and behaviors allow you to configure transfer security, authentication, authorization, impersonation, and delegation as well as auditing and logging. Transfer security is the means by which WCF secures messages over the network. WCF gives you two options to implement transfer security: transport security and message security. Transport security secures the entire communication channel (e.g., by using SSL), while message security secures each message individually. WCF supports a variety of authentication options including username, Windows, and certificate authentication. Depending on your authentication method, you can choose to authorize your users by using role-based security or resource-based security. Use WCF impersonation and delegation to flow the identity and security context of your client-side original caller to the back end in order to support a granular authorization approach.

Key Security Features

Any Service-Oriented Architecture (SOA) needs to support security features that provide auditing, authentication, authorization, confidentiality, and integrity for the messages exchanged between the client and the service. Windows Communication Foundation (WCF) provides these security features by default for any application that is built on top of the WCF framework.

Key security features include:

- **Auditing**. Effective auditing and logging is the key to non-repudiation. Non-repudiation guarantees that a user cannot deny performing an operation or initiating a transaction.
- **Authentication**. Authentication allows you to confidently identify the clients of your service. These might be end users, other services, processes, or computers. WCF supports mutual authentication, which identifies both the client and the service in tandem, to help in preventing man-in-the-middle attacks.

- **Authorization**. Authorization determines what system resources and operations can be accessed by the authenticated user. This allows you to grant specific application and resource permissions for authenticated users.
- **Confidentiality**. Confidentiality, also referred to as privacy, is the process of making sure that data remains private and confidential, and that it cannot be viewed by unauthorized users. Encryption is frequently used to enforce confidentiality. Privacy is a key concern, particularly for data / messages passed across networks.
- **Integrity**. Integrity is the guarantee that data is protected from accidental or deliberate modification. Like privacy, integrity is a key concern, particularly for data / messages passed across networks. Integrity for data in transit is typically provided by using hashing techniques and message authentication codes.

Scope of WCF Security

The above fundamental security features are covered in the following WCF features:

- **Transfer security.** Responsible for providing message confidentiality, data integrity, and authentication of communicating parties.
- **Authorization.** Responsible for providing a framework for making authorization decisions.
- **Auditing.** Responsible for logging security-related events to the audit log.

WCF provides access to these features through bindings and behavior configuration.

Bindings and Behaviors

When you create an overall security policy – for example, transfer security with authentication and authorization for your services – you can use bindings and behaviors to configure the required settings.

Bindings and behaviors are described as follows:

- **Bindings.** Bindings control the security mode, client credential type, and other security settings.
- **Behaviors.** Service behaviors control impersonation levels, how client credentials are authenticated and authorized, and service credentials.

You can configure bindings and behaviors, or you can program against the object model. Your binding selection determines the available security options for WCF. The following able summarizes the most commonly used bindings in WCF.

Binding	Common scenarios	Default security settings
basicHttpBinding	Legacy Web service protocols	No security
netTcpBinding	Binary TCP communication between machines	Transport security with Windows authentication
wsFederationHttpBinding	Federated security scenarios	Message security with issue token authentication

wsHttpBinding	Leveraging security standards (WS-Security)	Message security with Windows authentication

By default, every WCF binding will provide transfer security and user authentication except for **BasicHttpBinding**. If necessary, you can change the security settings to suit your scenario requirements.

Transfer Security

After selecting a binding, you can decide which type of transfer security, otherwise known as security mode, to use for your WCF service. You can provide security on the transport level or the message level. Each option has its own advantages and disadvantages. For instance, transport security secures the entire communication channel (e.g., by using SSL) and therefore only supports point-to-point communication over a single transport. Message security protects each message individually and therefore supports multipoint communication, multiple transports, or even partial message encryption if necessary. Most scenarios are best supported by using transport security. The following security modes are available across the standard bindings.

Mode	Description
None	No security is provided; all information is passed in clear text.
Transport	Mutual authentication and message protection are provided at the transport level.
Message	Mutual authentication and message protection are provided at the message level
Both	Mutual authentication and message protection are provided at both the transport and message levels. This is far more than is necessary for most scenarios.
TransportWithMessageCredential	Client authentication is provided at the message level, and message protection and service authentication are provided at the transport level.
TransportCredentialOnly	Mutual authentication is provided at the transport level; no message protection is provided. This option is available only on **basicHttpBinding**.

Transport Security

When using transport security, the user credentials and claims are passed using the transport layer. In other words, user credentials are transport-dependent, which allows fewer authentication options compared to message security. Each transport protocol (TCP, IPC, MSMQ, or HTTP) has its own mechanism for passing credentials and handling message

protection. The most common approach for this is to use Secure Sockets Layer (SSL) for encrypting and signing the contents of the packets sent over Secure HTTP (HTTPS).

Transport security is used to provide point-to-point security between the two endpoints (service and client). If there are intermediary systems between the client and the service, each intermediate point must forward the message over a new SSL connection.

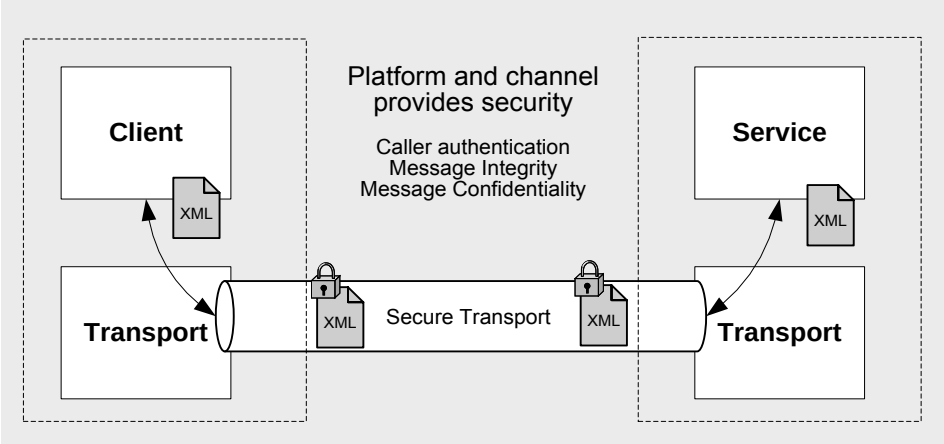


Figure 1. Transport Security

Use transport security for the following scenarios:

- You are sending a message directly from your application to a WCF service and the message will not be routed through intermediate systems.
- You have both the service and the client in an intranet.

Using transport security has the following advantages:

- It provides interoperability, meaning that communicating parties do not need to understand the WS-Security specification.
- It may result in better performance.
- Hardware accelerators can be used to further improve performance.

Using transport security has the following disadvantages:

- Because security is applied on a point-to-point basis, there is no provision for multiple hops or routing through intermediate application nodes.
- It supports a limited set of credentials and claims compared to message security.
- It is transport-dependent upon the underlying platform, transport mechanism, and security service provider such as NTLM or Kerberos.

Message Security

When using message security, the user credentials and claims are encapsulated in every message using the WS-Security specification to secure messages. This option gives the most

flexibility from an authentication perspective. You can use any type of security credentials you want, largely independent of transport, as long as both the client and the service agree.

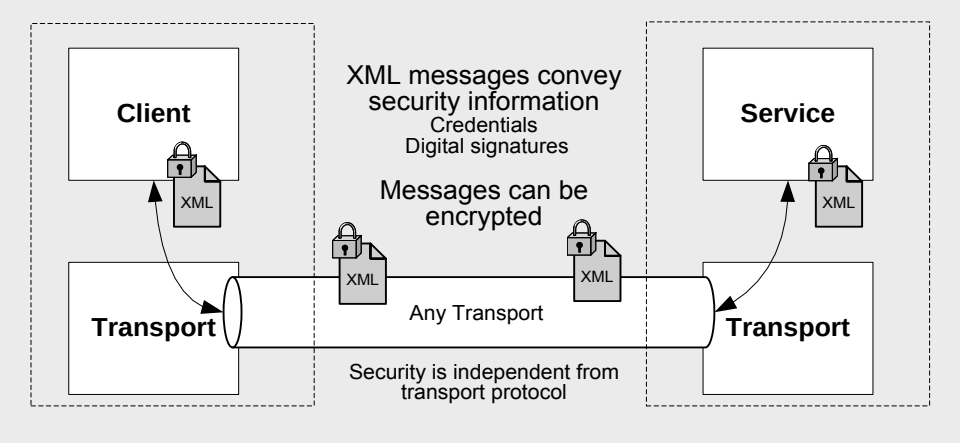


Figure 2. Message Security

Use message security for the following scenarios:

- You are sending a message to a WCF service, and the message is likely to be forwarded to other WCF services or may be routed through intermediate systems.
- Your WCF clients are accessing the WCF service over the Internet, it's possible that other intermediate systems may be used in between, and security is your top consideration.

Using message security has following advantages:

- It provides end-to-end security. Because message security directly encrypts and signs the message, having intermediaries does not break the security.
- It allows partial or selective message encryption and signing, thus improving overall application performance.
- Message security is transport-independent and can be used with any transport protocol.
- It supports a wide set of credentials and claims, including issue token, which enables federated security.

Using message security has following disadvantages:

- This option may reduce performance compared to transport security because each individual message is encrypted and signed.
- It does not support interoperability with older ASP.NET Web Services (ASMX) clients because it requires both the client and service to support WS-Security specifications.

Protection Levels

The following table shows the various protection levels available with message security.

Mode	Description
None	Disables message protection.
Sign	Signs but does not encrypt the message; should be used when data integrity is important.
EncryptAndSign	Signs and encrypts the message.

If you are using message security, you can configure message protection to sign but not encrypt each message. This allows you to verify the integrity of a message without the overhead of encryption in case there is no sensitive data requiring protection. With transport security, you cannot modify the level of protection because it is transport-dependent and the WCF framework does not control transport standards.

Configuring the Protection Level

Consider turning off encryption and only signing your message when you just need to validate the integrity of the information without concerns of confidentiality. This may be useful for operations or service contracts in which you need to validate the original sender but no sensitive data is transmitted. When reducing the protection level, be careful that the message does not contain any Personally Identifiable Information (PII). Configuring the service and the operation to only sign the message is shown in the following examples:

Service Contract Example of ProtectionLevel.Sign

The following is an example of using **ProtectionLevel.Sign** at the Service Contract level:

```
[ServiceContract(ProtectionLevel=ProtectionLevel.Sign]
public interface IService
{
 string GetData(int value);
}
```

Operation Contract Example of ProtectionLevel.Sign (For Granular Control)

The following is an example of using **ProtectionLevel.Sign** at the **OperationContract** level:

```
[OperationContract(ProtectionLevel=ProtectionLevel.Sign]
string GetData(int value);
```

Service Credentials Negotiation

In addition to client authentication, a WCF service needs to provide its credentials to the clients in order to support mutual authentication. Service credentials should be negotiated for mutual authentication.

By default, message security supports negotiation of service credentials for mutual authentication, where the service is requested for its security token at run time before any messages are exchanged.

In message security, you can configure your service to not negotiate service credentials. If you configure your service to not negotiate service credentials:

- The client and service needs to be in the same domain when using Windows authentication.
- The service certificate should be accessible to the client when using non-Windows authentication such as username or certificate authentication.

Not negotiating the service credentials improves security because clients who do not have access to the service certificate cannot access the service. However, this increases the administrative overhead of distributing the service certificates to trusted clients out-of-band.

Note: When using transport security, the service credentials are always negotiated and you have no control over configuration.

Secure Session

A secure session is a message security feature that reduces the overhead of one-off key exchange and validation. By default, secure sessions are enabled for message security. Secure sessions are more efficient if the caller makes multiple calls to the service; if the caller makes a single call, it may be more efficient to disable secure sessions.

A secure session can be established between the client and server by creating a security context token. All subsequent message exchanges will use this token, thereby creating a secure session. By default, the lifetime for this token is 15 minutes when issued, and the token is reissued if it is required beyond 15 minutes. Therefore, when multiple messages are exchanged in a 15-minute lifespan, both the messages will be secured by using the same security context token, so security in this case will be weaker. To overcome this vulnerability, you can use derived keys, where two keys are derived from a symmetric key. You can use one of the keys to sign the Simple Object Access Protocol (SOAP) message and the other to encrypt various parts of the SOAP message.

Authentication

The WCF authentication options available to you depend on the transfer security mode you use. Your choice of binding will also play a role in the authentication options because not all transport or message user credentials are supported in every binding.

Transport Security Mode Authentication Options

The follow authentication options are available when using transport security mode:

- **None**. When using this option, the WCF service does not authenticate the callers. This is not the recommended option from a security perspective – avoid using this option wherever possible.
- **Basic**. This option is available with the HTTP protocol only. The client is authenticated using the username and password against Active Directory. The client credentials are transported using the Base64 encode string, which is literally like a clear string and therefore is not the

most secure option. The service is authenticated by the SSL certificate used for secure communication.

- **NTLM**. This option is available with the HTTP protocol only. The client is authenticated using a challenge-response scheme against Windows accounts. The NTLM option is well suited for a workgroup environment. NTLM authentication is more secure than either Digest or Basic authentication. The service is authenticated using the Windows credentials of the process identity or using an SSL certificate if you are using the HTTP protocol.
- **Windows**. The Windows option tells the WCF service to use Kerberos when in a domain, or NTLM when deployed in a workgroup environment. This option uses a Windows token presented by the caller to authenticate against Active Directory. This is the most secure option compared to Basic, Digest, or NTLM authentication. The service is authenticated using the Windows credentials of the process identity or an SSL certificate if you are using the HTTP protocol.
- **Certificate**. When using this option, the caller presents an X.509 client certificate that the WCF service either validates with peer trust or trusts based on the issuer of the certificate. This option should be used when Windows authentication is not possible, as in the case of business-to-business (B2B) scenarios. The service is authenticated with the service certificate or by using an SSL certificate if you are using the HTTP protocol.

Message Security Mode Authentication Options

The follow authentication options are available when using message security mode:

- **None**. When using this option, the WCF service does not authenticate the callers. This is not the recommended option from a security perspective – avoid using this option wherever possible.
- **Windows**. When using this option, the WCF service uses Kerberos when in a domain, or NTLM when deployed in a workgroup environment. This option uses the Windows token presented by the caller in order to authenticate against the Active Directory. The service is authenticated using the Windows credentials of the process identity.
- **Username**. When using this option, the caller provides the username and password to the service. The service can authenticate against Windows, use membership providers such as **SqlMembershipProvider**, or use a custom validator to validate against the custom store. You should choose this option only when Windows authentication is not possible. The service is authenticated with a service certificate.
- **Certificate**. When using this option, the caller presents an X.509 client certificate; the WCF service looks up the certificate information on the host side and either validates it (peer trust) or trusts the issuer (chain trust) of the client certificate. This option should be used when Windows authentication is not possible, or in B2B scenarios. Service is authenticated with a service certificate.
- **Issue token**. When using this option, the client and service depend on a secure token service (STS) to issue tokens that the client and service trusts. Microsoft Windows CardSpace™ is a typical example of an STS.

Authorization in WCF

WCF supports the following three basic authorization approaches:

- **Role-based**. Access to WCF operations is secured based on the role membership of the caller. The role store can be Windows groups, ASP.NET roles, or a custom role store.
- **Identity-based**. WCF supports an Identity Model feature, which is an extension to role-based authorization. Identity Model enables you to manage claims and policies to authorize clients. With this approach, you can verify claims contained within the authenticated users' credentials.
- **Resource-based**. Individual resources are secured using Windows access control lists (ACLs). The WCF service impersonates the caller prior to accessing resources, which allows the operating system to perform standard access checks. All resource access is performed using the original caller's security context.

Role-based Authorization

WCF provides the following options for role-based authorization:

- **Windows groups**. If your WCF services and clients are deployed in the same Windows domain, you can use Windows groups for authorization.
- **ASP.NET roles**. Use ASP.NET roles if you have fine-grained roles requirements, or if the users cannot be mapped to Windows domain accounts. This option uses the Role Manager feature and provides three different role providers based on the role store:
    - **SqlRoleProvider**. If your role information is stored in a Microsoft SQL Server® database, consider using the **SqlRoleProvider** for role-based authorization.
    - **WindowsTokenRoleProvider.** If your roles are Window groups, and you want to leverage the Role Manager feature as a consistent way to check the role membership of your users, regardless of the underlying data store, consider using the **WindowsTokenRoleProvider** for role-basedauthorization.
    - **AuthorizationStoreRoleProvider**. If your role information is stored using the AzMan policy store in an XML file, in Active Directory, or in Active Directory Application Mode (ADAM), consider using the **AuthorizationStoreRoleProvider** for role-based authorization.
- **Custom Roles**. If your role information is stored in a custom store such as a SQL Server database, create a custom authorization policy to authorize your users.

Note: It is recommended that you implement a custom role provider, using the Role Manager feature, for your custom role store rather than using the custom roles option.

Impersonation / Delegation

Impersonation is a technique that WCF services use to assume the original caller's identity in order to authorize access to service resources (such as files or database tables). Service resources can be resources that are local to the service machine, or they can be resources that are remote to the service machine. Impersonation is used to access resources on the same

machine as the service, while delegation is used to access resources that are remote to the service.

Delegation allows you to use an impersonation token to access network resources. Your ability to use delegation depends on the authentication mechanism in use and appropriate account configuration.

Controlling Impersonation at the Service Side

You can control impersonation at the service side by using declarative impersonation. You can use the **ImpersonationOption** enumeration along with the **OperationBehaviorAttribute** attribute to control impersonation. The following impersonation options are available:

- **NotAllowed**. Impersonation is not performed in a particular operation.
- **Allowed.** Impersonation is performed if the original Windows token is available and the service is configured to impersonate on all operations using **ImpersonateCallerForAllOperations** in **ServiceAuthorizationBehavior**.
- **Required.** Impersonation is performed; the Windows identity token is required to be available.

Controlling Impersonation at the Client Side

You can control impersonation at the client side and prevent WCF services from using client identities to access local resources. Windows credentials have an **AllowedImpersonationLevel** property that is set to one of the following **TokenImpersonationLevel** options in order to control the impersonation level:

- **None**. The WCF service cannot authenticate or impersonate the user.
- **Anonymous.** The WCF service authenticates clients as anonymous, but cannot obtain information for identification or impersonation.
- **Identification.** The WCF service can authenticate clients and get information for identification, but cannot impersonate the clients. This is the default value.
- **Impersonation.** The WCF service can authenticate, get information for identification, and impersonate clients on local systems.
- **Delegation.** The WCF service can authenticate, get information for identification, and impersonate clients on local as well as remote systems.

Auditing

WCF Auditing allows you to audit security events such as authentication and authorization failures. WCF service auditing can allow you to detect an attack that has occurred or is in progress. In addition, auditing can help you debug security-related problems.

Auditing can be enabled via configuration by using the **ServiceSecurityAuditBehavior** element. The event log location for auditing the events can be specified in the **auditLogLocation** attribute.

WCF allows you to log the events that succeed or fail or both for auditing purpose. WCF provides auditing of these events both at the message authentication level and the service authorization level by using **messageAuthenticationAuditLevel** and **serviceAuthorizationAuditLevel** attributes, respectively. You can also suppress the failures that occur during auditing by setting the **suppressAuditFailure** property to true, which prevents an exception from being thrown if auditing fails (for instance, if the log files fill up).

WCF Message Logging allows you to log malformed SOAP messages or to trace incoming messages. Message Logging allows you to specify different logging levels that you can use to diagnose and analyze your applications in case of any problems. It also allows you to log the message at the Service level or the Transport level.

Chapter 5 – Authentication, Authorization. and Identities in WCF

Objectives

- Understand the WCF authentication options for security: message, transport, mixed, and both.
- Understand the WCF authorization options based on role, identity, and resource.
- Understand the various WCF identities and the meaning of process identity, security principle, and ServiceSecurityContext.
- Understand common implementation models that employ specific authentication, authorization, and identity options.

Overview

Your choice of an appropriate authentication and authorization option for your WCF service should be based on your particular deployment scenario, including credential store, location of clients on the Internet or intranet, and authorization constraints. Use authentication to positively identify the client consuming your service. Use authorization to restrict access to resources, or to make business decisions based on user roles.

Client Authentication and Service Authentication

When considering authentication, you may be used to thinking primarily of the client identity. However, in the context of WCF, authentication typically refers to *mutual* authentication. Mutual authentication not only allows positive identification of the clients, but also allows clients to positively identify the WCF services to which they are connected. Mutual authentication is especially important for Internet-facing WCF services, because an attacker may be able to spoof the WCF service and hijack the client's calls in order to reveal sensitive data.

The service credentials to be used depend largely on the client authentication scheme you choose. Typically, if you are using non-Windows client authentication such as username or certificate authentication, a service certificate is used for both service authentication and message protection. If you are using Windows client authentication, the Windows credentials of the process identity can be used for both service authentication and message protection.

Transfer Security Modes

Your WCF authentication options depend on the transfer security mode being used. For this reason, your authentication choices are partly determined by your transfer security mode. WCF offers the following transfer security modes:

- **Message security**. When using message security, the user credentials and claims are encapsulated in every message by using the WS-Security specification to secure messages. This option gives the most flexibility from an authentication perspective. You can use any

type of authentication credentials you want, largely independent of transport, as long as both client and service agree.

- **Transport security**. When using transport security, the user credentials and claims are passed by using the transport layer. In other words, user credentials are transport-dependent, which allows fewer authentication options compared to message security.
- **Mixed security**. Mixed security gives you the best of both worlds: transport security ensures the integrity and confidentiality of the messages, while the user credentials and claims are encapsulated in every message as in message security. This allows you to use a variety of user credentials that are not possible with strict transport security mechanisms, and to leverage transport security's performance.
- **Both security**. When using this option, the user credentials and claims are transferred at both the transport layer and message level. Similarly, message protection is provided at both the transport layer and message level. Note that this is not a common scenario, and only bindings that support the Microsoft Message Queuing (MSMQ) protocol support this security mode.

Your choice of binding will also affect your authentication options, as not all the transport or message user credentials are supported in every binding.

Authentication Options with Transport Security

The follow authentication options are available when using transport security:

- **None**. When using this option, the WCF service does not authenticate the callers. This is not the recommended option from a security perspective – avoid using this option wherever possible.
- **Basic**. This option is available with the HTTP protocol only. The client is authenticated by using the username and password against Active Directory. The client credentials are transported by using a Base64 encode string, which is very similar to a clear string and therefore not the most secure option. The service is authenticated by the Secure Sockets Layer (SSL) certificate used for secure communication.
- **NTLM**. This option is available with the HTTP protocol only. The client is authenticated by using a challenge-response scheme against Windows accounts. NTLM authentication is well suited for a workgroup environment and is more secure than Basic authentication. The service is authenticated by using an SSL certificate.
- **Windows**. When using this option, the WCF service uses Kerberos authentication when in a domain, or NTLM authentication when deployed in a workgroup environment. This option uses a Windows token presented by the caller to authenticate against the Active Directory. This is the most secure option compared to Basic or NTLM authentication. The service is authenticated by using the Windows credentials of the process identity, or an SSL certificate if you are using the HTTP protocol.
- **Certificate**. When using this option, the caller presents an X.509 client certificate that the WCF service validates by trusting the certificate (peer trust) or trusting the issuer of the certificate (chain trust). This option should be used when Windows authentication is not

possible, as in the case of business-to-business (B2B) scenarios. The service is authenticated with the service certificate, or by using an SSL certificate if you are using the HTTP protocol.

Authentication Options with Message Security

The following authentication options are available when using message security:

- **None**. When using this option, the WCF service does not authenticate the callers. This is not the recommended option from a security perspective – avoid using this option wherever possible.
- **Windows**. When using this option, the WCF service uses Kerberos authentication when in a domain, or NTLM authentication when deployed in a workgroup environment. This option uses the Windows token presented by the caller to authenticate against the Active Directory. Service is authenticated by using the Windows credentials of the process identity.
- **Username**. When using this option, the caller provides a username and password to the service. The service can either authenticate against Windows credentials, use a membership provider such as the SQL Server membership provider, or use a custom validator to validate against the custom store. You should choose this option only when Windows authentication is not possible. The service is authenticated by using a service certificate.
- **Certificate**. When using this option, the caller presents an X.509 client certificate. The WCF service looks up the certificate information on the host side and validates it (peer trust), or trusts the issuer of the client certificate (chain trust). This option should be used when Windows authentication is not possible, or in the case of B2B scenarios. The service is authenticated by using a service certificate.
- **Issue token**. When using this option, the client and service depend on the Secure Token Service (STS) to issue tokens that the client and service trusts. CardSpace is a typical example of an STS.

Authorization Options in WCF

WCF supports three basic authorization approaches:

- **Role-based**. Access to WCF operations is secured based on the role membership of the caller. Roles are used to partition your application's user base into sets of users that share the same security privileges within the application; for example, Senior Managers, Managers, and Employees. Users are mapped to roles, and if the user is authorized to perform the requested operation, the application executes the operation.
- **Identity-based**. WCF supports an Identity Model feature, which is an extension of role-based authorization. Identity Model enables you to manage claims and policies in order to authorize clients. With this approach, you can verify claims contained within the authenticated users' credentials. These claims can be compared with the set of authorization policies for the WCF service. Depending on the claims provided by the client, the service can either grant or deny access to the operation or resources. Identity Model is useful for fine-grained authorization and is most beneficial when using issue token authentication.

- **Resource-based**. With this option, individual resources are secured by using Windows access control lists (ACLs). The WCF service impersonates the caller prior to accessing resources, which allows the operating system to perform standard access checks. All resource access is performed using the original caller's security context. This authorization approach severely impacts application scalability, because it means that connection pooling cannot be used effectively within the application's middle tier.

In enterprise-level applications where scalability is essential, a role-based or identity-based approach for authorization is the best choice. For small-scale intranet applications that serve per-user content from resources (such as files) that can be secured with Windows ACLs, a resource-based approach may be appropriate.

Role-based Authorization Options in WCF

WCF rovides the following options for role-based authorization:

- **Windows groups**. For WCF services and clients that are deployed in the same Windows domain, you can use Windows groups for role authorization. Typically, this option is suitable when you have a small number of coarse role requirements.
- **ASP.NET roles**. Use ASPNET roles if you have fine-grained roles requirements, or if the users cannot be mapped to Windows domain accounts. This option uses the Role Manager feature and provides three different role providers, based on the role store:
    - **SqlRoleProvider**. If your role information is stored in a SQL Server database, consider using SqlRoleProvider for roles authorization.
    - **WindowsTokenRoleProvider.** If your roles are Window groups, and you want to leverage the Role Manager feature as a consistent way of checking the role membership of your users, regardless of the underlying data store, consider using WindowsTokenRoleProvider for roles authorization.
    - **AuthorizationStoreRoleProvider**. If your role information is stored using an AzMan policy store in an XML file, in Active Directory, or in Active Directory Application Mode (ADAM), consider using AuthorizationStoreRoleProvider for role authorization.
- **Custom roles**. If your role information is stored in a custom store such as a SQL Server database, create a custom authorization policy to authorize your users.

Note: It is recommended that you implement a custom role provider, using the Role Manager feature, for your custom role store rather than using the custom roles option.

Imperative Authorization

Imperative authorization supports fine-grained authorization choices based on business logic. Imperative role-based authorization is written into your code and processed at run time. Imperative security is useful when the resource to be accessed or action to be performed is not known until run time, or when you require finer-grained access control beyond the level of a code method.

Imperative Check Example

The following is an example of an Imperative check using the ASP.NET role provider:

```
if (Roles.IsUserInRole(@"accounting"))
{
  //authorized
}
else
{
  //authorization failed
}
```

Declarative Authorization

Declarative authorization can be added to application code at design time by specifying required access for a particular method or class declared as an attribute on the operation. Declarative role-based authorization is best for authorizing access to WCF at the operation level. Because attribute metadata is discoverable using reflection, it is easier to track the security principals that are allowed to access each method. Declarative authorization checks will work if you are using the ASP.NET role provider or Windows groups.

PrincipalPermission Example

The following code example shows how to use the **PrinciplePermission** attribute to perform declarative authorization:

```
[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]
public double Add(double a, double b)
{
 return a + b;
}
```

Identity-based Authorization Options in WCF

WCF uses the Identity Model feature to create claims from incoming messages. Identity Model classes can be extended to support new claim types for custom authorization schemes. Identity Model and claim-based authorization are outside the scope of this guide.

For more information, see “Managing Claims and Authorization with the Identity Model” at http://msdn.microsoft.com/en-us/library/ms729851.aspx.

Resource-based Authorization Options in WCF

A resource-based approach tends to work best for applications that provide access to resources that can be individually secured with Windows ACLs, such as files and other local resourcesThe approach starts to break down in cases where the requested resource consists of data that needs to be obtained and consolidated from a number of different sources; for example, multiple databases, database tables, external applications, or Web services. If the resources are remote, you have to use delegation.

The resource-based approach relies on the original caller's security context flowing through the application to the back-end resource managers. This can require complex configuration and significantly reduces the ability of a multi-tiered application to scale to large numbers of users, because it prevents the efficient use of pooling (for example, database connection pooling) within the application's middle tier.

The two most commonly used resource-based security models are:

- The trusted subsystem model
- The impersonation/delegation model

The Trusted Subsystem Model

With this model, the WCF service uses a fixed identity to access downstream services and resources.

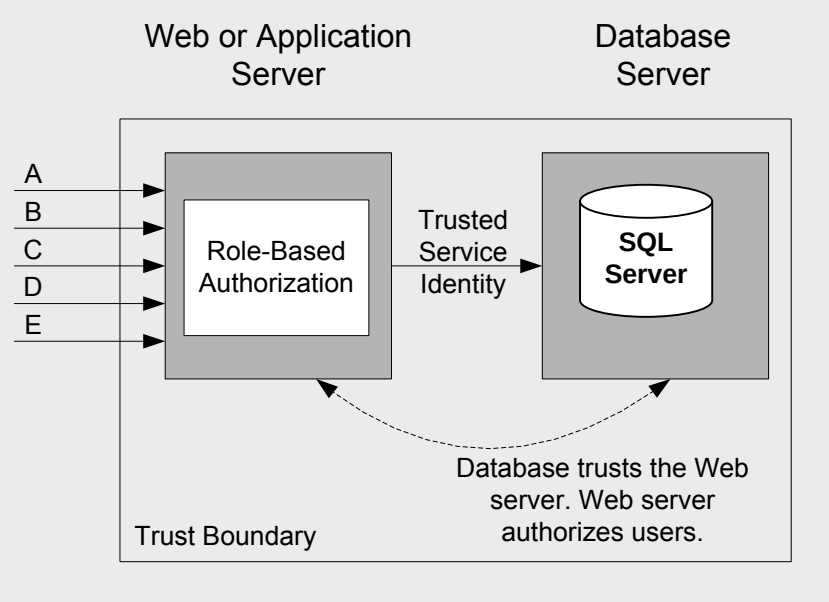


Figure 1. Trusted-Subsystem Model

A variation on this model may use a fixed identity per role in order to map multiple users to a set of permissions on a downstream resource.

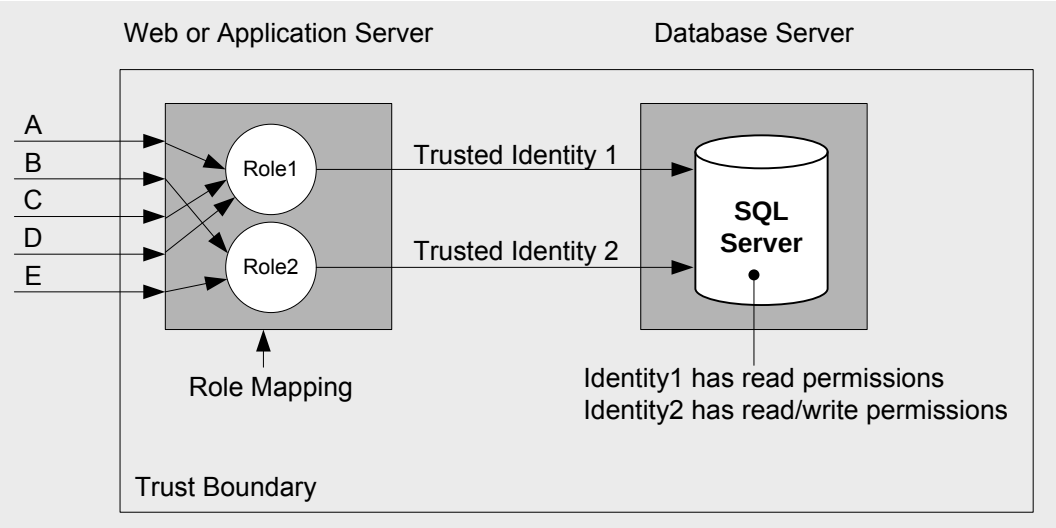


Figure 2. Trusted Subsystem Model with Fixed Role Identity

The security context of the original caller will not flow through the service at the operating-system level**.** You can flow the identity of the caller at the application level by passing it to the downstream resource manually for auditing purposes. For example, you can pass it as a string in the header of the message or a s a parameter in a stored procedure, etc.

It may need to do so in order to support back-end auditing requirements, or to support per-user data access and authorization.

The pattern for resource access in the trusted subsystem model is as follows:

1. Authenticate users.
2. Map users to roles.
3. Authorize based on role membership.
4. Access the downstream resource manager using a single or multiple fixed trusted identities.

The Impersonation / Delegation Model

With this model, the WCF service impersonates the client's identity before it accesses the next downstream service.

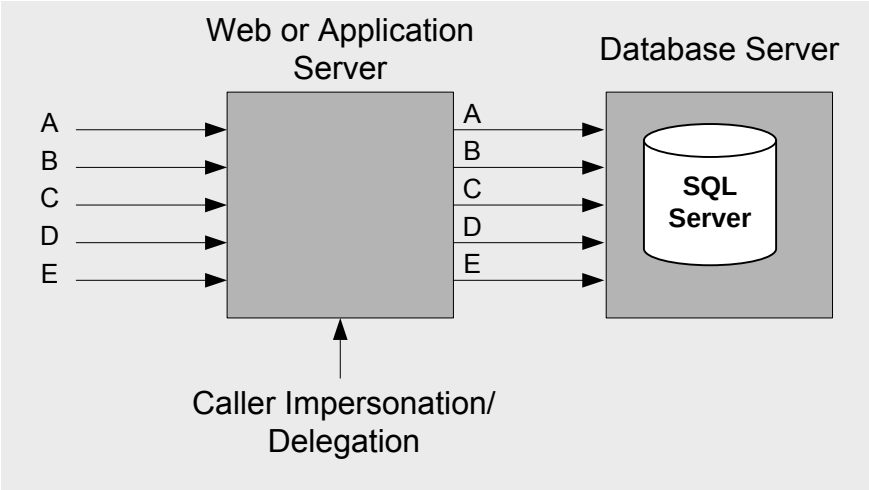


Figure 3. Impersonation/Delegation Model

If the next service in line is on the same computer, impersonation is sufficient. Delegation is required if the downstream service is located on a remote computer.

> The delegation mechanism is a powerful feature. Use the constrained delegation feature available in Windows 2003 when possible. With it, you can limit access to a particular service on a specific machine.

As a result of the delegation, the security context used for the downstream resource access is that of the client. This model is typically used for one of the following two reasons:

- It allows the downstream service to perform per-caller authorization using the original caller's identity.
- It allows the downstream service to use operating system–level auditing features.

For more information about the impersonation options, see "Chapter 6 – Impersonation and Delegation in WCF."

Identities in WCF

When designing your authentication and authorization strategies, you need to consider the following identity types:

- **Process identity**. This is the identity of the process hosting the WCF service. When the WCF service is hosted in Internet Information Services (IIS), it typically is **NETWORK SERVICE** by default.
  This means that the machine account credentials of the service host are presented to downstream resources. The process identity is important because it identifies what

Windows resources and back-end the service can access, when the WCF service is not impersonating the original caller. If a certificate is used to protect the transport, the process identity also needs access to the certificate's private keys in order to provide for message security or transport security with **netTcpBinding**.

- **Security principal**. The executing thread includes a security principal that contains the user identity and associated roles. The roles can be Windows roles if the principal is a Windows **Principal**; an ASP.NET role if it is a role **Principal**; or a custom role if it is a generic **Principal**. To be able to authorize – either with the **Roles.IsinRole** call, with **IPrincipals.IsInRole**, or with declarative authorization checks – a security principal must be present in the thread executing the WCF business logic. If a custom authentication is used in WCF, the security principals must be set in a class that derives from **IAuthorizationPolicy**, and this custom authorization policy must be configured in WCF.
- **ServiceSecurityContext**. This identity type, available in the WCF run time, contains all of the security-related objects available in the WCF context. These objects are the user identity and authorization context and polices. The service security context is available on both the service and the client side. In the authorization context, you can extract the claim set associated with a security token, whether it is a certificate, issue token, username, or Windows token. To get the service security context on the client side, you need to use the operation context instead.

Design Strategy for Authentication and Authorization

The following steps help you to develop an authentication and authorization strategy for your application:

- **Step 1 – Identify resources**
- **Step 2 – Choose an authorization approach**
- **Step 3 – Choose the identities to be used for resource access**
- **Step 4 – Choose an authentication approach**

Step 1 – Identify Resources

Identify resources that your application needs to expose to clients.

Typical resources include:

- Server resources such as contracts, service metadata, and static resources
- Database resources such as per-user data or application-wide data
- Network resources such as remote file system resources and data from directory stores such as Active Directory

You must also identify the system resources your application needs to access. This is in contrast to resources that are exposed to clients. Examples of system resources include the registry, event logs, and configuration files.

Step 2 – Choose an Authorization Approach

When choosing an authorization strategy, first decide if you want to use resource-based or role-based authorization. Next, consider what user store your users belong to.

Resource-based authorization uses ACLs on the resource to authorize the original caller. Role-based authorization allows you to authorize access to service operations or resources based on the group to which user belongs.

If scalability is essential to your application, consider a role-based approach. The following are the most common scenarios for role-based authorization:

- **User store is in Windows Active Directory**. Consider using role-based authorization with Windows groups for ease of maintenance as both roles and credentials reside in the Active Directory store.
    - If your roles are Windows groups and you want to leverage the Role Manager feature as a consistent way of checking your users' role membership, consider using **WindowsTokenRoleProvider** for role authorization.
    - If your application requires finely granular roles, using Windows groups might not be the best option. Instead, consider using ASP.NET roles with role providers such as **SqlRoleProvider** or **AuthorizationStoreRoleProvider**.
- **User store is in a SQL Server database.** Consider using role-based authorization with ASP.NET roles.
    - If your role information is stored in a SQL Server database along with the user information, consider using ASP.NET roles with SqlRoleProvider.
    - If your role information is stored in a custom store, consider using custom role authorization.
- **User information is stored in certificates**. Consider doing authorization by using the claims provided by the certificate, such as subject name, thumbprint, etc.
    - If the user certificates can be mapped to a Windows account, consider using Windows groups for role authorization.
    - If your certificates can be mapped to a Windows account, and if your application requires finely granular roles, consider using ASP.NET roles with role providers such as SqlRoleProvider or AuthorizationStoreRoleProvider.
- **User information is in a custom store**. Consider using custom roles for authorization by creating a custom authorization policy.

Consider using a resource-based approach for smaller-scale intranet applications that serve per-user content from resources (such as files) that can be secured with Windows ACLs.

You must also consider whether you want to use impersonation and delegation:

- If your technical requirement is to use the original caller's identity to access back-end resources on the same computer that is running the service, use Impersonation.
- If your technical requirement is to use the original caller's identity to access back-end resources on computers other than the computer running the service, use delegation.

Step 3 – Choose the Identities to Be Used for Resource Access

Choose the identity or identities that should be used to access resources across the various tiers of your application. The identities can be:

- **Original caller's identity**. This assumes an impersonation/delegation model in which the original caller's identity can be obtained and then flowed through each layer of your system. The delegation factor is a key criterion used to determine your authentication mechanism.
- **Process identity**. This is the default case (without specific impersonation). Local resource access and downstream calls are made using the current process identity. The feasibility of this approach depends on the boundary being crossed, because the process identity must be recognized by the target system.
  This implies that calls are made in one of the following ways:
  - Within the same Windows security domain
  - Across Windows security domains (using trust and domain accounts, or duplicated user names and passwords where no trust relationship exists)
- **Service account**. This approach uses a (fixed) service account. For example, for database access, this might be a fixed SQL user name and password presented by the component connecting to the database.

Step 4 – Choose an Authentication Approach

The choice of an authentication approach is influenced by the nature of your application's user base, and impersonation/delegation and auditing requirements. The available authentication options depend on both the security mode and the binding.

An authentication strategy is based on available client credentials, networking strategy, system maintenance, and security-level requirements. You can decide on an appropriate authentication strategy by considering the following factors.

Transfer Security

Because WCF authentication options depend on the transfer security mode being used, you should first select the appropriate transfer security mode for your WCF application.

WCF offers two security modes: transport and message**.** If you are using transport security, you cannot use negotiate, username, or Kerberos direct authentication. If you are using message security, you cannot use Basic or Digest authentication.

Transport Security

Use the following criteria to determine whether you should use transport security:

- **Point-to-point.** Transport security supports point-to-point communication and does not support intermediary scenarios or protocol transition.
- **Streaming.** Transport security can support streaming data scenarios.
- **Binding limitations.** Transport security does not work with the **wsDualHttpBinding** binding.

- **Authentication limitations.** Transport security does not work with negotiation, username, or Kerberos direct authentication.

Message Security

Use the following criteria to determine whether you should use message security:

- **Intermediaries.** Message security supports scenarios with intermediaries or protocol transition.
- **Encryption flexibility.** Message security allows you to encrypt part of a message while leaving other parts in clear text format.
- **Binding limitations.** Message security does not work with the **netNamedPipeBinding** binding.
- **Secure conversations.** Secure conversations only work with message security.
- **Authentication limitations.** Message security does not work with Basic or Digest authentication.

Bindings

The choice of binding also plays an important role in your authentication options because not all transport or message security authentication options are supported across all bindings.

The following bindings typically work well over the Internet:

- If your service is interacting with WCF clients, use **wsHttpBinding** because it provides the best WS-* interoperability features, including WS-SecureConversation, WS-Addressing, and WS-AtomicTransaction. The combination of features offered by **wsHttpBinding** makes for the most reliable connection offered by WCF over the Internet.
- If your service is interacting with ASP.NET Web Services (ASMX) clients, you must use **basicHttpBinding** because it is the only WCF binding that supports ASMX clients.
- Clients and services that require full-duplex communication should use **wsDualHttpBinding** because it is the only binding that supports full-duplex.
- If your service is interacting with WSE clients, you must use **customBinding**. The service must use a custom binding to be compatible with the August 2004 version of the WS-Addressing specification.

Most of the bindings also work in an intranet scenario, but **netTcpBinding** provides the best throughput performance. On an intranet, you generally do not need to worry as much about the connection going down as with an Internet connection, so some of the WS-* advantages that are supplied with **wsHttpBinding** may not be as necessary on an intranet.

Bindings Summary

The following table resents a list of bindings and details each binding's support for transfer security modes.

Name	None	Transport	Message	Mixed	Both
basicHttpBinding	√ (Default)	√	√	√	X
netTCPBinding	√	√ (Default)	√	√	X
netPeerTCPBinding	√	√ (Default)	√	√	X
netNamedPipeBinding	√	√ (Default)	X	X	X
wsHttpBinding / ws2007HttpBinding	√	√	√ (Default)	√	X
wsFederationHttpBinding /wsfederationHttpBinding	√	X	√ (Default)	X	X
wsDualHttpBinding	√	X	√ (Default)	X	X
netMsmqBinding	√	√ (Default)	√	X	√

User Store and Credential Management

Your authentication choice will depend greatly on whether you already have a user store in place:

- **User store is in Windows Active Directory**. Consider using Windows authentication for ease of maintenance as both roles and credentials reside in the Active Directory store. One of the key advantages of Windows authentication is that it enables you to let the operating system take care of credential management. By using Windows authentication with Active Directory, you benefit from a unified identity store, centralized account administration, enforceable account and password policies, and strong authentication that avoids sending passwords over the network.
- **User store is in a SQL server database**. Consider using username authentication and configure your application to use the ASP.NET membership feature. You can use username authentication when you need client username/password authentication (vs. client computer-level authentication) and the client and service are not in trusted domains; for example, clients are connecting over the Internet.
- **User information is stored in certificates**. Consider using certificate authentication, which works best when you need computer-level authentication vs. user-level authentication. Client certificates are used to authenticate the client computer connection, but not at a user level. Server certificates are used when you need to authenticate the server credentials to client connections.
- **User information is in a custom store**. Consider using custom roles for authorization by creating a custom authorization policy.

The following are some other factors to consider when choosing an authentication strategy:

- Determine which client credentials are available and map the credentials to the available WCF authentication methods.
- Determine if the client-service communication is within an intranet only or across the Internet. Integrated Windows authentication does not map well to Internet authentication scenarios.
- Consider total cost of ownership (TCO) when choosing an authentication scheme. Client certificates mapped to each user will be the most time-consuming to maintain. Windows accounts are easiest to maintain because they are built into the operating system, with maintenance tools and security supplied and tested by default.
- Consider whether you need user-level or machine-level authentication. Certificates work well for machine-level authentication.

Key Authentication and Authorization Scenarios

The following scenarios help illustrate best practices for authentication and authorization choices in common WCF deployment scenarios. Use these scenarios to help you choose your authentication and authorization strategy based on your user credential store location and the location of your clients on an intranet or the Internet.

Intranet Scenarios

The most common authentication scenario for intranet applications is where one or more client computers connect to a WCF service. Here the service acts as an intermediary to supply data back to the clients from resources to which it has access to resources on other computers.

A basic assumption for an intranet scenario is that your users have Windows accounts in the server's domain or in a trusted domain accessible by the server.

Authentication

Use Windows authentication when both the client and service are in trusted domains, such as in an intranet scenario. Windows authentication works with resource-based authorization by default.

Authorization

In an intranet scenario, if you are using Windows Active Directory for authentication, use Windows groups for role authorization in WCF. You can choose this option when you have very coarse or fewer role requirements.

If you have fine-grained role requirements, consider using ASP.NET roles. If your roles are Windows groups, consider using the **WindowsTokenRoleProvider** for role authorization.

Internet Scenarios

The most common authentication scenario for Internet applications is where one or more client computers connect to a WCF service. Here the service acts as an intermediary to supply data

back to the clients from resources to which it has access from resources on other computers within its network.

The basic assumptions for Internet scenarios are:

- Your users do not have Windows accounts in the server's domain or in a trusted domain accessible by the server.
- Your users have Windows accounts, but cannot access directly over the Internet.

If you want to access downstream services and resources on the Internet, use the Trusted Subsystem model.

Authentication

Keep in mind the following considerations related to authentication types:

- **Username authentication**. Use username authentication in the following scenarios:
    - If your users are in a custom store, use username authentication with a custom validator, or use username authentication with a membership provider by implementing a custom membership provider against the custom store.
    - If your users are in a SQL Server database, use username authentication with SqlMembershipProvider.
    - If your users are in Active Directory, use username authentication mapped to Windows.
- **Certificate authentication**. Use certificate authentication in the following scenarios.
    - If your service needs to be consumed by partner (B2B) applications. In this scenario, the client certificates can authenticate a machine account or multiple users to a WCF service.
    - If you have initial certificate investment.
    - If your users are in Active Directory, but you cannot use Windows authentication, use certificate authentication and map the certificates to a Windows account.

Authorization

You can implement role authorization either by using declarative or imperative authorization as follows:

- If you are using username authentication with **SqlMembershipProvider**, use **SqlRoleProvider** for role authorization.
- If you are using username authentication mapped to Windows, use **WindowsTokeRoleProvider** for role authorization using Windows groups.
- If you are using username authentication mapped to Windows, an AzMan policy store in an XML file, in Active Directory, or in Active Directory Application Mode (ADAM), consider using **AuthorizationStoreRoleProvider** for role authorization.
- If you are using certificate authentication with certificates mapped to Windows accounts, use **WindowsTokeRoleProvider** for role authorization using Windows groups.

- If you are using certificate authentication with certificates mapped to Windows accounts, an AzMan policy store in an XML file, in Active Directory, or in Active Directory Application Mode (ADAM), consider using **AuthorizationStoreRoleProvider** for role authorization.

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For more information on authentication, see “Authentication” at http://msdn.microsoft.com/en-us/library/ms733082.aspx
- For more information see the list of related How To articles at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=How%20Tos&referringTitle=Home
- For more general information, see “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Chapter 6 – Impersonation and Delegation in WCF

Objectives

- Understand the different WCF authentication security options: message security, transport security, mixed, and both.
- Understand the different authentication methods that are available when using these options.
- Understand the WCF authorization options based on role, identity, and resource.
- Understand the various WCF identities and the meaning of *process identity*, *security principle*, and ServiceSecurityContext.
- Understand common implementation models that use specific authentication, authorization, and identity choices.

Overview

Impersonation is a common technique that WCF services use to assume the original caller's identity in order to authorize access to service resources (such as files or database tables). Service resources can be resources that are either local to the service machine or remotely hosted. Impersonation is used to access resources on the same machine as the service, while delegation is used to access resources that are remotely hosted.

By default, impersonation is disabled and resources are accessed by using the WCF service's process identity. Impersonation allows you to access local resources and perform other operations using the authenticated user's identity or a specific Windows identity. You can enable impersonation either programmatically or by applying appropriate attributes at operation or service levels.

You can impersonate imperatively or declaratively. *Imperative impersonation* is performed programmatically at run time and can vary depending on business logic or other conditions. *Declarative impersonation* is applied with a static attribute that can be associated with an operation or an entire interface. In general, you should use imperative impersonation when you need the fine granularity made possible by writing the impersonation logic into your code. If you do not need such fine granularity, you can use declarative impersonation.

Delegation allows you to use an impersonation token to access network resources. Your ability to use delegation depends on the authentication mechanism in use and appropriate account configuration.

Impersonation Scenarios

The most common impersonation and delegation scenarios are:

- **Impersonate the original caller**. You want to access Windows resources that are protected with access control lists (ACLs) configured for your application's domain user accounts.
- **Impersonate the original caller temporarily**. You want to access resources predominantly by using the WCF service's process identity, but specific methods need to use the original caller's identity.
- **Impersonate a specific Windows identity**. You need to use a specific identity or several Windows identities to access particular resources.
- **Use delegation to access network resources**. You need to use an impersonated identity to access remote resources.

Impersonate the Original Caller

You should impersonate the original caller if you need to control access to Windows resources that are protected by ACLs that apply to the original callers of your service.

To impersonate the original caller for a specific operation or entire service, you need to use the declarative impersonation, for which you need to enable Windows authentication on the WCF service.

If you need to access specific resources such as local files by using the process identity, you can temporarily remove the impersonation token from the WCF request thread.

Impersonate the Original Caller Temporarily

If you need access to specific resources inside the scope of a service operation, you can use programmatic impersonation to limit the amount of time your service runs in the higher-privileged context of the original caller. There are three options for temporarily impersonating the original caller:

- **Use Windows authentication**. To temporarily impersonate the original caller within a particular operation, you need to obtain the **WindowsIdentity** object that represents the authenticated user, and then call its **Impersonate** method. To use this option, you need to enable Windows authentication on the WCF service.
- **Use the WindowsIdentity constructor (S4U Kerberos extensions)** Use this option if your users have Windows domain accounts, but you are using non-Windows authentication, such as certificate authentication. To impersonate the caller in this instance, you must programmatically create a **WindowsIdentity** object for the caller, which you can use to impersonate. Create a **WindowsIdentity** object by using the **WindowsIdentity(userPrincipalName)** constructor that takes a single parameter of a user principal name (UPN). With this approach, you do not need the account's password. See later sections in this chapter for information on S4U Kerberos.
- **Use the LogonUser API.** Use this option if your users have Windows domain accounts, but you are using non-Windows authentication, such as custom authentication. To impersonate the caller in this instance, you must programmatically create a **WindowsIdentity** object for

the caller, which you can use to impersonate. Create a **WindowsIdentity** object by using a logon token returned from the Win32 LogonUser API.

Impersonate a Specific Windows Identity

If you need to access local resources (such as the file system or a local database) while assuming the identity of the specific Windows account for the entire duration of the operation, configure the WCF service to run under that Windows identity.

If you need to use the process identity for most resource access, and then impersonate the specific Windows identity to perform specific operations or access specific resources, you can use programmatic impersonation. There are two options for impersonating a specific Windows identity temporarily:

- **Use the WindowsIdentity constructor (S4U Kerberos extensions).** To impersonate the specific Windows identity, create a **WindowsIdentity** object by using the **WindowsIdentity(userPrincipalName)** constructor that takes a single parameter of a UPN. With this approach, you do not need the account's password. See Additional Resources section in this chapter for S4U Kerberos information.
- **Use the LogonUser API.** To impersonate the specific Windows identity, create a **WindowsIdentity** object by using a logon token returned from the Win32 LogonUser API.

Use Delegation to Access Network Resources

If your service needs to access remote / network resources, you can access the resources on behalf of the original caller or a fixed identity in the following ways:

- **Use Kerberos authentication and delegation**. If you use Kerberos to authenticate your users, you can impersonate the original caller by using the techniques described in the sections "Impersonating the Original Caller" and "Impersonating the Original Caller Temporarily," and then use Kerberos delegation to gain access to network resources as follows:
  - If your WCF service runs under the Network Service account, configure your computer account in Active Directory to be trusted for delegation.
  - If your application runs under a custom domain account, you must register a service principal name (SPN) in Active Directory to associate the domain account with the HTTP service on your WCF server. You then configure your domain account in Active Directory to be trusted for delegation.

- **Use protocol transition**. With this approach, you use a non-Kerberos authentication mechanism, such as client certificates, to authenticate your users, and then use the new **WindowsIdentity** constructor to obtain a Windows token for the user on the server. Use

this approach when you cannot use Kerberos authentication to authenticate your users. Keep in mind the following considerations:
  - If your WCF service runs under the Network Service account, configure your computer account in Active Directory to be trusted for delegation and protocol transition.
  - If your application runs under a custom domain account, you must register an SPN in Active Directory to associate the domain account with the HTTP service on your WCF server. You then configure your domain account in Active Directory to be trusted for delegation and protocol transition.

- **Call LogonUser and request an Interactive logon session**. An interactive logon session has network credentials that allow you to authenticate against network servers. Use this approach when you cannot use Kerberos authentication to authenticate your users, and when you cannot use protocol transition.

Note that you must have access to both the username and password to call LogonUser. You can only use the token to access network resources over a single hop, whereas Kerberos delegation allows the impersonated identity to flow across multiple tiers.

Impersonation Options

There are three options available for impersonation:

- Impersonate using the **WindowsIdentity** token with Windows authentication.
- Impersonate using the **WindowsIdentity** constructor (S4U Kerberos extensions).
- Impersonate using the LogonUser API.

Impersonate Using the WindowsIdentity Token With Windows Authentication

With this option, you impersonate using the Windows token, obtained from the Security Support Provider Interface (SSPI) or Kerberos authentication, which is then cached on the service.

You can use this option when using Windows authentication or username authentication where users can be mapped to valid Windows accounts. This impersonation option supports programmatic and declarative impersonation in WCF.

Impersonate Using the WindowsIdentity Constructor (S4U Kerberos Extensions)

You can use this option when your clients are authenticated by using non-Windows authentication such as client certificates, which supports mapping to Windows accounts, or when you want to impersonate a Windows service account. This impersonation option supports programmatic impersonation in WCF.

With this option, you need to create a **WindowsIdentity** object by using the **WindowsIdentity(userPrincipalName)** constructor that takes a single parameter of a UPN. With this approach, you do not need the account's password.

The **WindowsIdentity** constructor relies on a Windows Server 2003 extension to the Kerberos protocol known as Service for User to Self (S4U2Self). You can use this approach if your application runs on Windows Server 2003 in a Windows Server 2003 domain. The advantage of this approach is that you do not have to store credentials as you do for LogonUser. However, the disadvantage is that if your code needs to access local resources, you must grant the "Act as part of the operating system" privilege to your Web application process account to get an impersonation-level token.

Token Types

The type of token generated by the S4U2Self extension determines what you can do with the token while impersonating. You can obtain the following token types:

- **Identify-level token** – This is returned by default. With this type of token, you can check to see what groups are contained in the token, but you cannot use it as an impersonation token to access local or remote resources.
- **Impersonation-level token** – If you grant your process account the "Act as part of the operating system" user right, you get this type of token from the **WindowsIdentity** constructor. With this type of token, you can impersonate and access local resources. **Note:** This places your process within the trusted computing base (TCB) of the WCF server, which makes your WCF service process very highly privileged. Where possible, you should avoid this approach because an attacker who manages to inject code and compromise your WCF application will have almost unrestricted capabilities on the local computer.
- **Delegate-level token** – If you configure your service or machine account in Active Directory to be trusted for constrained delegation and protocol transition, you will get a token that you can use to access network resources.

Impersonate Using the LogonUser API

You can use this option when you want to access network resources (delegation) but you do not have the "trusted for delegation" permission. You can also use this option if you want to access local resources but you do not want to grant higher privileges to your WCF process identity. This places the responsibility for maintaining the user credentials on the WCF service. This impersonation option supports programmatic impersonation in WCF.

You can create a Windows token and associated logon session for a domain or local account by using the Win32 LogonUser API. You must pass the user name and password to this API, together with other parameters including the type of logon session you require.

Note: You should protect the credentials passed to LogonUser by encrypting them.

Whether you can access local resources or network resources depends on the logon session type that you request. (You specify the logon session type in the third argument of LogonUser.) The most commonly used logon session types when calling this API are the following:

- **Interactive logon** – If you need to access remote resources, request an interactive logon session. This results in a logon session that has network credentials. The user account passed to LogonUser must be granted the "Log on locally" user right.
- **Network logon** – This establishes a logon session with no network credentials. This means you can impersonate the token and access local resources only. The user account passed to logon user must be granted the "Access this computer from the network" user right. By default, all accounts have this right because it is granted to the Everyone group.

Impersonation Methods

There are three methods used for impersonation:

- Impersonate the original caller declaratively on specific operations.
- Impersonate the original caller declaratively for the entire service.
- Impersonate the original caller programmatically within an operation.

Impersonate the Original Caller Declaratively on Specific Operations

You can impersonate declaratively on an operation when you want to impersonate the original caller for the entire duration of a specific operation. Use impersonation selectively and only on the operations that need it, since by nature it increases the potential attack surface of your application.

You can impersonate declaratively by applying the **OperationBehavior** attribute on any operation that requires client impersonation. The following example shows how to impersonate for a specific operation.

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
   return "test";
}
```

Impersonate the Original Caller Declaratively for the Entire Service

Impersonate declaratively on a service when you want to impersonate the original caller for all of the operations in your service. However, you should be careful with this option because it can significantly increase the attack surface of your application by running all of your code under a higher-privileged account.

You can impersonate the entire service by setting the **impersonateCallerForAllOperations** attribute to "true" in the WCF configuration file. The following example shows how to impersonate for entire service:

```
...
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceAuthorization impersonateCallerForAllOperations="true" />
    </behavior>
  </serviceBehaviors>
</behaviors>
...
```

If you are impersonating all operations in the service, the **Impersonation** property of the **OperationBehaviorAttribute** applied to each operation will be overriden. Therefore if the property on the operation is set to something other than **Allowed** or **Required**, impersonation will be turned off for that operation.

Note: When a service has higher credentials than the remote client, the credentials of the service are used if the **Impersonation** property is set to **Allowed**. That is, if a low-privileged user provides its credentials, a higher-privileged service executes the method with the credentials of the service, and can use resources that the low-privileged user would otherwise not be able to use.

Impersonate the Original Caller Programmatically Within an Operation

Impersonate programmatically when you want to impersonate the original caller for a short duration in a service operation. You can impersonate programmatically by calling the **Impersonate**() method on the Windows identity that you want to impersonate, as follows:

```
public string GetData(int value)
{
 using (ServiceSecurityContext.Current.WindowsIdentity.Impersonate())
 {
  // return the impersonated user (original users identity)
  return string.Format("Hi, {0}, you have entered: {1}",
  WindowsIdentity.GetCurrent().Name, value);
 }
}
```

Note: It is important to revert to impersonation. Failure to do so can form the basis for denial of service (DoS) or elevation of privilege attacks. In the example above, the **using** statement ensures that the impersonation is reverted after execution of the **using** block.

Controlling Impersonation on the Service Side

You can control impersonation on the service side by using declarative impersonation. You can use the **ImpersonationOption** enumeration along with the **OperationBehaviorAttribute** attribute to control impersonation. The following impersonation options are available:

- **NotAllowed.** Impersonation is not performed in a particular operation.
- **Allowed.** Impersonation is performed if the original Windows token is available and the service is configured to impersonate on all operations using the **ImpersonateCallerForAllOperations** in the **ServiceAuthorizationBehavior** attribute.
- **Required.** Impersonation is performed; the Windows identity token is required to be available.

The following example uses declarative impersonation to control impersonation on the service side:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
   return “test”;
}
```

Controlling Impersonation on the Client Side

You can control impersonation on the client side and prevent WCF services from using client identities to access local resources. Windows credentials have an **AllowedImpersonationLevel** property that is set to one of the following **TokenImpersonationLevel** options in order to control the impersonation level:

- **None.** The WCF service cannot authenticate or impersonate the user.
- **Anonymous.** The WCF service authenticates clients as anonymous, but cannot obtain information for identification or impersonation.
- **Identification.** The WCF service can authenticate clients and get information for identification, but cannot impersonate the clients. This is the default value.
- **Impersonation.** The WCF service can authenticate, get information for identification, and impersonate clients on local systems.
- **Delegation.** The WCF service can authenticate, get information for identification, and impersonate clients on local as well as remote systems.

The following example shows how to configure the impersonation level on the client side:

```
<behaviors>
    <endpointBehaviors>
        <behavior name="NewBehavior">
            <clientCredentials>
```

```
                <windows allowedImpersonationLevel="Impersonation" />
            </clientCredentials>
        </behavior>
    </endpointBehaviors>
</behaviors>
```

Note: The impersonation level obtained by the server when it impersonates the client token is not solely a function of this setting. It is also a function of the associated privileges and domain settings for the account in which the service is running.

Pitfalls and Issues with Impersonation

Impersonation and delegation are powerful tools that should be used cautiously and selectively. Improper use of impersonation and delegation can easily lead to serious security vulnerabilities. The following are examples of common mistakes to avoid:

- **Using impersonation by default**. Only use impersonation when you need to. Using impersonation can introduce security vulnerabilities, particularly in multi-threaded applications. The use of impersonation prevents the efficient use of connection pooling if you are accessing downstream databases using the impersonated identity. Be diligent when choosing to impersonate and think about other alternatives; for example, if downstream auditing is your goal, you might choose to pass the user identity as part of the method calls rather than using delegation.
- **Using callbacks when impersonating.** Avoid using callbacks when impersonating, as you do not have control over what code will be executed under the impersonated identity.
- **Failure to revert impersonation.** When using programmatic impersonation, be sure to explicitly revert the impersonation. If you do not remember to revert, your application's attack surface will be increased because it will continue to run under higher privileges.
- **Using the WindowsIdentity constructor (S4U Kerberos extensions).** Using S4U is sometimes necessary but requires you to place your process within the trusted computing base (TCB) of the WCF server. This has the side effect of making your WCF service process very highly privileged. Where possible, you should avoid this approach because an attacker who manages to inject code and compromise your WCF application will have almost unrestricted capabilities on the local computer.
- **Using the LogonUser API.** Using **LogonUser** requires you to maintain user credentials on the WCF service. If you must use this API, be sure to protect the credentials passed to **LogonUser** by encrypting the credentials.

Related Items

- For more information, see "How To – Use Delegation for Flowing the Original Caller Credentials to Back-end in WCF Calling from Windows Forms."

- For more information, see “How To – Use Protocol Transition for Impersonating and Delegating the Original Caller in WCF.”

Additional Resources

- For more information, see “Delegation and Impersonation with WCF” at http://msdn.microsoft.com/en-us/library/ms730088.aspx
- For more information, see “How To: Use Impersonation and Delegation in ASP.NET 2.0” at http://msdn.microsoft.com/en-us/library/ms998351.aspx
- For more information on S4U Kerberos Extensions, see http://msdn.microsoft.com/en-us/magazine/cc188757.aspx

Chapter 7 – Message and Transport Security

Objectives

- Understand the differences between message security and transport security in WCF.
- Understand how to use message and transport security together.
- Understand the benefits and tradeoffs involved in using each security type.
- Understand how to decide when to use each option.

Overview

When working with WCF or Web services, securing communication between the client and the service is important. *Transfer security* is concerned with guaranteeing the integrity and confidentiality of WCF service messages as they flow from application to application across the network. Use encryption to enforce confidentiality and protect your messages from eavesdropping. Use integrity checks, such as a signature-based checksum, to protect your message from tampering.

In WCF, transfer security is also responsible for providing authentication. In the context of WCF, *authentication* refers to mutual authentication, where clients are not only uniquely identified to the service, but the service is also uniquely identified to the client.

Transfer security in WCF is achieved through the use of either transport security or message security.

Transport Security

When using *transport security*, the user credentials and claims are passed by using the transport layer. In other words, user credentials are transport-dependent, which allows fewer authentication options compared to message security. Each transport protocol (TCP, IPC, MSMQ, or HTTP) has its own mechanism for passing credentials and handling message protection. The most common approach for this is to use Secure Sockets Layer (SSL) for encrypting and signing the contents of the packets sent over Secure HTTP (HTTPS).

Transport security is used to provide point-to-point security between the two endpoints (service and client). If there are intermediary systems between client and the service, each intermediate point must forward the message over a new SSL connection.

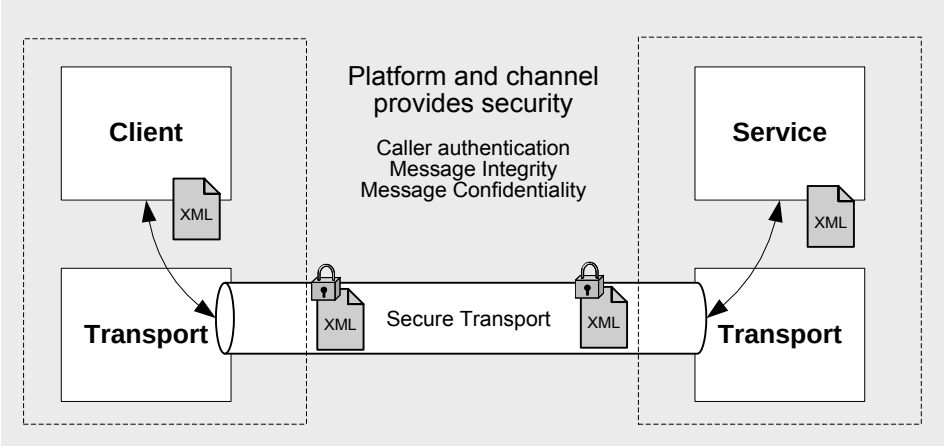


Figure 2. Transport Security

Use transport security in the following scenarios:

- You are sending a message directly from your application to a WCF service and the message will not be routed through intermediate systems.
- Both the service and the client are located in an intranet.

Using transport security offers the following advantages:

- It provides interoperability, meaning that communicating parties do not need to understand WS-Security specifications.
- It may result in better performance.
- Hardware accelerators can be used to further improve the performance.

Using transport security has the following disadvantages:

- Security is applied on a point-to-point basis, with no provision for multiple hops or routing through intermediate application nodes.
- It supports a limited set of credentials and claims compared to message security.
- It is transport-dependent upon the underlying platform, transport mechanism, and security service provider, such as NTLM or Kerberos.

Message Security

When using *message security*, the user credentials and claims are encapsulated in every message using the WS-Security specification to secure messages. This option gives the most flexibility from an authentication perspective. You can use any type of security credentials you want, largely independent of transport, as long as both the client and service agree.

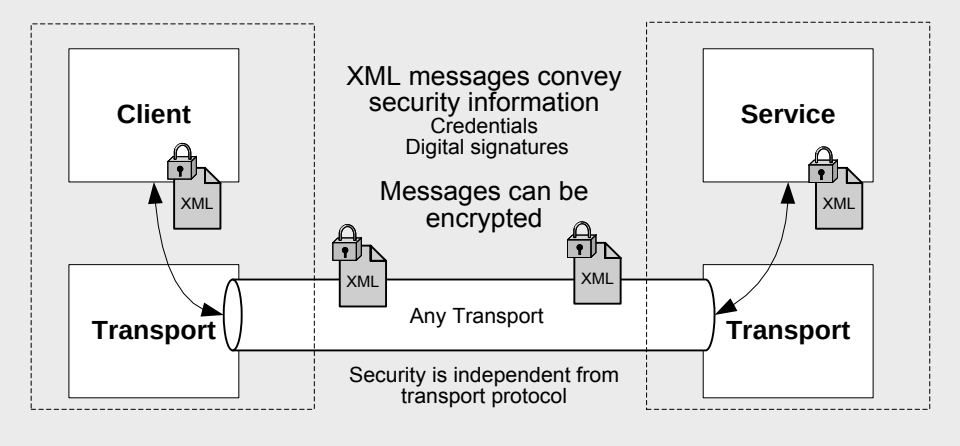


Figure 1. Message Security

Use message security in the following scenarios:

- You are sending a message to a WCF service, and the message is likely to be forwarded to other WCF services or may be routed through intermediate systems.
- Your WCF clients are accessing the WCF service over the Internet and messages may be routed through intermediate systems.

Using message security offers the following advantages:

- It provides end-to-end security. Because message security directly encrypts and signs the message, having intermediaries does not break the security.
- It allows partial or selective message encryption and signing, thus improving overall application performance.
- Message security is transport-independent and therefore can be used with any transport protocol.
- It supports a wide set of credentials and claims, including the issue token that enables federated security.

Using message security has following disadvantages:

- This option may reduce performance compared to transport security because each individual message is encrypted and signed.
- It does not support interoperability with older ASMX clients, as it requires both the client and service to support WS-Security specifications.

Transfer Security Modes

In WCF, you have two primary choices for providing transfer security: either you provide the transfer security on the transport level, or on the message level. Each option has its own advantages and disadvantages. The following table details the security modes available across most of the standard bindings.

Mode	Description
None	No security is provided; you should not use this option.
Transport	Mutual authentication and message protection are provided at the transport level.
Message	Mutual authentication and message protection are provided at the message level.
Both	Mutual authentication and message protection are provided at both the transport and message level. This is far more than is needed for most scenarios.
TransportWithMessageCredential	Client authentication is provided at the message level, and message protection and service authentication are provided at the transport level.
TransportCredentialOnly	Mutual authentication is provided at the transport level, but no message protection is provided. This option is available only on **BasicHttpBinding**.

Transport Security in WCF

In WCF, transport security depends on the binding and subsequent transport being used. Each protocol (TCP, HTTP, MSMQ, NamedPipes) has its own mechanism for passing credentials and handling message protection. Typically, you can use transport security when your client is deployed within an intranet, as it provides point-to-point security and better performance compared to message security. Note the following considerations:

- With transport security, the service credentials are negotiated by default.
- Transport security is available on all of the bindings except for **wsDualHttpBinding**.
- When using HTTP bindings, the WCF service typically is hosted in Internet information Services (IIS) and the transport security is provided by SSL. The SSL certificate is used to provide the message protection.
- With **netTcpBinding**, when using Windows authentication, the binding uses the service's Windows token to provide message protection. When using non-Windows authentication such as certificate authentication, you have to configure a service certificate as service credentials. The binding uses the service certificate for message protection.

Use the **<Security mode>** attribute to configure transport security on your binding. The following example shows how a **wsHttpBinding** binding is configured to use transport security:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security mode="Transport">
      </security>
    </binding>
```

```
  </wsHttpBinding>
</bindings>
…
```

Intranet Scenarios

The following are the authentication types and bindings that can be used in a typical intranet scenario:

- **Windows authentication with netTcpBinding** – By default, **netTcpBinding** uses Windows authentication and transport security. It uses the service account's Windows identity token to provide message protection. The credentials are negotiated with the Security Support Provider Interface (SSPI).
- **Certificate authentication with netTcpBinding** – By default, **netTcpBinding** uses transport security, which means you will have to configure the client credentials to use a certificate. To provide message protection at the transport level, you will have to configure a service certificate as service credentials. The certificate will negotiate a session key and service public key during the handshake, which will allow you to encrypt the content with the service certificate public key and sign the content with the private session key.

Note: In an intranet scenario, it is recommended that you use **netTcpBinding** unless you have a specific requirement to use other bindings such as **wsHttpBinding**. By default, **netTcpBinding** uses binary encoding and transport security, which delivers better performance.

Internet Scenarios

The following are the authentication types and bindings that can be used in a typical Internet scenario:

- **Basic authentication with basicHttpBinding** – By default, **basicHttpBinding** does not support any security, so you will need to configure the binding to use transport security. This is a good option when you want to support interoperability with non-WCF or non-Windows clients. In this scenario, you need to install a SSL certificate on IIS and then configure the virtual directory to require SSL. SSL will then negotiate a session key and service public key during the handshake, which will allow you to encrypt the content with the service certificate public key and sign the content with the private session key.
- **Certificate authentication with wsHttpBinding** – By default, **wsHttpBinding** uses message security and Windows authentication, so you will have to configure the binding to use transport security and configure the client credentials to use the certificate. To provide message protection at the transport level, install an SSL certificate on IIS and configure the virtual directory to require SSL.

Note: In an Internet scenario, you can only use the **HttpBinding** option.

Message Security in WCF

Message security uses the WS-Security specification to secure messages. The specification describes enhancements to Simple Object Access Protocol (SOAP) messaging to ensure confidentiality, integrity, and authentication at the SOAP message level (instead of the transport level). Typically, you can use transport security when your client is deployed over the Internet, as it provides end-to-end security.

With transport security, the service credentials are negotiated by default, but you can configure the message security to avoid service credential negotiation if you want to restrict clients from accessing your service. This is especially important when you are in partner scenario where your service is exposed to a number of clients. When you configure message security to not negotiate credentials, you have to make sure that the service credentials are available out-of-band to the client application.

Transport security is available on all of the bindings except for **netNamedPipeBinding**.

When using Windows authentication, message security uses the service's Windows token to provide message security. When using non-Windows authentication such as username, certificate, or issue token authentication, you have to configure a service certificate as service credentials. Message security uses the service certificate for message protection.

Use the **<Security mode>** attribute to configure message security on your binding. The following example shows **netTcpBinding** configured to use message security:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="netTcpEndpointBinding">
      <security mode="message">
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
…
```

By default, message security encrypts and signs the messages. Although it is not recommended, with message security you can lower the protection level or disable it based on your requirements.

Protection Level

You can use the **[ServiceContract(ProtectionLevel)]** attribute to specify message security protection levels on the interface or operation level. The available protection level options are:

- **None –** Use **None** to turn off signing and encryption on the operation or interface.
- **Sign –** Use **Sign** to sign the interface or operation but not encrypt it.
- **EncryptAndSign –** Use **EncryptAndSign** to both encrypt and sign the interface or operation.

The following code snippet creates an interface with the protection level set to **Sign**.

```
 [ServiceContract(ProtectionLevel=ProtectionLevel.Sign]
public interface IService
{
      string GetData(int value);
}
```

The following code snippet specifies an operation with the protection level set to **Sign**.

```
 [OperationContract(ProtectionLevel=ProtectionLevel.Sign]
string GetData(int value);
```

Intranet Scenarios

Message security is not the best choice in an intranet scenario, but if your requirements force you to use message security, the following authentication types and bindings can be used in a typical intranet scenario:

- **Windows authentication with netTcpBinding –** By default, **netTcpBinding** uses Windows authentication and transport security. You will have to configure the binding to use message security. The binding uses the service account's Windows identity token to provide message protection. The credentials are negotiated with SSPI.
- **Certificate authentication with netTcpBinding –** You will have to configure the binding to use message security and configure the client credentials to use the certificate. To provide message protection at the message level, you will have to configure a service certificate as the service credentials. The certificate will negotiate a session key and service public key during the handshake, which will allow you to encrypt the content with the service certificate public key and sign the content with the private session key.
- **Username authentication with netTcpBinding –** You will have to configure the binding to use message security and configure the client credentials to use username authentication. To provide message protection at the message level, you need to install and configure a service certificate as service credentials.

Note: In an intranet scenario, it is recommended that you use **netTcpBinding** unless you have a specific requirement to use other bindings such as **wsHttpBinding**. By default, **netTcpBinding** uses binary encoding and transport security, which may improve the performance of your service.

Internet Scenarios

Message security is the preferred option for Internet scenarios. The following are the authentication types and bindings that can be used in a typical Internet scenario:

- **Basic authentication with basicHttpBinding –** By default, **basicHttpBinding** does not support any security, so you will have to configure the binding to use message security. Using this option does not allow you to support interoperability. In this scenario, you need

to install and configure a service certificate as service credentials. The certificatewill negotiate a session key and service public key during the handshake, which will allow you to encrypt the content with the service certificate public key and sign the content with the private session key.

- **Certificate authentication with wsHttpBinding –** By default, **wsHttpBinding** uses message security and Windows authentication, so you will have to configure the client credentials to use the certificate. To provide message protection at the message level ,install and configure a service certificate as service credentials.

Note: In an Internet scenario, you can only use the **HttpBinding** option.

Chapter 8 – Bindings

Objectives

- Understand what bindings and behaviors are and how they are used in WCF.
- Understand when to use each binding supplied by WCF.
- Understand which bindings are best for the Internet.
- Understand which bindings are best for an intranet.

Overview

WCF is a framework for building services that allows you to transmit messages using different transport protocols and different XML representations. It allows you to enhance message interactions with a suite of Simple Object Access Protocol (SOAP) protocols. WCF uses a channel stack that handles all of these communication details.

It would be challenging to build a channel stack from scratch, as you would have to decide the ordering of the components and whether or not they are compatible with one another. For this reason, WCF indirectly configures the underlying channel stack with the help of configurable endpoints. An endpoint specifies an address, a binding, and a contract. The address specifies the network address where you want to listen for messages; the contract specifies what the messages arriving at the specified address should contain; and the binding provides the channel stack needed to process the message. When loading a service, WCF builds the channel stack by following the instructions outlined by the binding description.

WCF Built-in Bindings

Bindings define how clients can connect and communicate with your service. All the bindings in WCF are represented by the **System.ServiceModel.Channels.Binding** class, which is the base class for all bindings. Each class defines a different channel stack configuration through its implementation. A binding includes definition for the WS-* protocols used, the message encoding, and the transport protocol.

WCF comes out of the box with a set of bindings configured for the most-common scenarios. If none of the bindings are a good fit, you can create a custom binding to configure the service explicitly to meet your needs.

The following table summarizes common bindings.

Binding	Description
basicHttpBinding	Represents a binding that configures and exposes endpoints that are able to communicate with ASMX-based Web services and clients and other services that conform to the WS-I Basic Profile 1.1 specification. By default, **basicHttpBinding** has security disabled.

wsHttpBinding	Defines a secure, reliable, interoperable binding suitable for non-duplex service contracts. The binding implements the following specifications: WS-Reliable Messaging for reliability, and WS-Security for message security and authentication. The transport is HTTP, and message encoding is text/XML encoding. By default, it provides message security using Windows authentication.
ws2007HttpBinding	Defines a secure, reliable, interoperable binding suitable for non-duplex service contracts. The binding implements the following specifications: WS-Reliable Messaging for reliability, and WS-Security for message security and authentication. The transport is HTTP, and message encoding is text/XML encoding. The **ws2007HttpBinding** provides binding similar to **wsHttpBinding** but uses the standard for OASIS (Organization for the Advancement of Structured Information Standards). By default, it provides message security using Windows authentication.
netTcpBinding	Specifies a secure, reliable, optimized binding suitable for cross-machine communication. By default, it generates a run-time communication stack with transport security and Windows authentication as default security settings. It uses the Transmission Control Protocol (TCP) for message delivery, and binary message encoding.
netNamedPipeBinding	Defines a binding that is secure, reliable, and optimized for cross-process communication on the same machine. By default, it generates a run-time communication stack with WS-ReliableMessaging for reliability, transport security for transfer security, named pipes for message delivery, and binary message encoding. It is not secured by default.
netMsmqBinding	Defines a queued binding suitable for cross-machine communication.
wsFederationHttpBinding	Defines a binding that supports federated security. It helps in implementing federation, which is the ability to flow and share identities across multiple enterprises or trust domains for authentication and authorization. WCF implements federation over message and mixed mode security but not over transport security. Services configured with this binding must use the HTTP protocol as transport.
ws2007FederationHttpBinding	Defines a binding that derives from **wsFederationHttpBinding** and supports federated security. It helps in implementing federation. WCF implements federation over message and mixed mode security but not over transport security. Services configured with this binding must use the HTTP protocol as

	transport. The **ws2007FederationHttpBinding** provides binding similar to **ws2007FederationHttpBinding** but uses the OASIS standard.
wsDualHttpBinding	Defines a secure, reliable, and interoperable binding that is suitable for duplex service contracts or communication through SOAP intermediaries.
customBinding	Allows you to create a custom binding with full control over the message stack.

For more information on bindings, see "Windows Communication Foundation Bindings" at http://msdn.microsoft.com/en-us/library/ms733027.aspx.

Bindings Behaviors and Endpoints

A WCF service endpoint comprises an address, a binding, and a contract. Bindings define how clients can connect and communicate with your service. A binding includes definitions for the WS-* protocols used, the message encoding, and the transport protocol. For instance, **wsHttpBinding** uses HTTP, XML 1.0 encoding, message security, reliable sessions, and transactions by default. Bindings are exposed by a service endpoint that includes the binding plus a uniform resource identifier (URI) to which the client will send messages. Bindings can be configured either through code or by using configuration elements in the configuration file.

The following example shows **wsHttpBinding** configured to use transport security:

```
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security mode="Transport">
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

The following configuration snippet shows an endpoint that exposes this binding:

```
<endpoint address="" binding="wsHttpBinding"
bindingConfiguration="wsHttpEndpointBinding" name="wsHttpEndpoint"
contract="IService">
```

When creating an overall security policy for your services, you will use bindings and behaviors to configure your service as follows:

- **Bindings.** Bindings control the security mode, client credential type, and other security settings.
- **Behaviors.** Behaviors control impersonation levels, how client credentials are authenticated and authorized, and service credentials.

Bindings Summary

Use the following binding summaries to help you choose the right binding for your scenario.

basicHttpBinding

If your service needs to support legacy clients that expect an ASMX Web service, consider using **basicHttpBinding**. Because **basicHttpBinding** does not implement any security by default, if you require message or transport security, you should configure it explicitly on this binding. Use **basicHttpBinding** to expose endpoints that are able to communicate with ASMX-based Web services and clients and other services that conform to the WS-I Basic Profile 1.1 specification. When configuring transport security, **basicHttpBinding** defaults to no credentials just like a classic ASMX Web service. **basicHttpBinding** allows you to host your service in Internet Information Services (IIS) 5.0 or IIS 6.0.

wsHttpBinding

If your service will be called by WCF clients over the Internet, consider using **wsHttpBinding**. **wsHttpBinding** is a good choice for Internet scenarios in which you do not have to support legacy clients that expect an ASMX Web service. If you do need to support legacy clients, consider using **basicHttpBinding** instead. **wsHttpBinding** allows you to host your service in IIS 5.0 or IIS 6.0.

netTcpBinding

If you need to support clients within your intranet, consider using **netTcpBinding**. **netTcpBinding** is a good choice for an intranet scenario if transport performance is important to you and it is acceptable to host the service in a Windows service instead of in IIS. **netTcpBinding** uses the TCP protocol and provides full support for SOAP security, transactions, and reliability. Use this binding when you want to provide a secure and reliable binding environment for .NET-to-.NET cross-machine communication. **netTcpBinding** does not allow you to host your service in IIS 5.0 or IIS 6.0; instead, host in a Windows service or in IIS 7.0.

netNamedPipeBinding

If you need to support WCF clients on the same machine as your service, consider using **netNamedPipeBinding**. **netNamedPipeBinding** provides a secure and reliable binding environment for cross-process, same-machine communication. Use this binding when you want to make use of the Named-Pipe protocol and provide full support for SOAP security, transactions, and reliability. **netNamedPipeBinding** does not allow you to host your service in IIS 5.0 or IIS 6.0; instead, host in a Windows service or in IIS 7.0.

netMsmqBinding

If you need to support disconnected queuing, use **netMsmqBinding**. Queuing is provided by using Microsoft Message Queuing (MSMQ) as a transport, which enables support for disconnected operations, failure isolation, and load leveling. You can use **netMsmqBinding** when the client and the service do not have to be online at the same time. You can also manage any number of incoming messages by using load leveling. MSMQ supports failure isolation,

where messages can fail without affecting the processing of other messages. **netMsmqBinding** does not allow you to host your service in IIS 5.0 or IIS 6.0; instead, host in a Windows service or in IIS 7.0.

wsDualHttpBinding

If you need to support a duplex service, use **wsDualHttpBinding**. A *duplex service* is a service that uses duplex message patterns, which provides the ability for a service to communicate back to the client via a callback. You can also use this binding to support communication via SOAP intermediaries. **wsDualHttpBinding** does not allow you to host your service in IIS 5.0 or IIS 6.0; instead, host in a Windows service or in IIS 7.0.

CustomBinding

A custom binding is created in code by using the **CustomBinding** class found in the **System.ServiceModel.Channels** namespace. This class exposes a collection of binding elements to which you can add further binding elements. This allows you to compose a new binding based on a set of existing binding elements.

User-defined bindings are bindings that are created by inheriting from the **Binding** class. Creating user-defined bindings is preferred when you want to reuse the binding in a number of applications.

Internet Binding Scenarios

If you are exposing your WCF service interface over the Internet, use the following guidelines to help choose the appropriate binding:

- If you are exposing your WCF service over the Internet to clients that expect a legacy ASMX Web service, use **basicHttpBinding**. Keep in mind that this binding does not have any security enabled by default, so all messages will be sent in plaintext format.
- If you are exposing your WCF service over the Internet to Windows Forms clients, use **wsHttpBinding**. **wsHttpBinding** provides the best WS-* interoperability features, including WS-SecureConversation, WS-Addressing, and WS-AtomicTransaction. The combination of features offered by **wsHttpBinding** makes for the most reliable connection offered by WCF over the Internet.
- If you are exposing your WCF service over an intranet to an ASP.NET application, which in turn is exposed to the clients over the Internet, use **netTcpBinding**.
- If your clients and the service require full-duplex communication, then use **wsDualHttpBinding**. It is the only binding that supports full-duplex.
- If your service is interacting with Web Services Enhancements (WSE) clients, you must use **customBinding**. The service must use a custom binding to be compatible with the August 2004 version of the WS-Addressing specification.

Intranet Binding Scenarios

If you are exposing your WCF service interface over an intranet, use the following guidelines to help choose the appropriate binding:

- If you are exposing your WCF service over your intranet to clients that expect a legacy ASMX Web service, use **basicHttpBinding**. Keep in mind that this binding does not have any security enabled by default, so all messages will be sent in plaintext format.
- If you are exposing your WCF service over your intranet to Windows Forms or ASP.NET clients, use **netTcpBinding**. You can use any binding over an intranet, but **netTcpBinding** provides the best throughput performance. On an intranet, you generally do not need to worry as much about the connection going down as with an Internet connection, so some of the WS-* advantages supplied with **wsHttpBinding** may not be as necessary on an intranet.

Binding Elements

WCF provides numerous channels and encoders that are used in the preconfigured bindings. You can use these channels to provide binding elements that can be used in custom bindings.

A *binding element* is a class that derives from **System.ServiceModel.Channels.BindingElement**.

WCF provides some different lists of binding elements that include the Protocol Binding Elements, Message Encoding Binding Elements, Transport Security Binding Elements, and Transport Binding Elements.

Protocol Binding Elements

Protocol	Class	Element
Transaction Flow	**TransactionFlowBindingElement**	<transactionFlow/>
Reliable Messaging	**ReliableSessionBindingElement**	<reliableSession/>
Security	**SecurityBindingElement**	<security/>

Message Encoding Binding Elements

Message encoding	Class	Element
Text	**TextMessageEncodingBindingElement**	<textMessageEncoding/>
MTOM	**MtomMessageEncodingBindingElement**	<mtomMessageEncoding/>
Binary	**BinaryMessageEncodingBindingElement**	<binaryMessageEncoding/>

Transport Security Binding Elements

Transport security	Class	Element
Windows	**WindowsStreamSecurityBindingElement**	<windowsStreamSecurity/>
SSL	**SslStreamSecurityBindingElement**	<sslStreamSecurity/>

Transport Binding Elements

Transport	Class	Element
HTTP	**HttpTransportBindingElement**	<httpTransport/>
HTTPS	**HttpsTransportBindingElement**	<httpsTransport/>
TCP	**TcpTransportBindingElement**	<tcpTransport/>
Named pipes	**NamedPipeTransportBindingElement**	<namedPipeTransport/>
MSMQ	**MsmqTransportBindingElement**	<msmqTransport/>
MSMQ	**MsmqIntegrationBindingElement**	<msmqIntegration/>
P2P	**PeerTransportBindingElement**	<peerTransport/>

You can add binding elements by adding the desired **BindingElement** objects to its Elements collection. The order in which the binding element is added is very important. The order of adding the binding elements is as follows:

1. Transaction Flow (not required)
2. Reliable Messaging (not required)
3. Message Security (not required)
4. Composite Duplex (not required)
5. Message Encoding (**required**)
6. Transport Security (not required)
7. Transport (**required**)

The Transport binding element is the only required element when defining a custom binding. The Message Encoding element is required for each binding, but if you do not specify one, WCF will add a default encoding. The default encoding for HTTP(S) is text, and for all other transports it is binary.

The following code shows how to create a custom binding:

```
CustomBinding myHttpBinding = new CustomBinding();
myHttpBinding.Name = “myHttpBinding”;
myHttpBinding.Elements.Add(new HttpTransportBindingElement());

host.AddServiceEndpoint(typeof(IChat), myHttpBinding,
    “http://localhost:8080/chat/custom”);
```

The following code shows how to create a custom binding by using the **customBinding** element in the configuration:

```
 <bindings>
      <customBinding>
        <binding name=”myHttpBindingConfiguration”>
          <textMessageEncoding
            messageVersion=”Soap11WSAddressingAugust2004”/>
          <httpTransport
            useDefaultWebProxy=”true” transferMode=”Streamed”/>
        </binding>
      </customBinding>
</bindings>
```

Custom Binding Configuration Examples

The following example shows a custom binding that performs functions similar to those of **wsHttpBinding** and **netTcpBinding**:

```
<configuration>
  <system.serviceModel>
    …
    <bindings>
      <customBinding>

        <binding name=”myWSHttpBindingConfiguration”>
          <transactionFlow/>
          <reliableSession ordered=”true”/>
          <security authenticationMode=”SspiNegotiated”/>
          <binaryMessageEncoding/>
          <httpTransport/>
        </binding>
        <binding name=”myNetTcpBindingConfiguration”>
          <transactionFlow/>
          <textMessageEncoding/>
          <windowsStreamSecurity/>
          <tcpTransport/>
        </binding>

      </customBinding>
    </bindings>
    …
  </system.serviceModel>
</configuration>
```

The **myWSHttpBindingConfiguration** configuration is similar to the built-in **wsHttpBinding** except that it uses binary message encoding and enables transaction flow and ordered reliable messaging. The **myNetTcpBindingConfiguration** configuration is similar to **netTcpBinding** except that it uses text message encoding and enables transaction flow.

PART III

Intranet Application Scenarios

In This Part:

- **Intranet - Web to Remote WCF Using Transport Security (Original Caller, TCP)**
- **Intranet - Web to Remote WCF Using Transport Security (Trusted Subsystem,HTTP)**
- **Intranet - Web to Remote WCF Using Transport Security (Trusted Subsystem TCP)**
- **Intranet - Windows Forms to Remote WCF Using Transport Security (Original Caller, TCP)**

Chapter 9 - Intranet - Web to Remote WCF Using Transport Security (Original Caller, TCP)

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5

Scenario

In this scenario, your users have Windows accounts and use a Web client to connect over the intranet to an ASP.NET application on an IIS server that is hosted on an Application Server. The ASP.NET application makes calls to the WCF service. The business logic called by the WCF service requires fine-grained authorization and is backed by a SQL Server data store. The basic model for this application scenario is shown in the following figure.

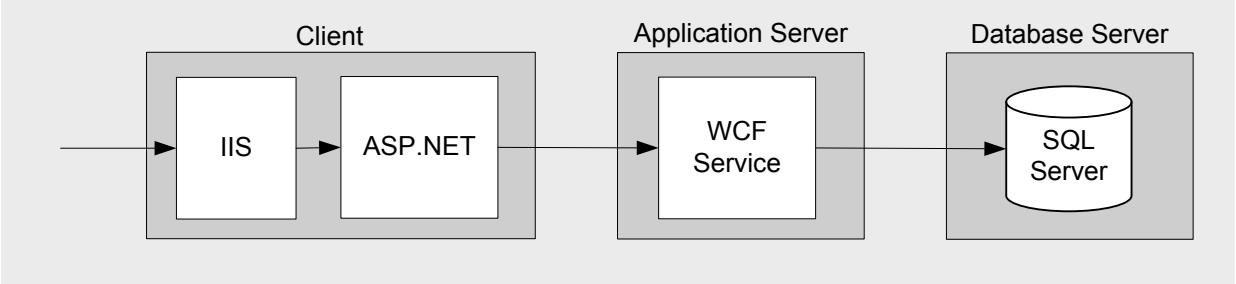


Figure 1. Web to Remote WCF Using Transport Security (Original Caller, TCP) – Model

Key Characteristics

This scenario applies to you if:

- Your users have browsers supporting Integrated Windows Authentication.
- Your user accounts are in Active Directory within a domain.
- Your user roles are Windows Groups.
- The business logic behind your WCF service requires fine-grained authorization.
- Your application transmits sensitive data over the network that needs to be protected.
- A high-performance connection between the ASP.NET application and the WCF service is more important than the ability to host the WCF service in IIS.

Solution

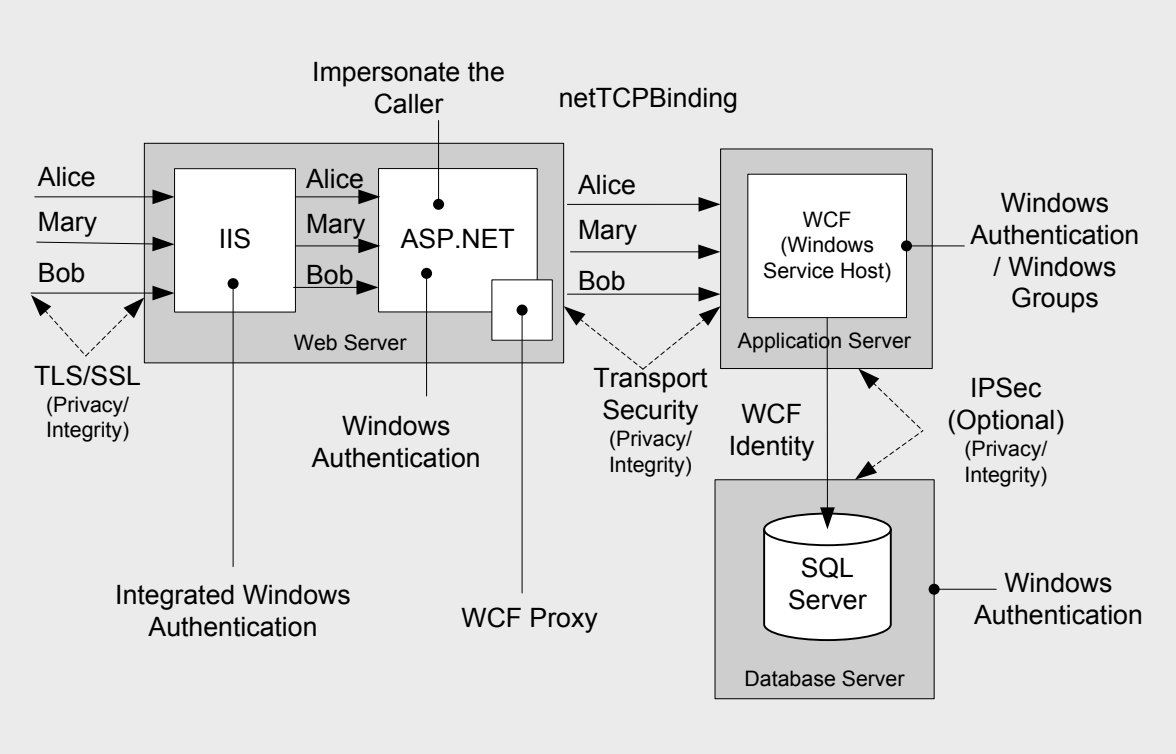


Figure 2. Web to Remote WCF Using Transport Security (Original Caller, TCP) – Solution

Solution Summary Table

In this solution you will:

- Use domain credentials to authenticate clients against an Active Directory user store.
- Impersonate the original caller when calling methods on the WCF service from the ASP.NET application.
- Use a service account to call the SQL Server from WCF (without impersonation).
- Use SSL to protect sensitive data between the Web client and IIS.
- Use Transport Security to protect sensitive data between the ASP.NET application and the WCF service.
- Use **netTcpBinding** to support the TCP transport for improved performance.
- Host WCF in a Windows Service since IIS does not support the TCP transport (prior to IIS7)

Web Server

Check & More Info	Example
IIS - Configuration	
A dedicated application pool is created and configured to run under a custom service account. Use a domain account.	
The Web application is configured to run under the service account. Assign the Web application to the custom application pool.	
Service Principal Name (SPN) is created if domain identity is used in the ASP.NET application pool. Create an SPN for both the DNS and NETBIOS machine name.	`setspn -a HTTP//WebServer.domain.com customDomainAccount` `setspn -a HTTP//WebServer customDomainAccount`
ASP.NET Process identity is configured as Trusted for delegation. If you use a network service account, enable the computer account for trusted for delegation. If you use a domain user account. enable the domain account for trusted for delegation.	
IIS - Authentication	
The IIS virtual directory is configured to use Windows Integrated Authentication. Users will be authenticated with Windows Authentication.	
Anonymous access is disabled.	

Check & More Info	Example
ASP.NET - Authentication	
ASP.NET is configured for Windows Integrated authentication. The Web application will authenticate the users.	`<authentication mode = "Windows" >`
ASP.NET - Authorization	

If you have role segmentation in your application then use URL authorization.
 	
The authorized users have access to specific pages.
	`<authorization>`
	`  <allow roles="domainName\RoleName" />`
	`  <deny users="*" />`
	`</authorization>`

Role Manager is enabled and Role-checks are performed using Role Manager API.
 	
Original users are authorized using the Windows groups before calling in WCF Service.
	`<roleManager enabled="true" defaultProvider= "AspNetWindowsTokenRoleProvider"/>`
	
	`if (Roles.IsUserInRole(@"npscode\Accounting") ) { }`

WCF Proxy	
ASP.NET has a proxy reference to the WCF service. The application has access to the WCF Service metadata to create a service reference.	`WCFTestService.MyServiceClient proxy = new WCFTestService.MyServiceClient();`
ASP.NET Impersonates the original callers before calling the WCF operation. Used for downstream authorization.	`using (((WindowsIdentity)HttpContext.Current.User.Identity).Impersonate()) { WCFTestService.MyServiceClient proxy = new WCFTestService.MyServiceClient(); proxy.GetData("data"); proxy.Close(); }`

Application Server

Check & More Info	Example
Windows Service - Configuration	
Windows Service is configured to run under a custom domain service account. Use a domain account if possible.	
WCF service is hosted in a Windows Service. Since IIS does not support **netTcpBinding**, host in Windows Service.	
Service Principal Name (SPN) is created since a custom domain account is used for the Windows service, and the ASP.NET application needs to restrict trust for delegation to only the WCF service.	`setspn -a WCFServiceHost//WebServer.domain.com customDomainAccount` `setspn -a WCFServiceHost//WebServer customDomainAccount`

Check & More Info	Example
Create an SPN for both the DNS and NETBIOS machine name.	
WCF Service - Configuration	

Configure the WCF service to use **netTcpBinding**.
 	
NetTcpBinding uses the TCP protocol and provides full support for SOAP security, transactions, and reliability. As client and WCF service both are in the intranet, this is a good choice from a performance perspective.
	`<endpoint`
	`address=""`
	`binding="netTcpBinding"`
	`bindingConfiguration=""`
	`name="TcpBinding"`
	`contract="WCFServicecHost.IMyService"`
	`/>`

A mex endpoint is created for publishing the metadata.
 	
This is required so that client can add reference to the WCF Service using SvcUtil utility.
	`<endpoint`
	`address="Mex"`
	`binding="mexTcpBinding"`
	`bindingConfiguration=""`
	`name="MexEndpoint"`
	`contract="IMetadataExchange" />`

Service Metadata is configured in service behavior. The service metadata entry is required for the Windows service host to start. Both HTTP and HTTPS get are disabled.	`<serviceMetadata />`
WCF Service - Authentication	

netTcpBinding is configured to use Windows Authentication and Transport security.
 	
netTcpBinding by default supports Windows Authentication and Transport Security.
	`<endpoint`
	`address=""`
	`binding="netTcpBinding"`
	`bindingConfiguration="" />`

WCF Service - Authorization	

Role Manager feature is enabled and WindowsTokenRoleProvider is configured for roles authorization.
 	
Roles authorization can be performed declaratively or imperatively in the operation contract.
	`<serviceAuthorization`
	`principalPermissionMode="UseAspNetRoles"`
	`roleProviderName="AspNetWindowsTokenRoleProv`
	`ider" />`

Perform role-checks declaratively using the **PrinciplePermission** attribute.
 	
Use declarative check to authorize the user on individual methods.
	`[PrincipalPermission(SecurityAction.Demand,`
	`Role = "npscode\\accounting")]`
	
	`public string GetData(string message)`
	`{`
	`return "hello";`

Check & More Info	Example
	`}`

Perform role-checks imperatively using IsUserInRole() method.
 	
Use programmatic check to authorize the user based on business logic.
	`if(Roles.IsUserInRole(@"npscode\Accounting")`
	`)`
	`{`
	`  //business operation for accounting`
	`}`
	`else`
	`{`
	`  //business operation for others`
	`}`

WCF Service - SQL	
The connection string for database is configured to use Windows Authentication. The service does not impersonate the original caller to benefit for connection pooling. The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes.	`SqlConnection sqlcon = new SqlConnection("Server=10.3.19.11;Database=Northwind;IntegratedSecurity=SSPI");`
Database connection is opened using the WCF process identity's security context. Service does not impersonate the original caller to benefit for connection pooling.	

Database Server

Check & More Info	Example
Configuration	
A SQL Server login is created for the WCF's service account (process identity). This grants access to the SQL Server.	`exec sp_grantlogin 'Custom Service Account'`

The login is mapped to a database user for the Web application.
 	
This grants access to the specified database.
	`use targetDatabase`
	`go`
	`exec sp_grantdbaccess ' Custom Service Account'`
	`go`

A database role is created in the target database.
 	
This allows access control and authorization to the DB.
	`use targetDatabase`
	`go`
	`exec sp_addrole 'DB Role Name'`
	`go`

Check & More Info	Example

The login is added to the database role.
 	
Grant minimum permissions. For example grant execute permissions to selected stored procedures and provide no direct table access.
	`use targetDatabase`
	`go`
	`exec sp_addrolemember 'DB Role Name',`
	`'Custom Service Account'`
	`go`

Authentication	
SQL Server is configured to use Windows Authentication.	

Communication Security

Check & More Info	Example
Browser to Web Server	
Use SSL between the browser and Web server to protect sensitive data on the wire. Install certificate in the Web site. Configure the virtual directory of the Web application to use SSL.	
App Server to Database Server	
You can use IPSec or SSL between the App Server and Database Server to protect sensitive data on the wire.	

Analysis

Web Server

Authentication

- To prevent unauthenticated and unauthorized users from accessing pages, anonymous access is disabled in IIS.
- Integrated Windows Authentication is a good choice for this scenario because all users have Windows accounts. Integrated Windows Authentication provides the benefit of keeping the user's password from ever being sent over the network. Additionally, the logon is transparent for the user because Windows uses the current user's logon session.

Authorization

- Use URL authorization to perform role checks against the original caller and restrict access to pages based on role permissions.
- The Roles Manager is a good choice for this scenario because it allows your service code to look up users' roles without writing and maintaining custom code.
- The original caller is passed to the WCF service to allow authorization decisions downstream.

WCF Proxy

- Because original user's credentials are passed to WCF for authentication and authorization, the original caller is impersonated before making calls into WCF Service. All calls through the WCF proxy and into the WCF service use the original user's security context.

Configuration

- In order to reduce attack surface and minimize the impact of a compromise, the ASP.NET application on the Web Server runs under the security context of the Service account using a least-privileged account.
- In order to support Kerberos mutual authentication, an SPN is created for your custom domain account running the ASP.NET application.
- Configure the custom domain account in Active Directory to trust for delegation. This allows ASP.NET to flow the original caller credentials to the WCF service.

Application Server

Authentication

- In order to authenticate the original caller in the WCF Service, WCF uses Windows Authentication.

Authorization

- For coarse-grained access control, the WCF Service manages authorization checks at the operation level, declaratively.
- For fine-grained access control, authorization checks are made programmatically within the operations.
- The Roles Manager is a good choice for this scenario because it allows your service code to look up users' roles without writing and maintaining custom code.

Data Access

- To reduce the risk of database credentials theft, the database connection string is configured to use Windows Authentication. This choice avoids storing credentials in files and passing credentials over the network to the Database Server.
- The WCF service accesses the database using the WCF process identity. As a result, all calls use the single process account and designated database connection pooling.

Configuration

- This scenario is optimized around transmission performance at the expense of interoperability with clients that expect a legacy Web service and the ability to host the service in IIS. For this reason, the best binding choice is **netTcpBinding**. By default, **netTcpBinding** supports Windows Authentication with Transport Security.
- Because IIS 6.0 does not support **netTcpBinding**, the WCF service is hosted in a Windows service.
- In order to reduce attack surface and minimize the impact of a compromise, the Windows service runs under the security context of the Service account using a least-privileged account.
- A metadata exchange (mex) endpoint is exposed to make it possible for the client to generate a proxy based on the service definition.

Database Server

- SQL Server database user roles are preferred to SQL Server application roles to avoid the associated password management and connection pooling issues associated with the use of SQL application roles. Applications activate SQL application roles by calling a built-in stored procedure with a role name and a password. Therefore, you must securely store the password. You must also disable database connection pooling when you use SQL application roles, which severely impacts application scalability.
- Creating a new user-defined database role and adding the database user to the role lets you give specific minimum permissions to the role. In this way, if the database account changes you don't have to change the permissions on all database objects.

Communication Security

- SSL protects sensitive data on the wire between the browser and Web Server.
- Transport Security protects sensitive data between the Web Server and App Server.
- You can use IPSec or SSL between the App Server and Database Server to protect sensitive data on the wire.

Example

Domain Controller

Configuration

Create a service principle name (SPN) based on these rules:

1. If the ASP.NET application runs in an application pool with a custom domain identity, create an SPN and map the custom domain identity with the HTTP service class and both the DNS machine name and the NETBIOS machine name:
```
setspn -a HTTP//WebServer.domain.com customDomainAccount
setspn -a HTTP//WebServer customDomainAccount
```

2. If the service runs under a custom domain identity:
```
setspn -a ServiceNameofWcfService//WebServer.domain.com
customDomainAccount
setspn -a ServiceNameofWcfService//WebServer customDomainAccount
```

3. If the service runs under the network service account:
```
setspn -a ServiceNameofWcfService//WebServer.domain.com computerName
setspn -a ServiceNameofWcfService//WebServer computerName
```

Note: You should specify the service name as it is displayed in the MMC services console.

4. Additionally:
   - The machine account of the Web application is configured trusted for delegation if the ASP.NET application runs under the network service account.
   - The domain account is configured trusted for delegation if the ASP.NET application runs under a custom domain identity.

Web Server

Code

- Role-authorization occurs before WCF service invocation.
- ASP.NET impersonates the original caller if it is authorized.

```
using System.Security.Principal;
…
protected void Button1_Click(object sender, EventArgs e)
{
  if (Roles.IsUserInRole(@"npscode\Business Represenatatives"))
   {
   using (((WindowsIdentity)HttpContext.Current.User.Identity).Impersonate())
      {
        WCFTestService.MyServiceClient proxy
          = new WCFTestService.MyServiceClient();
        proxy.GetData("data");
        proxy.Close();
       } //end using
    } //end if
 } //end function
```

Web.config Configuration

- Windows Authentication is enabled.
- URL authorization check is enabled.
- Role Manager is enabled.

```
<system.web>
   <assemblies>
     <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
     <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
     <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
     <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
    </assemblies>

   <authentication mode="Windows" />
      <authorization>
        <allow roles="npscode\BusinessDivision" />
        <deny users="*" />
   </authorization>

   <roleManager enabled="true"
        defaultProvider= "AspNetWindowsTokenRoleProvider"/>
   <pages>
      <controls>
        <add tagPrefix="asp" namespace="System.Web.UI"
```

```
            assembly="System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
        <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
            assembly="System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      </controls>
   </pages>

   <httpHandlers>
     <remove verb="*" path="*.asmx"/>

     <add verb="*" path="*.asmx" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>

     <add verb="*" path="*_AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>

     <add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
validate="false"/>
    </httpHandlers>

    <httpModules>
       <add name="ScriptModule" type="System.Web.Handlers.ScriptModule,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
    </httpModules>

</system.web>
```

Application Server

Code

- The service performs imperative authorization checks calling Roles.IsUserInRole.
- The service calls SQL using the security context of the WCF service and Windows Authentication.

```
using System.Data.SqlClient;
using System.Web.Security;

public string GetData(string myValue)
{
  if(Roles.IsUserInRole(@"npscode\Accounting"))
  {

    SqlConnection sqlcon = new
SqlConnection("Server=SQLserver;Database=Northwind;IntegratedSecurity=SSPI");

    sqlcon.Open();
    //do the business operation
    return "Authorization succeeded ";
  }
  else return "authorization failure";
```

```
}
```

Configuration

- The service has a binding endpoint that uses **netTcpbinding** with the default settings.
- The service has a mex endpoint to publish metadata.
- The service has a base address configured.
- The service configuration file has an entry for the **AspNetWindowsTokenRoleProvider** under system.web.
- The service behavior is configured with element serviceAuthorization to allow WindowsTokenRoleProvider as authorization provider.
- The service behavior is configured with element serviceMedata to allow metadata to be published.

```
<system.web>
    <roleManager enabled="true"
              defaultProvider="AspNetWindowsTokenRoleProvider" />
</system.web>

<system.serviceModel>
    <behaviors>
        <serviceBehaviors>
            <behavior name="BehaviorConfiguration">
                <serviceAuthorization
                    principalPermissionMode="UseAspNetRoles"
                    roleProviderName="AspNetWindowsTokenRoleProvider" />
                <serviceMetadata />
            </behavior>
        </serviceBehaviors>
    </behaviors>

    <bindings />

    <services>
        <service
          behaviorConfiguration="BehaviorConfiguration"
          name="WCFServicecHost.MyService">
            <endpoint address="Mex"
                      binding="mexTcpBinding"
                      bindingConfiguration=""
                      name="MexEndpoint"
                      contract="IMetadataExchange" />

            <endpoint address=""
                      binding="netTcpBinding"
                      bindingConfiguration=""
                      name="TcpBinding"
                      contract="WCFServicecHost.IMyService" />
            <host>
                <baseAddresses>
                    <add baseAddress=
                        "net.tcp://perfpres02.npscode.com/MyService" />
                </baseAddresses>
            </host>
```

```
        </service>
    </services>
</system.serviceModel>
```

Database Server

Configuration

- A SQL Server login is created for the WCF service account.
- The WCF login name is given access to the database.
- The role is created in the database.
- The WCF login name is added to the role.

```
-- Create a SQL Server login that matches the WCF machine name
EXEC SP_GRANTLOGIN 'npscode\perfpres02$'

-- Grant the login access to the application database
use testdb
go
exec sp_grantdbaccess 'npscode\perfpres02$'

-- Create the new database role
use testdb
go
exec sp_addrole 'myrole2','db_owner'

-- Add the new login to the role
use testdb
go
exec sp_addrolemember 'myrole2','npscode\aspnethost'
```

Additional Resources

- For more information on impersonation, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms731090.aspx.
- For further information on impersonation, see “How to: Impersonate a Client on a Service” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx.
- For more information on constrained delegation, see “How To: Use Protocol Transition and Constrained Delegation in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998355.aspx
- For more information on how to impersonate original caller from Web application, see How To: Impersonate the Original Caller in WCF Calling from a Web Application
- For more information on how to impersonate original caller from Windows forms application , see How To: Impersonate the Original Caller in WCF calling from Windows Forms

Chapter 10 - Intranet – Web to Remote WCF Using Transport Security (Trusted Subsystem, HTTP)

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5

Scenario

In this scenario, your users have Windows accounts and use a Web client to connect over the intranet to an ASP.NET application on an IIS server. The ASP.NET application makes calls to the WCF Service over HTTP. The business logic called by the WCF Service is backed by a SQL Server data store. The ASP.NET application, the WCF Service and the SQL Server data store are all part of a trusted subsystem. The basic model for this application scenario is shown in the following figure.

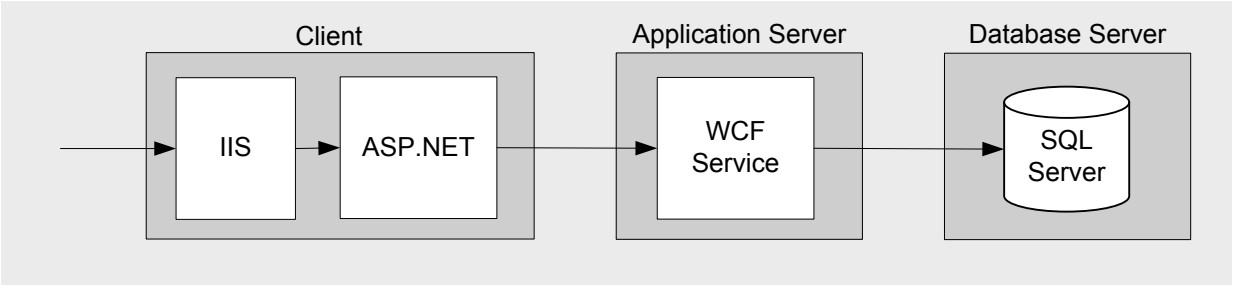


Figure 1. Web to Remote WCF Using Transport Security (Trusted Subsystem, HTTP) – Model

Key Characteristics

This scenario applies to you if:

- Your users have browsers supporting Integrated Windows Authentication.
- Your user accounts are in Active Directory within a domain.
- Your user roles are Windows Groups.
- Your users are accessing the web client from within the domain.
- The business logic behind your WCF Service does not require fine-grained authorization.
- Your ASP.NET application and WCF Service transmit sensitive data over the network that needs to be protected.
- The ability to host the WCF Service in IIS is more important than a high performance connection between the ASP.NET application and the WCF Service.
- Support for interoperability with non WCF clients is more important than a high performance connection between the ASP.NET application and the WCF Service.

Solution

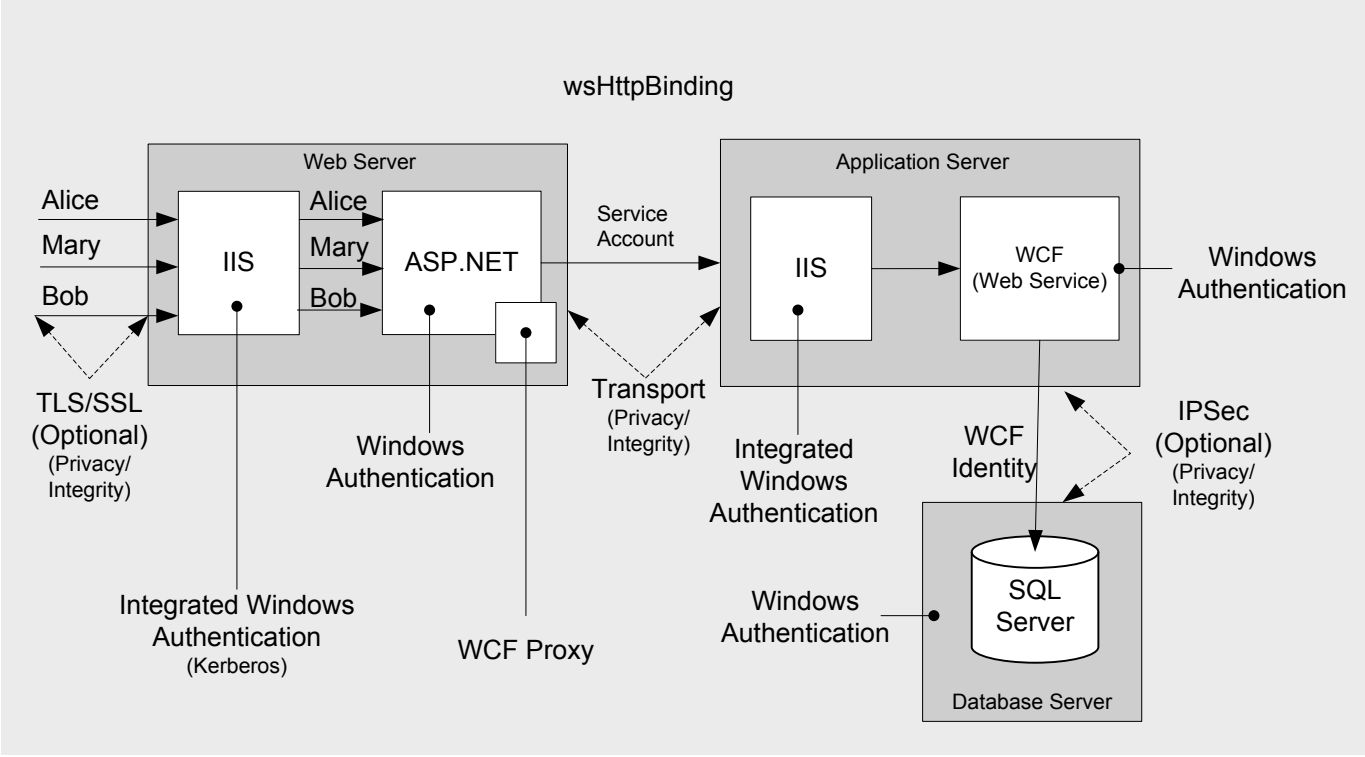


Figure 2. Web to Remote WCF Using Transport Security (Trusted Subsystem, HTTP) – Solution

Solution Summary Table

In this solution you will:

- Use domain credentials to authenticate clients against an Active Directory user store.
- Use a service account to call WCF from the ASP.NET application. The WCF Service uses Windows Authentication.
- Use a service account to call the SQL Server from WCF. The SQL Server uses Windows Authentication.
- Use SSL to protect sensitive data between the Web client and IIS.
- Use Transport security to protect sensitive data between the ASP.NET application and the WCF Service.
- Optionally, use IPSec to protect sensitive data between the WCF Service and SQL Server.
- Use **wsHttpBinding** to provide support for interoperability and allow the service to be hosted in IIS.
- Host WCF in IIS.

Web Server

Checks	Example
IIS - Configuration	
A dedicated application pool is used and configured to run under a custom service account. Use a domain account if possible.	
The Web application is configured to run under the service account. Assign the Web application to the custom application pool.	
Service Principal Name is created if the service account used in the ASP.NET application pool is a custom domain identity. Create an SPN for both the DNS and NETBIOS machine name.	`setspn -a HTTP//WebServer.domain.com customDomainAccount` `setspn -a HTTP//WebServer customDomainAccount`
IIS - Authentication	
The IIS virtual directory is configured to use Windows Integrated Authentication. Users will be authenticated with Windows authentication.	
Anonymous access is disabled.	
ASP.NET - Authentication	
ASP.NET is configured for Windows Integrated Authentication. The Web application will authenticate the users.	`<authentication mode = "Windows" >`
ASP.NET - Authorization	

If you have roles in your application, then use URL authorization.
 	
Use the <location> attribute to configure authorization settings for specific folders. Authorized users have access to specific folders and pages.
	`<authorization>`
	`<allow roles="domainName\RoleName" />`
	`<deny users="*" />`
	`</authorization>`

Role Manager is enabled and Role-checks are performed using Role Manager API.
 	
Original users are authorized using the Windows groups before calling in WCF
	`<roleManager enabled="true"`
	`defaultProvider=`
	`"AspNetWindowsTokenRoleProvider"/>`

Checks	Example
Service.	
WCF Proxy - Configuration	
ASP.NET has a proxy reference to the WCF Service. The application has access to the WCF metadata to create a service reference.	`WCFTestService.MyServiceClient proxy = new WCFTestService.MyServiceClient();`
Proxy invokes services with the security context of the ASP.NET process identity. The proxy will automatically invoke WCF operations using the security context of the service account.	`proxy.GetData("data");` `proxy.Close();`
The Root Ca of the certificate is installed in the Trusted Root Certification Authorities store of the user machine, either in Local Machine or Local User. You need to install the root ca because transport security performs trust chain validation. If the certificate comes from a known issuer, such as Verisign, this is unnecessary.	
WCF Proxy - Caller Identity	

For auditing purposes, the identity of the caller can be passed in custom message headers.
 	
Use transport security to protect against spoofing attacks.
	`using ( OperationContextScope scope = new OperationContextScope(proxy.InnerChannel))`
	`{`
	`string identity = ((WindowsIdentity)HttpContext.Current.User.Identity).Name;`
	
	`MessageHeader<string> headerIdentity = new MessageHeader<string>(identity);`
	
	`MessageHeader untypedMessageHeader = headerIdentity.GetUntypedHeader("identity", "ns");`
	
	`OperationContext.Current.OutgoingMessageHeaders.Add(untypedMessageHeader);`
	`TextBox1.Text = proxy.GetData("data");`
	`}`

Application Server

Checks & More Info	Example
IIS - Configuration	
A dedicated application pool is used and configured to run under a custom service account. Use a domain account if possible.	
The WCF Service is configured to run under the service account. Assign the WCF Service to the custom application pool.	
Service Principal Name is created if the service account used in the ASP.NET application pool of WCF Service is a custom domain identity. Create an SPN for both the DNS and NETBIOS machine name.	`setspn -a HTTP//WCFServer.domain.com customDomainAccount` `setspn -a HTTP//WCFServer customDomainAccount`
Certificate is installed in personal store of LocalMachine. The certificate needs to match the DNS or netbios machine name of the application server.	
Certificate is configured in the Web site of the application. Certificate is configured in the Web site for transport security using SSL.	
The virtual directory is configured to use SSL. SSL is configurable per virtual directory bases.	
The Root CA of the certificate is installed in the Trusted Root Certification Authorities store of the application machine either in Local Machine or Local User. You need to install the Root CA because transport security performs trust chain validation. If the certificate comes from a known issuer, such as Verisign, this is unnecessary.	
IIS - Authentication	
The IIS virtual directory is configured to use Anonymous access.	

Checks & More Info	Example
WCF Service - Configuration	

Configure the WCF Service with wsHttpbinding.
 	
The wsHttpBinding uses the http protocol and provides full support for SOAP security, transactions, reliability and interoperability.
	`<wsHttpBinding>`
	`  <binding name="httpsendpointconfig">`
	`  <security mode="Transport">`
	`  <transport clientCredentialType="Windows" />`
	`  </security>`
	`  </binding>`
	`</wsHttpBinding>`

Service Metadata is configured in service behavior.
httpGetEnabled is disabled and httpsGetEnabled is enabled.
 	
This is required so that client can add reference to the WCF Service using SvcUtil utility.
	`<serviceBehaviors>`
	`  <behavior name="behaviorConfiguration">`
	`  <serviceMetadata httpsGetEnabled="true" />`
	`  </behavior>`
	`</serviceBehaviors>`

WCF Service -Authentication	
The **wsHttpBinding** is configured to use Windows Authentication and transport security. **wsHttpBinding** by default supports Windows Authentication.	`<security mode="Transport">` `<transport clientCredentialType="Windows" />`
WCF Service - Caller Identity	
Service retrieves the identity of the caller from the **operationcontext** for auditing purposes. Use the identity to improve logging and auditing.	`string identity = OperationContext.Current.IncomingMessageHeaders.GetHeader<string>("identity", "ns");`
WCF Service - SQL	
The connection string for database is configured to use Windows Authentication. The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes.	`SqlConnection sqlcon = new SqlConnection("Server=10.3.19.11;Database=Northwind;IntegratedSecurity=SSPI");`
Database connection is opened using the WCF process identity's security context. Service does not impersonate the original caller so SQL Server benefits from connection pooling.	

Database Server

Check & More Info	Example
WCF Service - Configuration	
A SQL Server login is created for the WCF's service account (process identity). This grants access to the SQL Server.	`exec sp_grantlogin 'Custom Service Account'`

The login is mapped to a database user for the Web application.
 	
This grants access to the specified database.
	`use targetDatabase`
	`go`
	`exec sp_grantdbaccess ' Custom Service Account'`
	`go`

A database role is created in the target database.
 	
This allows access control and authorization to the DB.
	`use targetDatabase`
	`go`
	`exec sp_addrole 'DB Role Name'`
	`go`

The login is added to the database role.
 	
Grant minimum permissions. For example, grant execute permissions to selected stored procedures and provide no direct table access.
	`use targetDatabase`
	`go`
	`exec sp_addrolemember 'DB Role Name', 'Custom Service Account'`
	`go`

WCF Service - Authentication	
SQL Server is configured to use Windows authentication.	

Communication Security

What	Check
Browser to Web Server	SSL is used between browser and Web server to protect sensitive data on the wire. Install certificate in the Website. Configure the virtual directory of the Web application to use SSL.
App Server to Database	You can use IPSec or SSL between App server and database server to protect sensitive data on the wire.

Analysis

Web Server

Authentication

- To prevent unauthenticated and unauthorized users from accessing pages, anonymous access is disabled in IIS.
- Integrated Windows authentication is a good choice for this scenario because all users have Windows accounts. One benefit of integrated Windows authentication is preventing the user's password from ever being sent over the network. Additionally, the logon is transparent for the user because Windows uses the current user's logon session.

Authorization

- URL authorization is used to perform role checks against the original caller, and to restrict access to pages or folders based on role permissions.
- All authorization checks are performed in the Web application before calls are made to the WCF Service. The WCF Service trusts the Web application to perform this authorization and does not need to make fine-grained authorization decisions of its own.
- The Roles Manager is a good choice for this scenario because it allows your ASP.NET code to look up users' roles without writing and maintaining custom code.

WCF Proxy

- Because all authentication and authorization is handled in the ASP.NET application, calls into the WCF Service use the ASP.NET process identity’s security context. You don’t need to flow the original caller into the WCF Service.
- If you need to produce audit logs showing what service operations were called by each user, you can pass the identity of the original caller in a custom header.

Configuration

- In order to reduce attack surface and minimize the impact of a compromise, the ASP.NET application on the Web Server runs under the security context of a Service account using least privilages.
- Because HTTPS trusts chain validation, the root certificate authority that issued the certificate for WCF transport security needs to be installed in the trusted root certification authorities store of the local machine in the application server. In a production environment, this is not necessary as the certificate will be issued by a known issuer such as Verisign.

Application Server

Authentication

- In order to authenticate the ASP.NET service when it makes calls on the WCF Service, WCF is configured to use Windows authentication.

Authorization

- Since the WCF Service trusts the ASP.NET application to authorize the user, the WCF Service performs no authorization.

Data Access

- To reduce the risk of database credentials theft, the database connection string is configured to use Windows authentication. This choice avoids storing credentials in files and passing credentials over the network to the database server.
- The WCF Service accesses the database using the WCF process identity. As a result, all calls use the single process account and designated database connection pooling.

Configuration

- This scenario is optimized around interoperability and the ability to host the service in IIS at the expense of transmission performance. For this reason the best binding choice is **wsHttpBinding**. By default **wsHttpBinding** supports Windows authentication with message security.
- Since **wsHttpBinding** is supported by IIS 6.0, the WCF Service is hosted in IIS.
- In order to reduce attack surface and minimize the impact of a compromise, the WCF Service is running under the security context of a Service account using least privileges.
- A metadata exchange (mex) endpoint is exposed with mexHttpsBinding to make it possible for the client to generate a proxy based on the service definition.
- Because HTTPS trusts chain validation, the root certificate authority that issued the certificate for WCF transport security needs to be installed in the trusted root certification authorities store of the local machine in the application server. In a production environment, this is not necessary as the certificate will be issued by a known issuer such as Verisign.

Database Server

SQL Server database user roles are preferred to SQL Server application roles to avoid the various password management and connection pooling issues associated with the use of SQL application roles. Applications activate SQL application roles by calling a built-in stored procedure with a role name and a password. Therefore, the password must be stored securely. Moreover, using SQL application roles forces you to disable database connection pooling, which severely impacts application scalability.

Creating a new user-defined database role and adding the database user to the role allows you to give specific minimum permissions to the role. In this way, if the database account changes you don't have to change the permissions on all database objects.

Communication Security

- Use SSL between the browser and Web Server to protect sensitive data on the wire.
- Use Transport security to protect sensitive data between the Web Server and App Server.
- You can use IPSec or SSL between the App Server and Database Server to protect sensitive data on the wire.

Example

Domain Controller

Configuration

A Service Principle Name (SPN) is created based on these rules:

- If the ASP.NET application runs in an application pool with a custom domain identity, create an SPN, and map the custom domain identity with the HTTP service class and both the DNS machine name and the NETBIOS machine name:
```
setspn -a HTTP//WebServer.domain.com customDomainAccount
setspn -a HTTP//WebServer customDomainAccount
```

- If the WCF application runs in an application pool with a custom domain identity, create an SPN and map the custom domain identity with the HTTP service class and both the DNS machine name and the NETBIOS machine name:
```
setspn -a HTTP//WCFServer.domain.com customDomainAccount
setspn -a HTTP//WCFServer customDomainAccount
```

Web Server

Code

- Role-authorization occurs before WCF Service invocation.
- Identity of the original caller is retrieved from the HttpContext.
- Message Header containing the caller identity is created and passed to the operation context for auditing purposes.

```
using System.Security.Principal;
using System.ServiceModel;
using System.ServiceModel.Channels;
…
```

```
protected void Button1_Click(object sender, EventArgs e)
{
 if (Roles.IsUserInRole(@"npscode\Accounting"))
 {
  WCFTestclient.MyServiceClient proxy =
          new WCFTestclient.MyServiceClient();

 using ( OperationContextScope scope =
          new OperationContextScope(proxy.InnerChannel))
  {
   string identity =
      ((WindowsIdentity)HttpContext.Current.User.Identity).Name;
   MessageHeader<string> headerIdentity =
       new MessageHeader<string>(identity);
   MessageHeader untypedMessageHeader =
       headerIdentity.GetUntypedHeader("identity", "ns");

OperationContext.Current.OutgoingMessageHeaders.Add(untypedMessageHeader);
          proxy.GetData("data");
   }

    proxy.Close();
  }
}
```

Configuration

- Windows authentication is enabled.
- URL authorization role check is enabled.
- Role Manager is enabled.

```
<system.web>
   <assemblies>
     <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
     <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
     <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
     <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
   </assemblies>

   <authentication mode="Windows" />
      <authorization>
        <allow roles="npscode\BusinessDivision" />
        <deny users="*" />
      </authorization>

      <roleManager enabled="true"
             defaultProvider= "AspNetWindowsTokenRoleProvider"/>

      <pages>
        <controls>
```

```
          <add tagPrefix="asp" namespace="System.Web.UI"
               assembly="System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>

          <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
               assembly="System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>

        </controls>
      </pages>

   <httpHandlers>
      <remove verb="*" path="*.asmx"/>
      <add verb="*" path="*.asmx" validate="false"
           type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>




      <add verb="*" path="*_AppService.axd" validate="false"
         type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>

      <add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
validate="false"/>

   </httpHandlers>

   <httpModules>
      <add name="ScriptModule"
           type="System.Web.Handlers.ScriptModule, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
   </httpModules>

</system.web>
```

Application Server

Code

- The service retrieves the identity of the caller from the operation context if it is required for auditing purposes.
- The service calls SQL using the security context of the WCF Service.

```
using System.Data.SqlClient;
public string GetData(string myValue)
{
 SqlConnection sqlcon = new
  SqlConnection("Server=SqlServer;Database=testdb;Integrated Security=SSPI");
```

```
 sqlcon.Open();

 //do the business operation

 string identity =
OperationContext.Current.IncomingMessageHeaders.GetHeader<string>("identity",
"ns");

 return “some data” ;
}
```

Configuration

- The service has a binding endpoint that uses **wsHttpbinding** with a binding configuration to use Windows authentication and transport security.
- The service has a service behavior configuration to publish metadata.
- The service behavior is configured with element serviceMedata to allow metadata exposure.

```
<system.serviceModel>
  <bindings>
    <wsHttpBinding>
      <binding name="httpsendpointconfig">
        <security mode="Transport">
          <transport clientCredentialType="Windows" />
        </security>
      </binding>
    </wsHttpBinding>
  </bindings>

  <client/>
  <services>
     <service behaviorConfiguration="behaviorConfiguration"
              name="MyService">

        <endpoint binding="wsHttpBinding"
                  bindingConfiguration="httpsendpointconfig"
                  name="httpsendpoint"
                  contract="IMyService2"/>
     </service>
  </services>

  <behaviors>
     <serviceBehaviors>
       <behavior name="behaviorConfiguration">
          <serviceMetadata httpsGetEnabled="true" />
       </behavior>
     </serviceBehaviors>
  </behaviors>

</system.serviceModel>
```

Database Server

Configuration

- A SQL server login is created for the WCF Service account.
- The WCF login name is given access to the database.
- The role is created in the database.
- The WCF login name is added to the role.

```
-- Create a SQL Server login  that matches the WCF machine name
EXEC SP_GRANTLOGIN 'npscode\perfpres02$'

-- Grant the login access to the application database
use testdb
go
exec sp_grantdbaccess 'npscode\perfpres02$'

-- Create the new database role
use testdb
go
exec sp_addrole 'myrole2','db_owner'

-- Add the new login to the role
use testdb
go
exec sp_addrolemember 'myrole2','npscode\aspnethost'
```

Additional Resources

- For more information on WCF Transport Layer Security using **wsHttpBinding** and SSL, see How To: Use wsHttpBinding with Windows Authentication and Transport Security in WCF Calling from Windows Forms
- For more information on how to work with temporary certificates, see “How to: Create Temporary Certificates for Use During Development” at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on how to view certificates by using the Microsoft Management Console (MMC) snap in, see “How to: View Certificates with the MMC Snap-in” at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see “Differences Between Service Certificate Validation Done by Internet Explorer and WCF” at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see “Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security” at http://msdn2.microsoft.com/en-us/library/aa702579.aspx

Chapter 11 - Intranet – Web to Remote WCF Using Transport Security (Trusted Subsystem, TCP)

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5

Scenario

In this scenario, your users have Windows accounts and use a Web client to connect over the intranet to an ASP.NET application on an IIS server. The ASP.NET application makes calls to the WCF Service over TCP. The business logic called by the WCF Service is backed by a SQL Server data store. The ASP.NET application, the WCF Service and the SQL Server data store are all part of a trusted subsystem. The basic model for this application scenario is shown in the following figure.

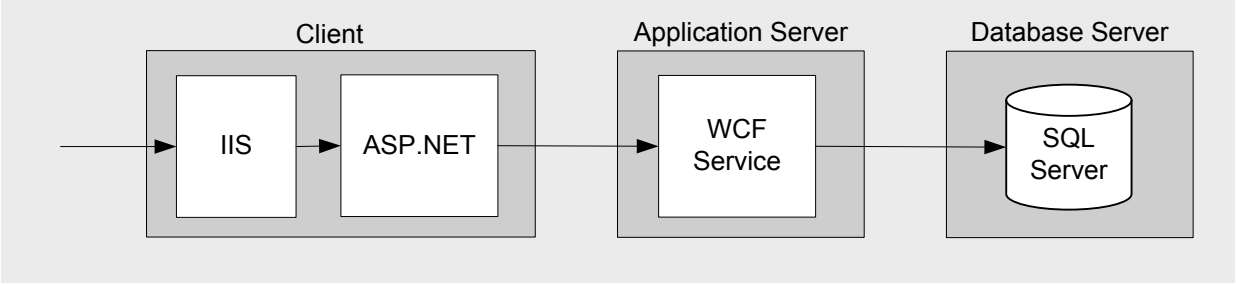


Figure 1. Web to Remote WCF Using Transport Security (Trusted Subsystem, TCP) – Model

Key Characteristics

This scenario applies to you if:

- Your users have browsers supporting Integrated Windows Authentication.
- Your user accounts are in Active Directory within a domain.
- Your user roles are Windows Groups.
- Your users are accessing the web client from within the domain
- The business logic behind your WCF Service does not require fine-grained authorization.
- Your ASP.NET application and WCF Service transmit sensitive data over the network that needs to be protected.
- A high-performance connection between the ASP.NET application and the WCF Service is more important than the ability to host the WCF Service in IIS.
- A high performance connection between the ASP.NET application and the WCF Service is more important than to provide interoperability support for non WCF clients.

Solution

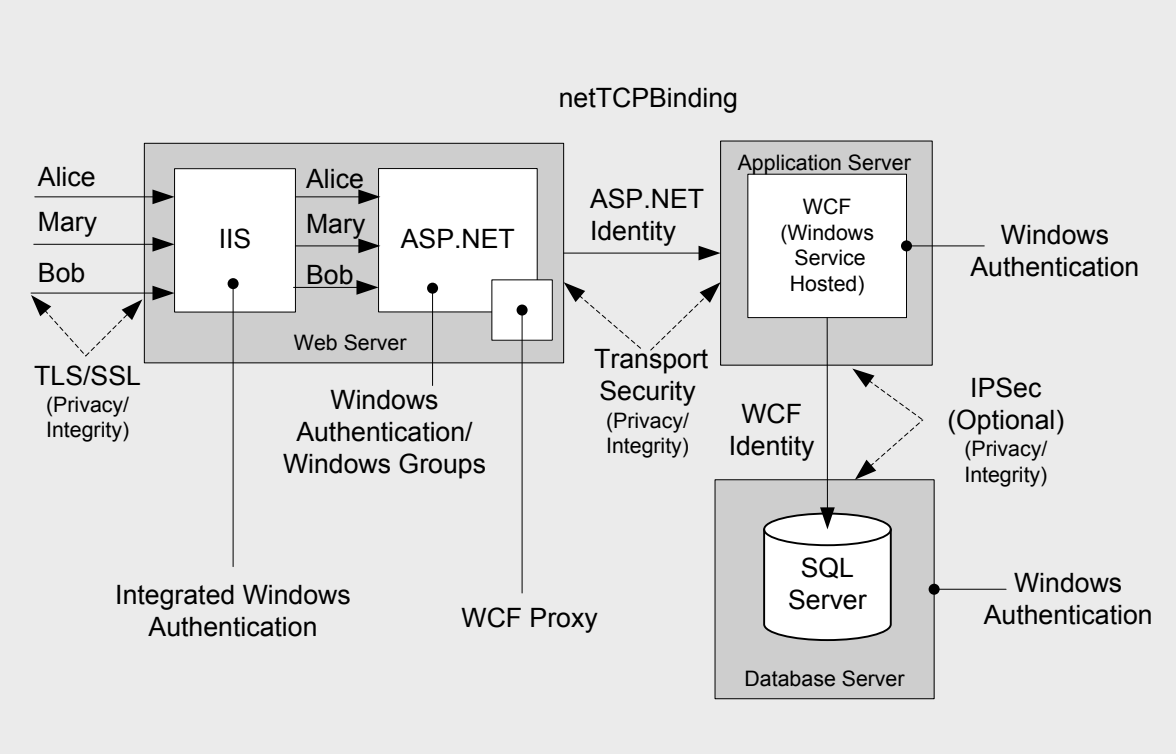


Figure 2. Web to Remote WCF Using Transport Security (Trusted Subsystem, TCP) – Solution

Solution Summary Table

In this solution you will:

- Use domain credentials to authenticate clients against an Active Directory user store.
- Use a service account to call WCF from the ASP.NET application. The WCF Service uses Windows Authentication.
- Use a service account to call the SQL Server from WCF. The SQL Server uses Windows Authentication.
- Use SSL to protect sensitive data between the Web client and IIS.
- Use Transport security to protect sensitive data between the ASP.NET application and the WCF Service.
- Optionally, use IPSec to protect sensitive data between the WCF Service and SQL Server.
- Use **netTcpBinding** to support the TCP transport for improved performance.
- Host WCF in a Windows Service since IIS 6.0 does not support the TCP transport.

Web Server

Checks & More Info	Example
IIS - Configuration	
A dedicated application pool is used and configured to run under a custom service account. Use a domain account if possible.	
The Web application is configured to run under the service account. Assign the Web application to the custom application pool.	
A Service Principal Name is created if the service account used in the ASP.NET application pool is a custom domain account. Create an SPN for both the DNS and NETBIOS machine name.	`setspn -a HTTP//WebServer.domain.com customDomainAccount` `setspn -a HTTP//WebServer customDomainAccount`
IIS - Authentication	
The IIS virtual directory is configured to use Windows Integrated Authentication. Users will be authenticated with Windows authentication.	
Anonymous access is disabled.	

Checks & More Info	Example
ASP.NET - Authentication	
ASP.NET is configured for Windows Integrated Authentication. The Web application will authenticate the users.	`<authentication mode = "Windows" >`
ASP.NET - Authorization	

If you have roles in your application, use URL authorization.
 	
Use the <location> attribute to configure authorization settings for specific folders. The authorized users have access to specific foders and pages.
	`<authorization>`
	`  <allow roles="domainName\RoleName" />`
	`  <deny users="*" />`
	`</authorization>`

Role manager is enabled and Role-checks are performed using a role manager API.	`<roleManager enabled="true"`
	`  defaultProvider=`
	`  "AspNetWindowsTokenRoleProvider"/>`

Checks & More Info	Example
Original users are authorized using the Windows groups before calling in WCF Service.	

Checks & More Info	Example
WCF Proxy - Configuration	
ASP.NET has a proxy reference to the WCF Service. The application has access to the WCF metadata to create a service reference.	`WCFTestService.MyServiceClient proxy = new WCFTestService.MyServiceClient();`
Proxy invokes services with the security context of the ASP.NET process identity. The proxy will automatically invoke WCF operations using the security context of the service account.	`proxy.GetData("data");` `proxy.Close();`
The Root Ca of the certificate is installed in the Trusted Root Certification Authorities store of the user machine, either in Local Machine or Local User. You need to install the root ca because transport security performs trust chain validation. If the certificate comes from a known issuer, such as Verisign, this is unnecessary.	
WCF Proxy - Caller Identity	

For auditing purposes, the identity of the caller can be passed in custom message headers.
 	
Use transport security to protect against spoofing attacks.
	`using ( OperationContextScope scope = new OperationContextScope(proxy.InnerChannel))`
	`{`
	`string identity = ((WindowsIdentity)HttpContext.Current.User.Identity).Name;`
	
	`MessageHeader<string> headerIdentity = new MessageHeader<string>(identity);`
	
	`MessageHeader untypedMessageHeader = headerIdentity.GetUntypedHeader("identity", "ns");`
	
	`OperationContext.Current.OutgoingMessageHeaders.Add(untypedMessageHeader);`
	
	`TextBox1.Text = proxy.GetData("data");`
	`}`

Application Server

Checks & More Info	Example
Windows Service - Configuration	
Windows Service is configured to run under a custom domain service account. Use a domain account if possible.	
WCF Service is hosted in Windows Service. Since IIS does not support **netTcpBinding**, host in Windows Service.	
WCF Service - Configuration	
Configure the WCF Service to use **netTcpBinding**. The **NetTcpBinding** uses the TCP protocol and provides full support for SOAP security, transactions, and reliability. As client and WCF Service both are in the intranet, this is a good choice from a performance perspective.	`<service behaviorConfiguration="BehaviorConfiguration" name="WCFServicecHost.MyService"> <endpoint address="" binding="netTcpBinding" bindingConfiguration="" name="TcpBinding" contract="WCFServicecHost.IMyService" /> <host> <baseAddresses> <add baseAddress="net.tcp://WCFApp01.npscode.com/MyService" /> </baseAddresses> </host> </service>`
A mex endpoint is created for publishing the metadata. This is required so that client can add a reference to the WCF Service using **SvcUtil utility**.	`<services> <service behaviorConfiguration="BehaviorConfiguration" name="WCFServicecHost.MyService"> <endpoint address="Mex" binding="mexTcpBinding" bindingConfiguration="" name="MexEndpoint" contract="IMetadataExchange" /> </service>`
Service Metadata is configured in service behavior. The service metadata entry is required for the Windows Service host to start. The HTTP and HTTPS get are disabled.	`<behaviors> <serviceBehaviors> <behavior name="BehaviorConfiguration"> <serviceMetadata /> </behavior> </serviceBehaviors> </behaviors>`
WCF Service - Authentication	

Checks & More Info	Example
The **netTcpBinding** is configured to use Windows Authentication and Transport Security. **netTcpBinding** by default supports Windows Authentication and Transport Security.	`<endpoint address="" binding="netTcpBinding" bindingConfiguration="" />`
WCF Service - Caller Identity	
Service retrieves the identity of the caller from the operationcontext for auditing purposes. Use the identity to improve logging and auditing.	`string identity = OperationContext.Current.IncomingMessageHeaders.GetHeader<string>("identity", "ns");`
WCF Service - SQL	
The connection string for database is configured to use Windows Authentication. The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes.	`SqlConnection sqlcon = new SqlConnection("Server=10.3.19.11;Database=Northwind;IntegratedSecurity=SSPI");`
Database connection is opened using the WCF process identity's security context. Service does not impersonate the original caller so SQL Server benefits from connection pooling.	

Database Server

Check & More Info	Example
Configuration	
A SQL Server login is created for the WCF's service account (process identity). This grants access to the SQL Server.	`exec sp_grantlogin 'Custom Service Account'`

Check & More Info	Example

The login is mapped to a database user for the Web application.
 	
This grants access to the specified database.
	`use targetDatabase`
	`go`
	`exec sp_grantdbaccess ' Custom Service Account'`
	`go`

A database role is created in the target database.
 	
This allows access control and authorization to the DB.
	`use targetDatabase`
	`go`
	`exec sp_addrole 'DB Role Name'`
	`go`

The login is added to the database role.
 	
Grant minimum permissions. For example, grant execute permissions to selected stored procedures and provide no direct table access.
	`use targetDatabase`
	`go`
	`exec sp_addrolemember 'DB Role Name', 'Custom Service Account'`
	`go`

Authentication	
SQL Server is configured to use Windows Authentication.	

Communication Security

What	Check
Browser to Web Server	SSL is used between the browser and Web server to protect sensitive data on the wire. Install certificate in the Web site. Configure the virtual directory of the Web application to use SSL.
App Server to Database	IPSec or SSL can be used between App server and database server to protect sensitive data on the wire.

Analysis

Web Server

Authentication

- To prevent unauthenticated and unauthorized users from accessing pages, IIS disables anonymous access.

- Integrated Windows Authentication is a good choice for this scenario because all users have Windows accounts. One benefit of Integrated Windows Authentication is preventing the user's password from ever being sent over the network. Additionally, the logon is transparent for the user because Windows uses the current user's logon session.

Authorization

- URL authorization performs role checks against the original caller and restricts access to pages or folders based on role permissions.
- All authorization checks are performed in the Web application before calls are made to the WCF Service. The WCF Service trusts the Web application to perform this authorization and does not need to make fine-grained authorization decisions of its own.
- The Roles Manager is a good choice for this scenario because it allows your service code to look up users' roles without writing and maintaining custom code.

WCF Proxy

- Because you are taking care of all authentication and authorization in the ASP.NET application, all calls through the WCF proxy and into the WCF Service use the ASP.NET process identity’s security context. You don’t need to flow the original caller into the WCF Service.
- If you need to produce audit logs showing what service operations were called by each user, you can pass the identity of the original caller in a custom header.

Configuration

- In order to reduce attack surface and minimize the impact of a compromise, the ASP.NET application on the Web Server runs under the security context of a Service account using least privileges.

Application Server

Authentication

- In order to authenticate the ASP.NET service when it makes calls on the WCF Service, WCF is configured to use Windows Authentication.

Authorization

- Since the WCF Service trusts the ASP.NET application to authorize the user, the WCF Service performs no authorization.

Data Access

- To reduce the risk of database credentials theft, the database connection string is configured to use Windows authentication. This choice avoids storing credentials in files and passing credentials over the network to the database server.
- The WCF Service accesses the database using the WCF process identity. As a result, all calls use the single process account and enables database connection pooling.

Configuration

- This scenario is optimized around transmission performance at the expense of interoperability with clients that expect a legacy Web service and the ability to host the service in IIS. For this reason, the best binding choice is **netTcpBinding**. By default, **netTcpBinding** supports Windows Authentication with Transport security.
- Because **netTcpBinding** is not supported by IIS 6.0, the WCF Service is hosted in a Windows service.
- In order to reduce attack surface and minimize the impact of a compromise, the Windows Service is running under the security context of a Service account using least privileges.
- A metadata exchange (mex) endpoint is exposed so the client can use svcutil to generate a proxy based on the service definition.

Database Server

- SQL Server database user roles are preferred to SQL server application roles to avoid the assorted password management and connection pooling issues associated with the use of SQL application roles. Applications activate SQL application roles by calling a built-in stored procedure with a role name and a password. Therefore, the password must be stored securely. Moreover, using SQL application roles forces you to disable database connection pooling, which severely impacts application scalability.
- Creating a new user-defined database role and adding the database user to the role lets you give specific minimum permissions to the role. In this way, if the database account changes you don't have to change the permissions on all database objects.

Communication Security

- SSL is used between browser and Web server to protect sensitive data on the wire.
- Transport security is used to protect sensitive data between the Web Server and App Server.
- IPSec or SSL can be used between the App Server and Database Server to protect sensitive data on the wire.

Example

Domain Controller

Configuration

Create a service principle name (SPN) based on these rules:

1. If the ASP.NET application runs in an application pool with a custom domain identity, create an SPN and map the custom domain identity with the HTTP service class and both the DNS machine name and the NETBIOS machine name:
```
setspn -a HTTP//WebServer.domain.com customDomainAccount
setspn -a HTTP//WebServer customDomainAccount
```

2. If the service runs under a custom domain identity:
```
setspn -a ServiceNameofWcfService//WebServer.domain.com
customDomainAccount
setspn -a ServiceNameofWcfService//WebServer customDomainAccount
```

3. If the service runs under the network service account:
```
setspn -a ServiceNameofWcfService//WebServer.domain.com computerName
setspn -a ServiceNameofWcfService//WebServer computerName
```

Note: You should specify the service name as it is displayed in the MMC services console.

Web Server

Code

- Role-authorization occurs before WCF Service invocation.
- ASP.NET calls WCF Service if it is authorized.
- Identity of the original caller is retrieved from the HttpContext.
- Message Header containing the caller identity is created and passed to the operation context for auditing purposes.

```
using System.Security.Principal;
using System.ServiceModel;
using System.ServiceModel.Channels;
…
protected void Button1_Click(object sender, EventArgs e)
{
  if (Roles.IsUserInRole(@"npscode\Accounting"))
  {
    WCFTestclient.MyServiceClient proxy
        = new WCFTestclient.MyServiceClient();
    using ( OperationContextScope scope
        = new OperationContextScope(proxy.InnerChannel))
    {
      string identity
        = ((WindowsIdentity)HttpContext.Current.User.Identity).Name;
```

```
      MessageHeader<string> headerIdentity
        = new MessageHeader<string>(identity);
      MessageHeader untypedMessageHeader
        = headerIdentity.GetUntypedHeader("identity", "ns");

 OperationContext.Current.OutgoingMessageHeaders.Add(untypedMessageHeader);
               proxy.GetData("data");
     }
     proxy.Close();

   } // endif
}
```

Configuration

- Windows authentication is enabled.
- URL authorization role-check is enabled.
- Role Manager is enabled.

```
   <system.web>
     <assemblies>

       <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
   PublicKeyToken=B77A5C561934E089"/>
       <add assembly="System.Web.Extensions, Version=3.5.0.0,
   Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
       <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0,
   Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
       <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral,
   PublicKeyToken=B77A5C561934E089"/>

     </assemblies>

     <authentication mode="Windows" />

     <authorization>
       <allow roles="npscode\BusinessDivision" />
       <deny users="*" />
     </authorization>

     <roleManager enabled="true"
                  defaultProvider= "AspNetWindowsTokenRoleProvider"/>
        <pages>
           <controls>
             <add tagPrefix="asp"
                  namespace="System.Web.UI"
                  assembly="System.Web.Extensions, Version=3.5.0.0,
   Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>

             <add tagPrefix="asp"
                  namespace="System.Web.UI.WebControls"
                  assembly="System.Web.Extensions, Version=3.5.0.0,
   Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
           </controls>
```

```
      </pages>

      <httpHandlers>
         <remove verb="*" path="*.asmx"/>

         <add verb="*"
              path="*.asmx"
              validate="false"
              type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>

         <add verb="*"
              path="*_AppService.axd"
              validate="false"
              type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>

         <add verb="GET,HEAD"
              path="ScriptResource.axd"
              type="System.Web.Handlers.ScriptResourceHandler,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" validate="false"/>

      </httpHandlers>

       <httpModules>
         <add name="ScriptModule"
              type="System.Web.Handlers.ScriptModule,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
       </httpModules>

</system.web>
```

Application Server

Code

- The service retrieves the identity of the caller from the operation context if it is required for auditing purposes.
- The service calls SQL using the security context of the WCF Service.

```
using System.Data.SqlClient;

public string GetData(string myValue)
{
   SqlConnection sqlcon
      = new SqlConnection("Server=SqlServer;Database=testdb;Integrated
Security=SSPI");

   sqlcon.Open();
```

```
    //do the business operation

    string identity =
OperationContext.Current.IncomingMessageHeaders.GetHeader<string>("identit
y", "ns");

    return “some data” ;
}
```

Configuration

- The service has a binding endpoint that uses **netTcpbinding** with the default settings.
- The service has a service behavior configuration to publish metadata.
- The service has a base address configured.
- The service behavior is configured with element serviceMedata to allow metadata exposure.

```
<system.serviceModel>
    <behaviors>
        <serviceBehaviors>
            <behavior name="BehaviorConfiguration">
                <serviceMetadata />
            </behavior>
        </serviceBehaviors>
    </behaviors>

    <bindings />

    <services>
      <service behaviorConfiguration="BehaviorConfiguration"
               name="WCFServicecHost.MyService">
         <endpoint address="Mex"
                   binding="mexTcpBinding"
                   bindingConfiguration=""
                   name="MexEndpoint"
                   contract="IMetadataExchange" />

         <endpoint address=""
                   binding="netTcpBinding"
                   bindingConfiguration=""
                   name="TcpBinding"
                   contract="WCFServicecHost.IMyService" />
            <host>
             <baseAddresses>
               <add
                baseAddress="net.tcp://perfpres02.npscode.com/MyService" />
             </baseAddresses>
            </host>
```

```
        </service>
      </services>

 </system.serviceModel>
```

Database Server

Configuration

- A SQL server login is created for the WCF Service account.
- The WCF login name is given access to the database.
- The role is created in the database.
- The WCF login name is added to the role.

```
-- Create a SQL Server login  that matches the WCF machine name
EXEC SP_GRANTLOGIN 'npscode\perfpres02$'

-- Grant the login access to the application database
use testdb
go
exec sp_grantdbaccess 'npscode\perfpres02$'

-- Create the new database role
use testdb
go
exec sp_addrole 'myrole2','db_owner'

-- Add the new login to the role
use testdb
go
exec sp_addrolemember 'myrole2','npscode\aspnethost'
```

Additional Resources

- For more information on security authentication best practices, see “Best Practices for Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms731059.aspx
- For additional information on message security, see “Message Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For more information on hosting in a Windows service, see “How to: Host a WCF Service in a Managed Windows Service.”
- For more information on WCF hosing considerations, see “Hosting Services” at http://msdn2.microsoft.com/en-us/library/ms730158.aspx
- For more information on netTcpBinding configuration options see “<netTcpBinding>” at http://msdn2.microsoft.com/en-us/library/ms731343.aspx
- For more information on how use netTcpbinding with windows authentication, see to How To: Use netTcpBinding with Windows Authentication and Transport Security in WCF from Windows Forms

Chapter 12 - Intranet – Windows Forms to Remote WCF Using Transport Security (Original Caller, TCP)

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5

Scenario

In this scenario, your users have Windows accounts and use a Windows Forms client to connect over the intranet to your WCF Service. The business logic called by the WCF Service is backed by a Microsoft SQL Server® data store. The following figure illustrates the basic model for this application scenario.

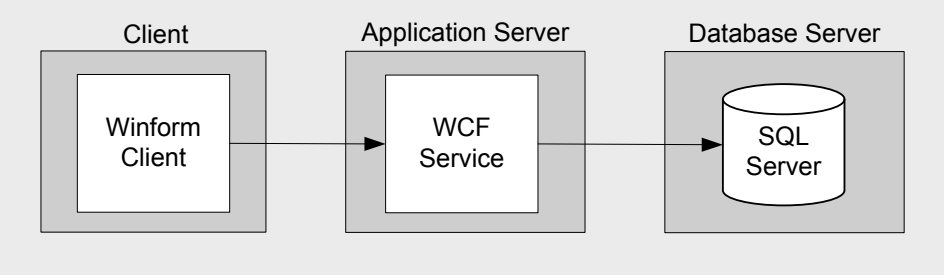


Figure 1. Windows Forms to Remote WCF Using Transport Security (Original Caller, TCP) – Model

Key Characteristics

This scenario applies to you if:

- Your users have Windows Forms clients.
- Your user accounts are in Active Directory within a domain.
- Your user roles are Windows Groups.
- Your application transmits sensitive data over the network and needs to be protected.

Solution

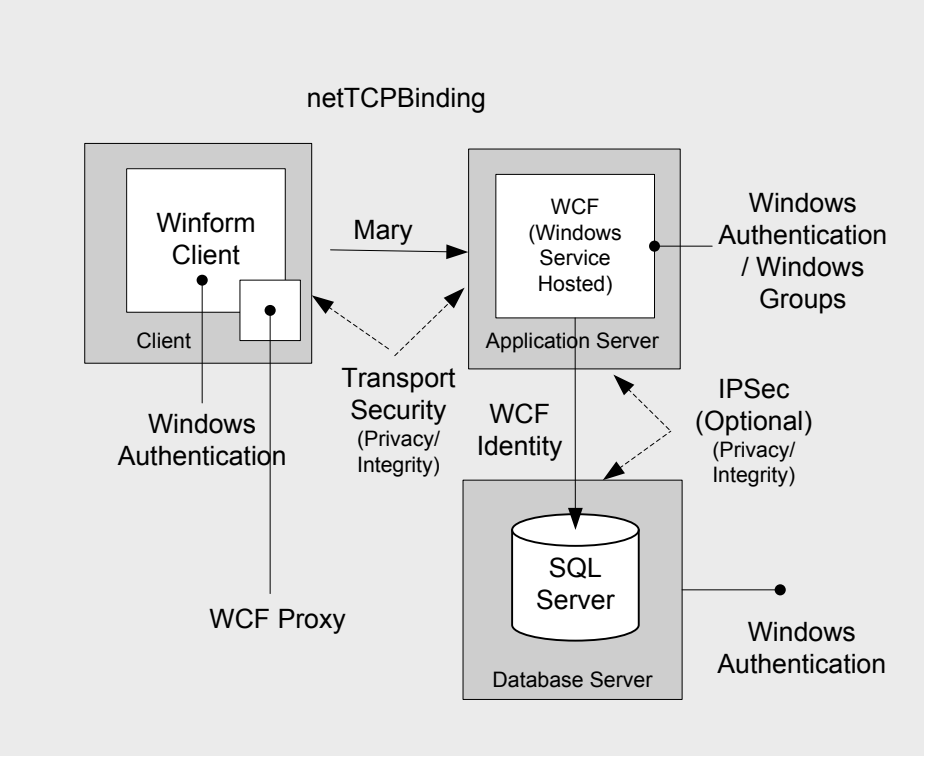


Figure 2. Windows Forms to Remote WCF Using Transport Security (Original Caller, TCP) – Solution

Solution Summary Table

In this solution you will:

- Use domain credentials to authenticate clients against an Active Directory user store.
- Use a service account to call the SQL Server from WCF.
- Use transport security to protect sensitive data between the Windows Forms client and the WCF Service.
- Use netTcpBinding to support the TCP transport for improved performance.
- Host WCF in a Windows Service since IIS does not support the TCP transport.

Thick Client

Checks	Example
WCF Proxy	
Application has a proxy reference to the WCF Service. The application has access to the WCF	`WCFTestService.MyServiceClient proxy = new WCFTestService.MyServiceClient();`

metadata to create a service reference.	
Proxy invokes services with the security context of the logon user. The proxy will automatically invoke WCF operations using the security context of the current user.	`proxy.GetData("data");` `proxy.Close();`

Application Server

Check & More Info	Example
Windows Service - Configuration	
Windows Service is configured to run under a custom domain service account. Use a domain account if possible.	
WCF Service is hosted in a Windows Service. Since IIS does not support netTcpBinding, host in a Windows service instead.	

Check & More Info	Example
WCF Service - Configuration	

Configure the WCF Service to use netTcpbinding. The base address is provided in the configuration section.
 	
NetTcpBinding uses the TCP protocol and provides full support for SOAP security, transactions, and reliability. As client and WCF Service both are in intranet this is a good choice from a performance perspective.
	`<endpoint`
	`address=""`
	`binding="netTcpBinding"`
	`bindingConfiguration=""`
	`name="TcpBinding"`
	`contract=`
	`"WCFServicecHost.IMyService"/>`
	
	`<baseAddresses>`
	`<add baseAddress=`
	`"net.tcp://WCFApp01.npscode.com/MyService" />`
	
	`</baseAddresses>`

Check & More Info	Example

A mex endpoint is created for publishing the metadata.
 	
This is required so that client can add reference to the WCF Service using SvcUtil utility.
	`<endpoint`
	`  address="Mex"`
	`  binding="mexTcpBinding"`
	`  bindingConfiguration=""`
	`  name="MexEndpoint"`
	`  contract="IMetadataExchange"`
	`/>`

Service Metadata is configured in service behavior. The service metadata entry is required for the windows service host to start. HTTP and HTTPS get are disabled.	`<serviceMetadata />`
WCF Service - Authentication	

The netTcpBinding is configured to use Windows Authentication and Transport security.
 	
By default netTcpBinding is configured to use Windows Authentication and Transport Security.
	`<endpoint`
	`  address=""`
	`  binding="netTcpBinding"`
	`  bindingConfiguration="" />`

WCF Service - Authorization	

Role Manager feature is enabled and WindowsTokenRoleProvider is configured for roles authorization.
 	
Roles authorization can be performed declaratively or imperatively in the operation contract.
	`<serviceAuthorization`
	`  principalPermissionMode`
	`  ="UseAspNetRoles"`
	`  roleProviderName`
	`="AspNetWindowsTokenRoleProvider" />`

Perform Role-checks declaratively using Windows Identity Token, for checking Active Directory group membership.
 	
Declarative role-checks on operations is the preferred mechanism.
	`[PrincipalPermission(SecurityAction.Demand,`
	`Role = "npscode\\accounting")]`
	
	`public string GetData(string message)`
	`{`
	`  return "hello";`
	`}`

Check & More Info	Example

Perform Role-checks imperatively using Windows Identity Token, for checking Active Directory group membership.
 	
If you need finer-grained authorization control, you can use imperative role checks in the code itself. Use call to Roles.IsUserInRole to perform the check.
	`public string GetData(string myValue)`
	`{`
	`if(Roles.IsUserInRole(@"npscode\Accounting"))`
	`  {`
	`   //Do something for Accounting role`
	`  }`
	`else`
	`  {`
	`   //Do something for non-accounting role or throw an error`
	`   }`
	`}`

WCF Service - SQL	
The connection string for the database is configured to use Windows authentication. The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes.	`SqlConnection sqlcon = new SqlConnection("Server=10.3.19.11;Database=Northwind;IntegratedSecurity=SSPI");`
Database connection is opened using the WCF process identity's security context. Service does not impersonate the original caller to benefit for connection pooling.	

Database Server

Check & More Info	Example
Configuration	
A SQL Server login is created for the WCF's service account (process identity). This grants access to the SQL Server.	`exec sp_grantlogin 'Custom Service Account'`

The login is granted access to the target database.
 	
This grants access to the specified database.
	`use targetDatabase`
	`go`
	`exec sp_grantdbaccess 'Custom Service Account'`
	`go`

A database role is created in the target database.
 	
This allows access control and authorization to the DB.
	`use targetDatabase`
	`go`
	`exec sp_addrole 'DB Role Name'`
	`go`

Check & More Info	Example

The login is added to the database role.
 	
Grant minimum permissions. For example, grant execute permissions to selected stored procedures and provide no direct table access.
	`use targetDatabase`
	`go`
	`exec sp_addrolemember 'DB Role Name', 'Custom Service Account'`
	`go`

Authentication	
SQL Server is configured to use Windows authentication.	

Communication Security

What	Check
App server to Database	You can use IPSec or SSL between App server and database server to protect sensitive data on the wire.

Analysis

Thick Client

WCF Proxy

- Because WCF requires the original user's credentials for Authentication and Authorization, the original user's security context makes all calls through the WCF proxy and into the WCF Service.

Application Server

Authentication

- In order to authenticate the original users when the Thick Client makes calls on the WCF Service, WCF is configured to use Windows Authentication.

Authorization

- For coarse grained access control, authorization checks are performed declaratively in the WCF Service at the operation level.

- For fine grained access control, authorization checks are performed programmatically within the operations.

- The Roles Manager is a good choice for this scenario because it allows your service code to look up users' roles without writing and maintaining custom code.

Data Access

- To reduce the risk of stolen database credentials, the database connection string is configured to use Windows authentication. This avoids storing credentials in files and passing credentials over the network to the database server.

- The WCF Service accesses the database using the WCF process identity. As a result, all calls use the single process account and the designated database connection pooling.

Configuration

- This scenario is optimized around transmission performance at the expense of interoperability with clients that expect a legacy Web service and the ability to host the service in IIS. For this reason the best binding choice is **netTcpBinding**. By default, **netTcpBinding** supports Windows Authentication with Transport security.

- Because **netTcpBinding** is not supported by IIS 6.0, the WCF Service is hosted in a Windows service.

- In order to reduce attack surface and minimize the impact of a compromise, the Windows service is running under the security context of the Service account using a least privileged account.

- In order to make it possible for the client to generate a proxy based on the service definition, we've exposed a metadata exchange (mex) endpoint.

Database Server

- SQL Server database user roles are preferred to SQL Server server application roles to avoid the associated password management and connection pooling issues associated with the use of SQL application roles. Applications activate SQL application roles by calling a built-in stored procedure with a role name and a password. Therefore, the password must be stored securely. Database connection pooling must also be disabled when you use SQL application roles, which severely impacts application scalability.

- Creating a new user-defined database role, and adding the database user to the role, lets you give specific minimum permissions to the role. Therefore, if the database account changes you don't have to change the permissions on all database objects.

Communication Security

- Transport security protects sensitive data between Thick Client and WCF Service.
- You can use IPSec or SSL between WCF Service and the database server to protect sensitive data on the wire.

Example

Application Server

Code

The service performs imperative authorization checks calling Roles.IsUserInRole.
The service calls SQL using windows authentication.

```
using System.Data.SqlClient;
using System.Web.Security;

public string GetData(string myValue)
{
  if(Roles.IsUserInRole(@"npscode\Accounting"))
  {
    SqlConnection sqlcon = new
SqlConnection("Server=10.3.19.11;Database=Northwind;IntegratedSecurity=SSPI")
;
    sqlcon.Open();
    //do the business operation
    return "Authorization succeeded ";
  }
  else return "authorization failure";
}
```

Configuration

- The service has a binding endpoint that uses **netTcpbinding** with no binding configuration and windows authentication (default settings).
- The service has a mex endpoint to publish metadata.
- The service has a service behavior configuration to expose the role provider to the WCF Service.
- The service has a base address configured to reduce the size of the binding addresses in the config.
- The service configuration file has an entry for the A**spNetWindowsTokenRoleProvider** under system.web to define which role provider is being used.
- The service behavior is configured with the element **serviceAuthorization** to allow **WindowsTokenRoleProvider** as the authorization provider.
- The service behavior is configured with the element **serviceMetadata** to allow metadata exposure.

```
<system.web>
    <roleManager enabled="true"
             defaultProvider="AspNetWindowsTokenRoleProvider" />
</system.web>

<system.serviceModel>
  <behaviors>
     <serviceBehaviors>
        <behavior name="BehaviorConfiguration">
```

```
          <serviceAuthorization principalPermissionMode="UseAspNetRoles"
                      roleProviderName="AspNetWindowsTokenRoleProvider" />
                  <serviceMetadata />
       </behavior>
     </serviceBehaviors>
  </behaviors>

  <bindings />

  <services>
    <service behaviorConfiguration="BehaviorConfiguration"
             name="WCFServicecHost.MyService">
      <endpoint address="Mex"
                binding="mexTcpBinding"
                bindingConfiguration=""
                name="MexEndpoint"
                contract="IMetadataExchange" />
      <endpoint address=""
                binding="netTcpBinding"
                bindingConfiguration=""
                name="TcpBinding"
                contract="WCFServicecHost.IMyService" />
      <host>
         <baseAddresses>
           <add baseAddress="net.tcp://perfpres02.npscode.com/MyService" />
         </baseAddresses>
      </host>
    </service>
  </services>

</system.serviceModel>
```

Database Server

Configuration

- A SQL Server login is created for the WCF Service account.
- The WCF login name is given access to the application database.
- The role is created in the application database.
- The WCF login name is added to the role.

```
-- Create a SQL Server login  that matches the WCF machine name
EXEC SP_GRANTLOGIN 'npscode\perfpres02$'

-- Grant the login access to the application database
use testdb
go
exec sp_grantdbaccess 'npscode\perfpres02$'

-- Create the new database role
use testdb
go
exec sp_addrole 'myrole2','db_owner'
```

```
-- Add the new login to the role
use testdb
go
exec sp_addrolemember 'myrole2','npscode\perfpres02$'
```

Additional Resources

- For more information on security authentication best practices, see “Best Practices for Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms731059.aspx
- For additional information on message security, see “Message Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For more information on hosting in a Windows service, see the document “How To: Host WCF in a Windows Service.”
- For more information on WCF hosting considerations, see “Hosting Services” at http://msdn2.microsoft.com/en-us/library/ms730158.aspx
- For more information on **netTcpBinding** configuration options see “<netTcpBinding>” at http://msdn2.microsoft.com/en-us/library/ms731343.aspx

PART IV

Internet Application Scenarios

In This Part:

- **Internet - WCF and ASMX Client to Remote WCF Using Transport Security (Trusted Subsystem, HTTP)**
- **Internet - Web to Remote WCF Using Transport Security (Trusted Subsystem, TCP)**
- **Internet - Windows Forms Client to Remote WCF Using Message Security (Original Caller, HTTP)**

Chapter 13 - Internet – WCF and ASMX Client to Remote WCF Using Transport Security (Original Caller, HTTP)

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5

Scenario

In this scenario, your users do not have Microsoft Windows® accounts but use a Windows Forms client to make calls to the WCF service through either a WCF or ASMX client proxy. User accounts are stored in Microsoft SQL Server®, and users are authenticated with username authentication.

The business logic called by the WCF service is backed by a SQL Server data store. The following figure illustrates the basic model for this application scenario.

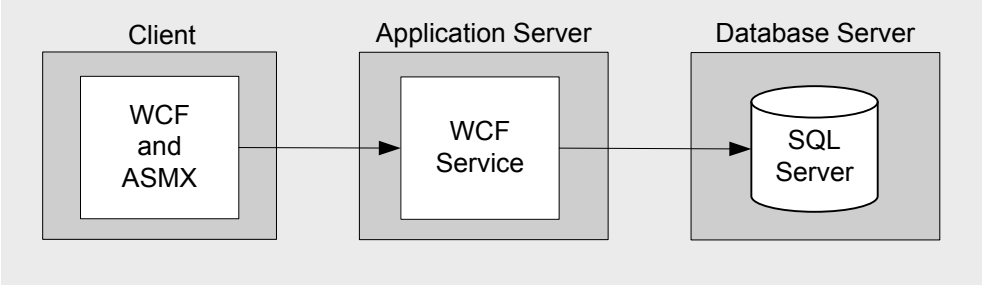


Figure 1. WCF and ASMX Client to Remote WCF Using Transport Security (Original Caller, HTTP) – Model

Key Characteristics

This scenario applies to you if:

- Your users are WCF and ASMX clients.
- Your user accounts are stored in SQL Server. Internet Information Services (IIS) authenticates users against the SQL Server membership provider, via a custom HTTP module.
- Your user roles are stored in SQL Server. WCF authorizes users with ASP.NET roles.
- Your application transmits user credentials and other sensitive data over the network and needs to be protected.
- The service is compatible with legacy ASMX clients with prior versions of the Microsoft .NET Framework

Solution

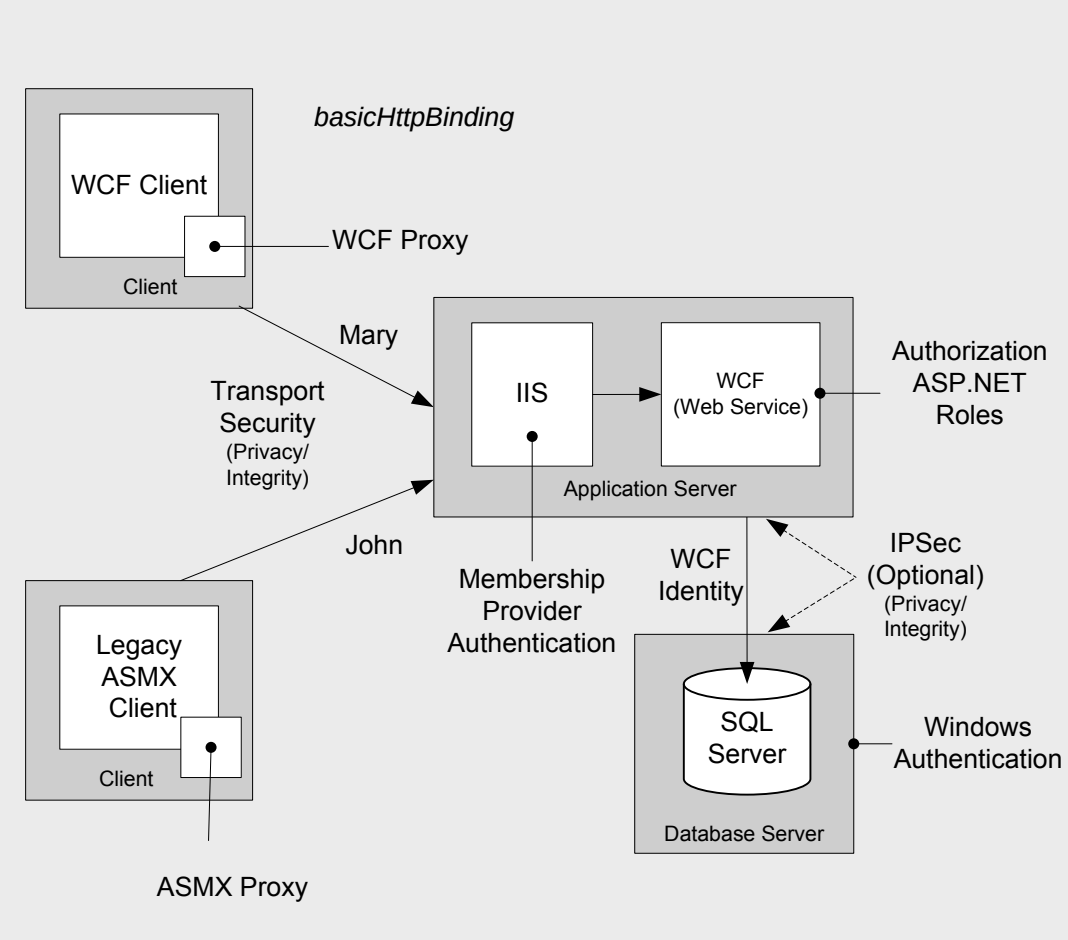


Figure 2. WCF and ASMX Client to Remote WCF Using Transport Security (Original Caller, HTTP) – Solution

Solution Summary Table

In this solution, you will:

- Authenticate clients by using the SQL Server membership provider.
- Authenticate clients by using the SQL Server membership provider with IIS via a custom HTTP module.
- Use WCF to authorize users with roles in SQL Server by using the ASP.NET role provider.
- Use a service account to call the SQL Server from WCF.
- Use transport security to protect user credentials and sensitive data passed between the clients and the WCF service.
- Use basicHttpBinding with transport security to ensure that the service is compatible with legacy ASMX clients.
- Authenticate clients using a custom HTTP module in order to transmit user credentials over the transport, to ensure that the service is compatible with legacy ASMX clients.

Clients

Checks & More Info	Example
WCF proxy	

Client needs to manually configure the authentication type as Basic. In a generated proxy, you will need to change the value from None to Basic.
 	
If the proxy is generated, this value will be generated as None because the WCF service sets the authentication as None. The Basic authentication type is needed in order for authentication negotiation to occur, so the authentication header is sent to service.
	`<security mode="Transport">`
	`  <transport clientCredentialType="Basic"/>`
	`</security>`

Client has a WCF proxy reference to the WCF service. The application has access to the WCF metadata in order to create a service reference. The client will be prompted with credentials to get the metadata.	
Root CA certificate for the service is installed in "Trusted Root Certification Authorities." This is required for Secure Sockets Layer (SSL) authentication. All certificates that are signed with this certificate will be trusted by the client machine.	

Proxy invokes the service passing user credentials to the WCF proxy.
 	
The **UserName** and **Password** properties must be set before the proxy invokes a WCF method.
	`WCFTestService.ServiceClient myService = new WCFTestService.ServiceClient();`
	`myService.ClientCredentials.UserName.UserName = "username";`
	`myService.ClientCredentials.UserName.Password = "p@ssw0rd";`
	`myService.GetData(123);`
	`myService.Close();`

Checks & More Info	Example
ASMX proxy	
Client has an ASMX Web service proxy reference to the WCF service.	

The application has access to the WCF metadata in order to create a service reference. The client will be prompted with credentials to get the metadata.	
Root CA certificate for the service is installed in "Trusted Root Certification Authorities." All certificates that are signed with this certificate will be trusted by the client machine.	
Proxy invokes the service passing user credentials to the ASMX Web service proxy. The proxy's credentials need to be set with the username and password before invoking a WCF method .	`NetworkCredential netCred = new NetworkCredential("username", " p@ssw0rd"); asmxwebservice.Service proxy = new asmxwebservice.Service(); proxy.Credentials = netCred; proxy.GetData(21, true);`

Application Server

Checks & More Info	Example
IIS - Configuration	
A dedicated application pool is created and configured to run under a custom service account. Use a domain account if possible.	
The WCF service is configured to run under the service account. Assign the WCF service to the custom application pool.	

A custom HTTP module is configured in Web configuration.
 	
The custom HTTP module will authenticate the users against the Sql Server MemberShip Provider.
	`<httpModules>`
	`…`
	`<add`
	`name="BasicAuthentication Module"`
	`type="Module.UserNameAuthenticator,Authenticator" />`
	
	`</httpModules>`

An ASP.NET database is created for use with the SQL Server membership provider and SQL Server role provider. Aspnet_regsql.exe creates the SQL database to store the user and role information.	`aspnet_regsql -S .\SQLExpress -E -A r m`

The connection string is configured to point to the user and role stored in SQL Server.	`<add`
	`name="MyLocalSQLServer"`
	`connectionString="Initial Catalog=aspnetdb;data source=localhost;Integrated`

Checks & More Info	Example
The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes for Windows Authentication.	`Security=SSPI;" />`
Sql Server Membership Provider is configured as a membership provider. The membership feature helps protect credentials, can enforce strong passwords, and provides consistent APIs for user validation and secure user management.	`<membership defaultProvider="MySqlMembershipProvider"> <providers> <clear/> <add name= "MySqlMembershipProvider" connectionStringName="MyLocalSQLServer" applicationName="MyAppName" type="System.Web.Security.SqlMembershipProvider"/> </providers> </membership>`
The Role Manager feature is enabled and Sql Server Role Provider is configured for roles authorization. The Role Manager allows you to look up users' roles without writing and maintaining custom code.	`<roleManager enabled="true" defaultProvider="MySqlRoleProvider" > <providers> <clear/> <add name="MySqlRoleProvider" connectionStringName="MyLocalSQLServer" applicationName="MyAppName" type="System.Web.Security.SqlRoleProvider" /> </providers> </roleManager>`

The WCF service process identity is given access permissions on the ASP.NET database.
 	
Your WCF service process identity requires access to the Aspnetdb database.
	`-- Create a SQL Server login for the Network Service account sp_grantlogin '<<Custom Service Account>>'`
	
	`-- Grant the login access to the membership database USE aspnetdb GO sp_grantdbaccess '<<Custom Service Account>>', '<<Custom Service Account>>'`
	
	`-- Add user to database role USE aspnetdb GO sp_addrolemember 'aspnet_Membership_FullAccess', '<<Custom Service Account>>'`
	
	`sp_addrolemember 'aspnet_Roles_FullAccess', '<<Custom Service Account >>'`

WCF service - Configuration	
The WCF Service is configured to use **basicHttpBinding** binding. The **basicHttpBinding** binding uses the HTTP protocol and provides compatibility with ASMX clients.	`<services> <service behaviorConfiguration="ServiceBehavior" name="Service"> <endpoint address="" binding="basicHttpBinding" bindingConfiguration="BindingConfiguration" name="basicEndpoint" contract="IService" /> </service> </services>`
Service Metadata is configured in service behavior to enable **httpsGetEnabled** and disable	`<serviceBehaviors> <behavior name="ServiceBehavior"> <serviceMetadata httpGetEnabled="false"`

Checks & More Info	Example

httpGetEnabled.
 	
The service metadata entry is required in order to publish metadata to the clients.
	`httpsGetEnabled="true" />`
	`  </behavior>`
	`</serviceBehaviors>`

The service is configured for ASP.NET compatibility mode, both in configuration and in service implementation.
 	
The ASP.NET compatibility mode is necessary because IIS is performing authentication.
	`Configuration`
	`<system.serviceModel>`
	` <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />`
	`…`
	`</system.serviceModel>`
	
	`Service Implementation`
	`[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]`
	`public class Service : IService`

WCF service - Authentication	

basicHttpBinding is configured to use transport security and no authentication.
 	
The authentication will be performed by the ASP.NET HTTP module against the Sql Server MemberShip Provider.
	`<basicHttpBinding>`
	`  <binding name= "BindingConfiguration">`
	`   <security mode="Transport">`
	`    <transport clientCredentialType="None" />`
	`   </security>`
	`  </binding>`
	`</basicHttpBinding>`

Sql Server Membership Provider is configured to provide user authentication.
 	
The membership feature automatically authenticates and creates the authentication ticket for you.
	`<membership defaultProvider="MySqlMembershipProvider">`
	`  <providers>`
	`    <clear/>`
	`    <add name="MySqlMembershipProvider" connectionStringName="MyLocalSQLServer" applicationName="MyAppName" type="System.Web.Security.SqlMembershipProvider"/>`
	`  </providers>`
	`</membership>`

WCF service - Authorization	

The Role Manager feature is enabled with aspnetroles, and the provider is configured for roles authorization.
 	
Roles authorization can be performed declaratively or imperatively in the operation contract.
	`<serviceBehaviors>`
	`  <behavior name="ServiceBehavior">`
	`    <serviceAuthorization principalPermissionMode="UseAspNetRoles" roleProviderName="MySqlRoleProvider" />`
	`  </behavior>`
	`</serviceBehaviors>`

A class that derives from IAuthorizationPolicy is implemented to set the principal of the current thread to do declarative authorization, and to set the identity so that it is available in a WCF security context.	

Checks & More Info	Example
The authorization policy class will allow to assign the principal to the context of WCF, so users can be authorized, and It will also allow to assign the identity to the context of WCF, so the it can be retrieved from WCF security context.	

The authorization policy is included in the configuration file.	`<authorizationPolicies>`
	`<add`
	`policyType="AuthorizationPolicy.HttpContextPrincipal Policy, AuthorizationPolicy" />`
	`</authorizationPolicies>`

Perform role checks declaratively by using a Windows Identity Token, for checking Microsoft Active Directory® group membership.
 	
A declarative role check is preferred over an imperative role check for a service operation.
	`[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]`
	`public string GetData(string message)`
	`{`
	`return "hello";`
	`}`

Perform role checks imperatively using a Windows Identity Token, for checking Active Directory group membership.
 	
If you need more fine-grained authorization control, you can use imperative role checks in the code itself. Use a call to **Roles.IsUserInRole** to perform the check.
	`public string GetData(string myValue)`
	`{ if(Roles.IsUserInRole(@"Accounting"))`
	`{`
	`//Do something for Accounting role`
	`}`
	`else`
	`{`
	`//Do something for non-accounting role or throw an error`
	`}`
	`}`

WCF service - SQL	
The connection string for the database is configured to use Windows authentication. The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes.	`SqlConnection sqlcon = new SqlConnection("Server=SqlServer;Database=Northwind;IntegratedSecurity=SSPI");`
A database connection is opened by using the WCF process identity's security context. This happens by default.	

Database Server

Check	Example
Configuration	
A SQL Server login is created for the WCF service account (process identity). This grants access to the SQL Server.	`exec sp_grantlogin 'Custom Service Account'`

The login is granted access to the target database.
 	
This grants access to the specified database.
	`use targetDatabase`
	`go`
	`exec sp_grantdbaccess 'Custom Service Account'`
	`go`

A database role is created in the target database.
 	
This allows access control and authorization to the database.
	`use targetDatabase`
	`go`
	`exec sp_addrole 'DB Role Name'`
	`go`

The login is added to the database role.
 	
Grant minimum permissions. For example, grant execute permissions to selected stored procedures, and provide no direct table access.
	`use targetDatabase`
	`go`
	`exec sp_addrolemember 'DB Role Name', 'Custom Service Account'`
	`go`

Authentication	
SQL Server is configured to use Windows authentication.	

Communication Security

What	Check	Example	More info
App server to Database	You can use Internet Protocol security (IPSec) or SSL between the application server and database server to protect sensitive data in transit.		

Analysis

Clients

WCF Proxy

- The client configuration file is configured to use Basic authentication in order to allow the authentication negotiation to occur.
- The user's credentials are required in the ASP.NET HTTP module for authentication. Username credentials are set on the WCF proxy, and all calls to the WCF service are made through that proxy instance.
- The user's credentials are required in WCF for authorization. Username credentials are set on the WCF proxy, and all calls to the WCF service are made through that proxy instance.
- For validating the service certificate, the Root CA certificate is installed on the client machine in the "Trusted Root Certification Authorities" location.

ASMX Web Service Proxy

- The user's credentials are required in the ASP.NET HTTP module for authentication. \Network credentials are set on the ASMX Web service proxy, and all calls to the WCF service are made through that proxy instance.
- The user's credentials are required in WCF for authorization. Username credentials are set on the ASMX Web service, and all calls to the WCF service are made through that proxy instance.
- For validating the service certificate, the Root CA certificate is installed on the client machine in the "Trusted Root Certification Authorities" location.

Application Server

Authentication

- Because the users communicate with the WCF service over the Internet and you cannot assume that they have a Windows account, the user information is stored in SQL Server. Since WCF does not support transport security with username authentication, a custom HTTP module is created that will authenticate the user against the SQL Server Membership Provider. This will support both WCF and ASMX Web services clients.
- WCF is configured to use no authentication because the ASP.NET HTTP module will handle authentication.
- To protect the user credentials in transit, a Service Certificate is installed and is configured to be used as Service Credentials in WCF.

Authorization

- For coarse-grained access control, authorization checks are performed in the WCF service at the operation level, declaratively. Unless fine-grained access control is needed, declarative authorization should be preferred over imperative authorization.

- For fine-grained access control or for implementing business logic, authorization checks are made within the operations programmatically.
- The Roles Manager is a good choice for this scenario because it allows you to look up users' roles without writing and maintaining custom code.

Data Access

- To reduce the risk of stolen database credentials, the database connection string is configured to use Windows authentication. This avoids storing credentials in files and passing credentials over the network to the database server.
- The WCF service accesses the database by using the WCF process identity. As a result, all calls use the single process account and the designated database connection pooling.

Configuration

- Since all of the clients communicate over the Internet, the best transport protocol for this scenario is the HyperText Transfer Protocol (HTTP). Additionally, since compatibility with ASMX Web services clients is required, **basicHttpBinding** is an ideal choice.
- Because **basicHttpBinding** is supported by IIS 6.0, the WCF service is hosted in IIS.
- In order to reduce attack surface and minimize the impact of a compromise, the WCF service runs under the security context of the Service account, using a least-privileged account.
- In order to reduce attack surface and minimize the impact of a compromise, the Windows service runs under the security context of the Service account, using a least-privileged account.

Database Server

- SQL Server database user roles are preferred to SQL Server application roles in order to avoid the associated password management and connection pooling issues associated with the use of SQL Server application roles. Applications activate SQL Server application roles by calling a built-in stored procedure with a role name and a password. Therefore, the password must be stored securely. Database connection pooling must also be disabled when you use SQL Server application roles, which severely impacts application scalability.
- Creating a new user-defined database role, and adding the database user to the role, lets you give specific minimum permissions to the role. Therefore, if the database account changes, you do not have to change the permissions on all database objects.

Communication Security

- Transport security protects sensitive data between the Thick Client and WCF service.
- You can use IPSec or SSL between the WCF service and the database server in order to protect sensitive data in transit.

Example

Clients

WCF Client

Code

- The client passes user credentials explicitly when making calls to the service.
- The client needs to provide credentials when creating a service reference.

```
WCFTestService.ServiceClient myService = new
WCFTestService.ServiceClient();
myService.ClientCredentials.UserName.UserName = "username";
myService.ClientCredentials.UserName.Password = "p@ssw0rd";
myService.GetData(123);
myService.Close();
```

Configuration

- The client is configured to use Basic authentication.

```
<security mode="Transport">
  <transport clientCredentialType="Basic"
             proxyCredentialType="None"
             realm="" />
</security>
```

ASMX Web Service Client

Code

- The client passes user credentials explicitly when making calls to the service.
- The client needs to provide credentials when creating a service reference.

```
NetworkCredential netCred = new NetworkCredential("username", "
p@ssw0rd");
asmxwebservice.Service proxy = new asmxwebservice.Service();
proxy.Credentials = netCred;
proxy.GetData(21, true);
```

Application Server

IIS

Code

- A class that derives from **IHttpModule** is implemented. This class authenticates the users against SQL Membership Provider.

- Initially, the class checks to determine if there is an authorization header in the request from the client. If the header is not present, the status of the context is assigned as 401(not authorized) and a WWW-Authenticate header is created and sent in the response to the client. This is the “handshake” for the authentication process. The client will know that it needs to send credentials for authentication.
- Once the credentials have been sent by the client, they are extracted from the authorization header, so they can be used to call the SQL Membership Provider.
- The class authenticates the user, calling Membership.ValidateUser(username, password) to validate the user against the SQL Membership Provider.
- If the user is authenticated, an identity is created and assigned to the **HttpApplication.Context.User** property.
- If the user is not authenticated, a 401 status is returned to the client and the user is denied access.

HTTP Module Code

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;
using System.Web.Security;
using System.Security.Principal;

namespace Module
{
  public class UserNameAuthenticator : IHttpModule
  {
     public void Dispose()
     {
     }

     public void Init(HttpApplication application)
     {
      application.AuthenticateRequest += new
        EventHandler(this.OnAuthenticateRequest);
      application.EndRequest += new
        EventHandler(this.OnEndRequest);
     }

     public void OnAuthenticateRequest(object source, EventArgs eventArgs)
     {
       HttpApplication app = (HttpApplication)source;
       //the Authorization header is checked if present
       string authHeader = app.Request.Headers["Authorization"];

       if (!string.IsNullOrEmpty(authHeader))
       {
         string authStr = app.Request.Headers["Authorization"];

         if (authStr == null || authStr.Length == 0)
```

```
        {
         // No credentials; anonymous request
         return;
        }



        authStr = authStr.Trim();
        if (authStr.IndexOf("Basic", 0) != 0)
         {
          //header not correct we do not authenticate
          return;
         }

         authStr = authStr.Trim();
         string encodedCredentials = authStr.Substring(6);
         byte[] decodedBytes = Convert.FromBase64String(encodedCredentials);
         string s = new ASCIIEncoding().GetString(decodedBytes);

         string[] userPass = s.Split(new char[] { ':' });
         string username = userPass[0];
         string password = userPass[1];

         //the user is validated against the SqlMemberShipProvider
         //If it is validated then the roles are retrieved from the
         //role provider and a generic principal is created
         //the generic principal is assigned to the user context
         // of the application


         if (Membership.ValidateUser(username, password))
         {
          string[] roles = Roles.GetRolesForUser(username);
          app.Context.User = new GenericPrincipal(new
          GenericIdentity(username, "Membership Provider"), roles);
         }
         else
         {
           DenyAccess(app);
           return;
         }

       }  //end of-  if (!string.IsNullOrEmpty(authHeader))
       else
       {
         //the authorization header is not present
         //the status of response is set to 401 and it ended
         //the end request will check if it is 401 and add
         //the authentication header so the client knows
         //it needs to send credentials to authenticate

         app.Response.StatusCode = 401;
         app.Response.End();

         //context.Response.StatusCode = 401;
         //context.Response.End();
        }
```

```
        }  //End class function

        public void OnEndRequest(object source, EventArgs eventArgs)
        {
            if (HttpContext.Current.Response.StatusCode == 401)
            {

            //if the status is 401 the WWW-Authenticated is added to
            //the response so client knows it needs to send credentials

            HttpContext context = HttpContext.Current;
            context.Response.StatusCode = 401;
            context.Response.AddHeader("WWW-Authenticate", "Basic Realm");
            }
        }

        private void DenyAccess(HttpApplication app)
        {
          app.Response.StatusCode = 401;
          app.Response.StatusDescription = "Access Denied";

          // error not authenticated
          app.Response.Write("401 Access Denied");

          app.CompleteRequest();
        }
    } // End Class
}  //End Namespace
```

Configuration

- The custom module is configured in the web.config file, in the HTTP modules section.
- The service configuration file has an entry with a connection string pointing to the SQL Server store for authentication and authorization.
- The service configuration file has an entry for the **SqlRoleProvider** under system.web to define which role provider is being used.
- The service configuration file has an entry for the **SqlMemberShipProvider** under system.web to define the SQL Server membership provider for authentication.

```
<configuration>
…
  <connectionStrings>
    <add name="MyLocalSQLServer"
         connectionString="Initial Catalog=aspnetdb;data
source=10.3.19.60;Integrated Security=SSPI;"/>
  </connectionStrings>

   <system.web>
      <membership defaultProvider="MySqlMembershipProvider" >
        <providers>
          <clear/>
          <add name="MySqlMembershipProvider"
               connectionStringName="MyLocalSQLServer"
```

```
                applicationName="MyAppName"
                type="System.Web.Security.SqlMembershipProvider" />
          </providers>
        </membership>

        <roleManager enabled="true" defaultProvider="MySqlRoleProvider" >
          <providers>
            <clear/>
            <add name="MySqlRoleProvider"
                connectionStringName="MyLocalSQLServer"
                applicationName="MyAppName"
                type="System.Web.Security.SqlRoleProvider" />
          </providers>
        </roleManager>

        <httpModules>
        …
        <add name="BasicAuthenticationModule"
        type="Module.UserNameAuthenticator,Authenticator" />
        </httpModules>
      </system.web>

  </configuration>
```

WCF

Code

- The service performs imperative authorization checks, calling **Roles.IsUserInRole**.
- If auditing is required, the service retrieves the identity of the caller.
- The Authorization policy class is developed to set the security principal to WCF context. This way when the business logic runs in the operation contract, it's possible to do authorization checks and auditing with the identity.
- do declarative authorization, and to have the identity in a WCF security context.
- The service calls SQL Server by using Windows authentication.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IdentityModel.Claims;
using System.IdentityModel.Policy;
using System.Web;
using System.Security.Principal;

namespace AuthorizationPolicy
{
    // syncs Thread.CurrentPrincipal and identity in WCF with whatever is set
    // by the HTTP pipeline on Context.User (optional)

    public class HttpContextPrincipalPolicy : IAuthorizationPolicy
    {
        public bool Evaluate(EvaluationContext evaluationContext,
                             ref object state)
```

```
        {
            HttpContext context = HttpContext.Current;

            if (context != null)
            {
                evaluationContext.Properties["Principal"] = context.User;
                evaluationContext.Properties["Identities"] =
                   new List<IIdentity>() { context.User.Identity };
            }
            return true;
        }

        public System.IdentityModel.Claims.ClaimSet Issuer
        {
            get { return ClaimSet.System; }
        }

        public string Id
        {
            get { return "HttpContextPrincipalPolicy"; }
        }
    }
}
```

The service does imperative or declarative authorization, as shown in the following sections.

Imperative

```
using System.Data.SqlClient;
using System.Web.Security;

public string GetData(int value)
{
     if (Roles.IsUserInRole(@"accounting"))
     {
      SqlConnection sqlcon = new SqlConnection("Server=sqlServer;
            Database=testdb;Integrated Security=SSPI");
      sqlcon.Open();

      string identity =
      HttpContext.Current.User.Identity.Name;
      return “data”
     }
     else return "not authorized";
}
```

Declarative

```
using System.Data.SqlClient;
using System.Web.Security;

PrincipalPermission(SecurityAction.Demand, Role = "accounting")]

public string GetData(int value)
{
```

```
 SqlConnection sqlcon = new SqlConnection("Server=sqlServer;
Database=testdb;Integrated Security=SSPI");
 sqlcon.Open();

 string identity = HttpContext.Current.User.Identity.Name;
 return “data”
}
```

Configuration

- The service has a binding endpoint that uses **basicHttpbinding** with binding configuration that enables transport security and no authentication.
- The service behavior is configured with the element **serviceMetadata** to allow publishing metadata.
- The service behavior is configured with the element **ServiceAuthorization** to use ASP.NET roles for authorization.

```
<system.serviceModel>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />

    <bindings>
      <basicHttpBinding>
        <binding name="BindingConfiguration">
          <security mode="Transport">
            <transport clientCredentialType="None" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>

    <services>
      <service behaviorConfiguration="ServiceBehavior" name="Service">
        <endpoint address="" binding="basicHttpBinding"
                  bindingConfiguration="BindingConfiguration"
                  name="basicEndpoint" contract="IService" />
      </service>
    </services>

    <behaviors>
      <serviceBehaviors>
        <behavior name="ServiceBehavior">
          <serviceMetadata  httpGetEnabled="false" httpsGetEnabled="true" />
          <serviceAuthorization principalPermissionMode="UseAspNetRoles"
                                roleProviderName="MySqlRoleProvider">
            <authorizationPolicies>
             <add policyType="AuthorizationPolicy.HttpContextPrincipalPolicy,
AuthorizationPolicy" />
            </authorizationPolicies>
          </serviceAuthorization>
        </behavior>
      </serviceBehaviors>
    </behaviors>
```

```
</system.serviceModel>
```

Database Server

Configuration

- A SQL Server login is created for the WCF service account.
- The WCF login name is given access to the application database.
- The role is created in the application database.
- The WCF login name is added to the role.

```
-- Create a SQL Server login  that matches the WCF machine name
EXEC SP_GRANTLOGIN 'npscode\perfpres02$'

-- Grant the login access to the application database
use testdb
go
exec sp_grantdbaccess 'npscode\perfpres02$'

-- Create the new database role
use testdb
go
exec sp_addrole 'myrole2','db_owner'

-- Add the new login to the role
use testdb
go
exec sp_addrolemember 'myrole2','npscode\perfpres02$'
```

Additional Resources

- For more information on how to work with the ASP.NET Role Provider, see “How to: Use the ASP.NET Role Provider with a Service” at http://msdn2.microsoft.com/en-us/library/aa702542.aspx
- For more information on how to work with the ASP.NET Role Manager, see “How To: Use Role Manager in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998314.aspx
- For more information on how to work with the ASP.NET Membership Provider, see “How to: Use the ASP.NET Membership Provider” at http://msdn2.microsoft.com/en-us/library/ms731049.aspx
- For more Information on IHTTP Module interface, see http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.aspx
- For more information on how to use transport security with username authentication, see How To – Use Username Authentication with Transport Security in WCF from Windows Forms

Chapter 14 - Internet – Web to Remote WCF Using Transport Security (Trusted Subsystem, TCP)

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5

Scenario

In this scenario, your users do not have Windows accounts and use a Web client to connect over the Internet to an ASP.NET application on an IIS server. The business logic called by the WCF service is backed by a SQL Server data store. The basic model for this application scenario is shown in the following figure.

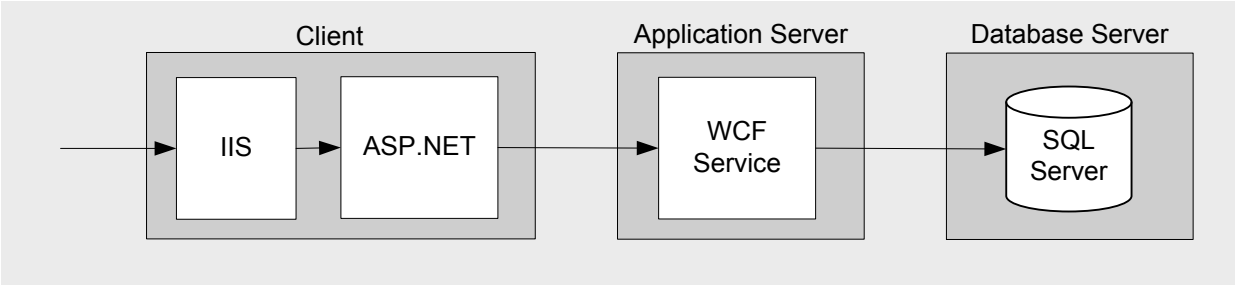


Figure 1. Web to Remote WCF Using Transport Security (Trusted Subsystem, TCP) – Model

Key Characteristics

This scenario applies to you if:

- Your users have Web clients.
- Your user accounts are stored in SQL.
- Your user roles are stored in SQL.
- The business logic behind your WCF service does not require fine-grained authorization.
- Your application transmits sensitive data over the network that needs to be protected.
- A high performance connection between the ASP.NET application and the WCF service is more important than the ability to host the WCF service in IIS.

Solution

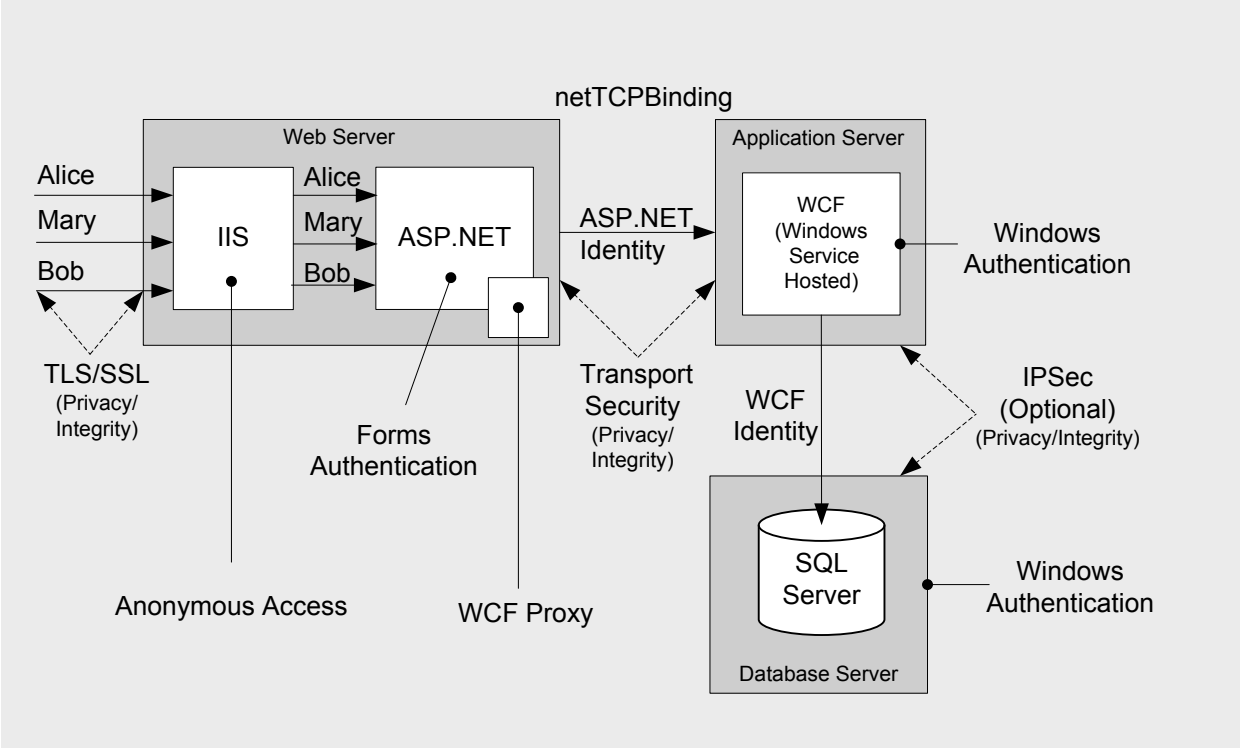


Figure 2. Web to Remote WCF Using Transport Security (Trusted Subsystem, TCP) – Solution

Solution Summary Table

In this solution you will:

- Use username and password to authenticate users against the SQL Server Membership Provider.
- Use a service account to call WCF from the ASP.NET application.
- Use a service account to call the SQL Server from WCF.
- Use SSL to protect sensitive data between the Web client and IIS.
- Use Transport Security to protect sensitive data between the ASP.NET application and the WCF service.
- Use **netTcpBinding** to support the TCP transport for improved performance.
- Host WCF in a Windows Service since IIS does not support the TCP transport.

Web Server

Checks	Example
IIS - Configuration	
A dedicated application pool is created and configured to run under a custom service account. Use a domain account if possible.	
The Web application is configured to run under the service account. Assign the Web application to the custom application pool.	
IIS - Authentication	
The IIS virtual directory is configured to use Anonymous access. Users will be allowed to access pages and if required will be redirected to forms authentication page.	

Checks	Example
ASP.NET - Configuration	
ASP.NET database is created for the SQL Membership Provider and SQL Role Provider. Aspnet_regsql.exe creates the SQL database to store the user and role information.	`aspnet_regsql -S .\SQLExpress -E -A r m`
Connection string is configured to point to the user and role store in SQL Server. The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes for Windows Authentication.	`<add name="MyLocalSQLServer" connectionString="Initial Catalog=aspnetdb;data source=localhost;Integrated Security=SSPI;" />`

Checks	Example

Web application process identity is given access permissions on the ASPNET database.
 	
Your Web application process identity requires access to the Aspnetdb database. If you host the Web application in Internet Information Services (IIS) 6.0 on Microsoft Windows Server® 2003, the NT AUTHORITY\Network Service account is used by default to run the Web application.
	`-- Create a SQL Server login for the Web application process identity`
	`sp_grantlogin 'Customdomainserviceaccount`
	
	`-- Grant the login access to the membership database`
	`USE aspnetdb`
	`GO`
	`sp_grantdbaccess 'Customdomainserviceaccount', 'Custom Service'`
	
	`-- Add user to database role`
	`USE aspnetdb`
	`GO`
	`sp_addrolemember 'aspnet_Membership_FullAccess', 'Custom Service'`
	
	`sp_addrolemember 'aspnet_Roles_FullAccess', 'Custom Service'`

ASP.NET - Authentication	
ASP.NET is configured for Forms authentication. The Web application will authenticate the users.	`<authentication mode = "Forms" >`

ASP.NET application is configured to deny access to all unauthenticated users.
 	
Only authenticated users will be able to access the application.
	`<authorization>`
	`  <deny users="?"/>`
	`  <allow users="*"/>`
	`</authorization>`

SqlMembershipProvider is configured to use with Membership feature for forms authentication.
 	
The membership feature helps protect credentials, can enforce strong passwords, and provides consistent APIs for user validation and secure user management. The membership feature also automatically creates the authentication ticket for you.
	`<membership defaultProvider="MySqlMembershipProvider">`
	`  <providers>`
	`  <clear/>`
	`  <add name="MySqlMembershipProvider"`
	
	`connectionStringName="MyLocalSQLServer"`
	`  applicationName="MyAppName"`
	
	`type="System.Web.Security.SqlMembershipProvider"/>`
	`  </providers>`
	`</membership>`

ASP.NET - Authorization	
Role Manager feature is enabled and	`<roleManager enabled="true"`

Checks	Example
SqlRoleProvider is configured for roles authorization. Role Manager feature allows you to look up users' roles without writing and maintaining code. Additionally, the role providers offer a consistent way for you to check the role membership of your users, regardless of the underlying data store.	`defaultProvider="MySqlRoleProvider" >   <providers>   <clear/>   <add name="MySqlRoleProvider"  connectionStringName="MyLocalSQLServer"   applicationName="MyAppName"  type="System.Web.Security.SqlRoleProvider" />   </providers> </roleManager>`
Role-checks are performed using role manager APIs.	`if (User.IsInRole("Role")) {   //business operation  }`

Checks	Example
WCF Proxy	
ASP.NET has a proxy reference to the WCF service. The application has access to the WCF metadata to create a service reference.	`WCFTestService.ServiceClient myService = new WCFTestService.ServiceClient();`
Proxy invokes services with the security context of service account . The proxy will automatically invoke WCF operations using the security context of the service account.	`myService.GetData(123);`
WCF Proxy - Caller Identity	

For auditing purposes, the identity of the caller can be passed in custom message headers during
Proxy call. Additionally custom headers can be defined in message contracts or service contracts.
 	
Use transport security to protect against spoofing attacks.
	`if (User.IsInRole("accounting"))
	 {
	  WCFTestService.MyServiceClient proxy = new
	  WCFTestService.MyServiceClient();
	  using (OperationContextScope scope = new
	
	OperationContextScope(proxy.InnerChannel))
	  {
	  string identity = User.Identity.Name;
	  MessageHeader<string> headerIdentity =
	  new
	MessageHeader<string>(identity);
	  MessageHeader untypedMessageHeader =`

Checks	Example

	`headerIdentity.GetUntypedHeader("identity" , "ns");`
	`}`
	
	`OperationContext.Current.OutgoingMessageHeaders.Add(untypedMessageHeader);`
	`proxy.GetData("data");`
	`}`
	`proxy.Close();`

Application Server

Checks	Example
Windows Service -Configuration	
Windows Service is configured to run under a custom domain service account. Use a domain account if possible.	
WCF service is hosted in a Windows Service. Since IIS does not support **netTcpBinding**, host it in Windows Service.	

Checks	Example
WCF Service - Configuration	

Configure the WCF service to use **netTcpBinding**.
 	
NetTcpBinding uses the TCP protocol and provides full support for SOAP security, transactions, and reliability. As client and WCF service both are in the Intranet, this is a good choice from a performance perspective.
	`<endpoint address=""`
	`binding="netTcpBinding"`
	`bindingConfiguration="" name="TcpBinding"`
	`contract="WCFServicecHost.IMyService" />`

A mex endpoint is created for publishing the metadata.
 	
This is required so that client can add reference to the WCF Service using
	`<endpoint address="Mex"`
	`binding="mexTcpBinding"`
	`bindingConfiguration=""`
	`name="MexEndpoint"`
	`contract="IMetadataExchange" />`

Checks	Example
SvcUtil utility.	
Service Metadata is configured in service behavior. The service metadata entry is required for the Windows Service host to start. Both HTTP and HTTPS get are disabled.	`<serviceMetadata />`
WCF Service - Authentication	

netTcpBinding is configured to use Windows Authentication and Transport Security.
 	
By default, **netTcpBinding** is configured to use Windows Authentication and Transport Security.
	`<endpoint address=""`
	`binding="netTcpBinding"`
	`bindingConfiguration="" />`

WCF Service - Caller Identity	

Service retrieves the identity of the caller from the operationcontext
For auditing purposes.
 	
Use the identity to improve logging and auditing.
	`string identity =`
	`OperationContext.Current.IncomingMessageHea`
	`ders.GetHeader<string>("identity", "ns");`

WCF Service - SQL	
The connection string for database is configured to use Windows Authentication. The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes.	The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes
Database connection is opened using the WCF process identity's security context. Service does not impersonate the original caller to benefit for connection pooling.	

Database Server

Check	Example
Configuration	
A SQL Server login is created for the WCF's service account (process identity). This grants access to the SQL Server.	`exec sp_grantlogin 'Custom Service Account'`

The login is mapped to a database user for the Web application.
 	
This grants access to the specified database.
	`use targetDatabase`
	`go`
	`exec sp_grantdbaccess ' Custom Service Account'`
	`go`

A database role is created in the target database.
 	
This allows access control and authorization to the DB.
	`use targetDatabase`
	`go`
	`exec sp_addrole 'DB Role Name'`
	`go`

The login is added to the database role.
 	
Grant minimum permissions. For example, grant execute permissions to selected stored procedures and provide no direct table access.
	`use targetDatabase`
	`go`
	`exec sp_addrolemember 'DB Role Name', 'Custom Service Account'`
	`go`

Authentication	
SQL Server is configured to use Windows Authentication.	

Communication Security

What	Check	More Info
Browser to Web Server	SSL is used between browser and Web Server to protect sensitive data on the wire.	Install certificate in the Website. Configure the virtual directory of the Web application to use SSL.
App Server to Database Server	You can use IPSec or SSL between the App Server and Database Server to protect sensitive data on the wire.	

Analysis

Web Server

Authentication

- To allow unauthenticated and unauthorized users to access pages and redirect to the login page, anonymous access in IIS is enabled.
- Forms authentication is a good choice for this scenario because users come from the Internet and have accounts in SQL.
- The membership feature is a good choice to use with forms authentication, as it allows user authentication without writing and maintaining custom code.

Authorization

- URL authorization performs role checks against the original caller and restricts access to pages based on role permissions.
- All authorization checks occur in the Web application before it makes calls to the WCF service. The WCF service trusts the Web application to perform this authorization and does not need to make fine-grained authorization decisions of its own.
- The Roles Manager is a good choice for this scenario because it allows the application to look up users' roles without writing and maintaining custom code.

WCF Proxy

- Because you are taking care of all authentication and authorization in the ASP.NET application, all calls through the WCF proxy and into the WCF service use the ASP.NET process identity’s security context.
- If you need to produce audit logs showing what service operations each user called, you can pass the identity of the original caller in a custom header.

Configuration

- In order to reduce attack surface and minimize the impact of a compromise, the ASP.NET application on the Web Server runs under the security context of the Service account using a least privileged account.

Application Server

Authentication

- WCF is configured to use Windows Authentication in order to authenticate the ASP.NET service when it makes calls on the WCF Service.

Authorization

- Since the WCF Service trusts the ASP.NET application to authorize the user, the WCF service performs no authorization. .

Data Access

- To reduce the risk of database credentials theft, the database connection string is configured to use Windows Authentication. This choice avoids storing credentials in files and passing credentials over the network to the Database Server.
- The WCF service accesses the database using the WCF process identity. As a result, all calls use the single process account and designated database connection pooling.

Configuration

- This scenario is optimized around transmission performance at the expense of interoperability with clients that expect a legacy Web service and the ability to host the service in IIS. For this reason, the best binding choice is **netTcpBinding**. By default, **netTcpBinding** supports Windows Authentication with Transport Security.
- Because IIS 6.0 does not support **netTcpBinding**, the WCF service is hosted in a Windows service.
- In order to reduce attack surface and minimize the impact of a compromise, the Windows Service is running under the security context of the Service account using a least privileged account.
- A metadata exchange (mex) endpoint is exposed to make it possible for the client to generate a proxy based on the service definition.

Database Server

- SQL Server database user roles are preferred to SQL Server application roles to avoid the associated password management and connection pooling issues associated with the use of SQL application roles. Applications activate SQL application roles by calling a built-in stored procedure with a role name and a password. Therefore, you must store the password securely. You must also disable database connection pooling when you use SQL application roles, which severely impacts application scalability.
- Creating a new user-defined database role and adding the database user to the role lets you give specific minimum permissions to the role. In this way, if the database account changes you don't have to change the permissions on all database objects.

Communication Security

- SSL protects sensitive data on the wire between the browser and Web server.
- Transport Security protects sensitive data between the Web Server and App Server.
- You can use IPSec or SSL between the App Server and Database Server to protect sensitive data on the wire.

Example

Web Server

Code

- Form is created to perform Forms authentication.
- Role-authorization occurs before WCF service invocation.
- ASP.NET calls WCF service if it is authorized.
- Identity of the original caller is retrieved from the User ticket context.
- Message Header containing the caller identity is created and passed to the operation context for auditing purposes.

Form to do forms authentication

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Login.aspx.cs"
Inherits="Login" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    <asp:Login ID="Login1"  runat="server">
    </asp:Login>
    <asp:CreateUserWizard ID="CreateUserWizard1" runat="server">
        <WizardSteps>
            <asp:CreateUserWizardStep runat="server" />
            <asp:CompleteWizardStep runat="server" />
        </WizardSteps>
    </asp:CreateUserWizard>
    </form>
</body>
</html>
```

```
//Proxy call invocation
using System.ServiceModel;
using System.ServiceModel.Channels;
…
protected void Button1_Click(object sender, EventArgs e)
 {
  if (User.IsInRole("accounting"))
  {
    WCFTestService.MyServiceClient proxy
         = new WCFTestService.MyServiceClient();
```

```
    using (OperationContextScope scope
         = new OperationContextScope(proxy.InnerChannel))
    {
     string identity = User.Identity.Name;
     MessageHeader<string> headerIdentity
          = new MessageHeader<string>(identity);
     MessageHeader untypedMessageHeader
          = headerIdentity.GetUntypedHeader("identity", "ns");

   OperationContext.Current.OutgoingMessageHeaders.Add(untypedMessageHeader);
     proxy.GetData("data");
    }
    proxy.Close();

   } //endif

 } //end function
```

Configuration

- Windows and anonymous authentication are enabled.
- Connection string to the **SqlMembershipProvider** and to the **SqlRoleProvider** are configured.
- **SQLmembershipProvider** is enabled.
- Only authenticated users are allowed to browse the site.
- Role Manager is enabled.

```
<configuration>
   …
<connectionStrings>
   <add name="MyLocalSQLServer" connectionString="Initial
        Catalog=aspnetdb;data source=10.3.19.60;Integrated Security=SSPI;"/>
</connectionStrings>

<system.web>
  <membership defaultProvider="MySqlMembershipProvider">
     <providers>
        <clear/>
        <add name="MySqlMembershipProvider"
             connectionStringName="MyLocalSQLServer"
             applicationName="MyAppName"
             type="System.Web.Security.SqlMembershipProvider"/>
     </providers>
  </membership>

  <roleManager enabled="true" defaultProvider="MySqlRoleProvider">
      <providers>
        <clear/>
        <add name="MySqlRoleProvider"
             connectionStringName="MyLocalSQLServer"
             applicationName="MyAppName"
             type="System.Web.Security.SqlRoleProvider"/>
      </providers>
```

```
  </roleManager>

  <authentication mode="Forms"/>
  <authorization>
    <deny users="?"/>
    <allow users="*"/>
  </authorization>

  <pages>
    <controls>
      <add tagPrefix="asp"
           namespace="System.Web.UI"
           assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>

      <add tagPrefix="asp"
           namespace="System.Web.UI.WebControls"
assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>

    </controls>
  </pages>

  <httpHandlers>
      <remove verb="*" path="*.asmx"/>

      <add verb="*" path="*.asmx" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>

      <add verb="*" path="*_AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>

      <add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
validate="false"/>

  </httpHandlers>

  <httpModules>
     <add name="ScriptModule" type="System.Web.Handlers.ScriptModule,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>

  </httpModules>

</system.web>
```

Application Server

Code

- The service retrieves the identity of the caller from the operation context if it is required for auditing purposes.
- The service calls SQL using the security context of the WCF service.

```
using System.Data.SqlClient;
public string GetData(string myValue)
  {
   SqlConnection sqlcon = new
   SqlConnection("Server=SqlServer;Database=testdb;Integrated
Security=SSPI");
   sqlcon.Open();
   //do the business operation
   string identity =
OperationContext.Current.IncomingMessageHeaders.GetHeader<string>("identity",
"ns");
   return “some data” ;
   }
```

Configuration

- The service has a binding endpoint that uses **netTcpbinding** with the default settings.
- The service has a service behavior configuration to publish metadata.
- The service has a base address configured.
- The service behavior is configured with element serviceMedata to allow metadata exposure.

```
<system.serviceModel>
    <behaviors>
        <serviceBehaviors>
            <behavior name="BehaviorConfiguration">
                <serviceMetadata />
            </behavior>
        </serviceBehaviors>
    </behaviors>
    <bindings />

    <services>
        <service behaviorConfiguration="BehaviorConfiguration"
                 name="WCFServicecHost.MyService">
            <endpoint address="Mex"
                      binding="mexTcpBinding"
                      bindingConfiguration=""
                      name="MexEndpoint"
                      contract="IMetadataExchange" />
            <endpoint address=""
                      binding="netTcpBinding"
                      bindingConfiguration=""

                     name="TcpBinding"
                     contract="WCFServicecHost.IMyService" />
```

```
            <host>
                <baseAddresses>
                    <add
                  baseAddress="net.tcp://perfpres02.npscode.com/MyService" />
                </baseAddresses>
            </host>
        </service>
    </services>
</system.serviceModel>
```

Database Server

Configuration

- A SQL server login is created for the WCF service account.
- The WCF login name is given access to the database.
- The role is created in the database.
- The WCF login name is added to the role.

```
-- Create a SQL Server login  that matches the WCF machine name
EXEC SP_GRANTLOGIN 'npscode\perfpres02$'

-- Grant the login access to the application database
use testdb
go
exec sp_grantdbaccess 'npscode\perfpres02$'

-- Create the new database role
use testdb
go
exec sp_addrole 'myrole2','db_owner'

-- Add the new login to the role
use testdb
go
exec sp_addrolemember 'myrole2','npscode\aspnethost'
```

Additional Resources

- For more information on security authentication best practices, see “Best Practices for Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms731059.aspx
- For additional information on message security, see “Message Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For more information on hosting in a Windows service, see the document “How To: Host WCF in a Windows Service.”
- For more information on WCF hosting considerations, see “Hosting Services” at http://msdn2.microsoft.com/en-us/library/ms730158.aspx
- For more information on **netTcpBinding** configuration options see “<netTcpBinding>” at http://msdn2.microsoft.com/en-us/library/ms731343.aspx

Chapter 15 - Internet – Windows Forms Client to Remote WCF Using Message Security (Original Caller, HTTP)

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5

Scenario

In this scenario, your users do not have Microsoft® Windows® accounts and use a Windows Forms client to connect over the Internet to your WCF service. The business logic called by the WCF service is backed by a SQL Server data store. The basic model for this application scenario is shown in the following figure.

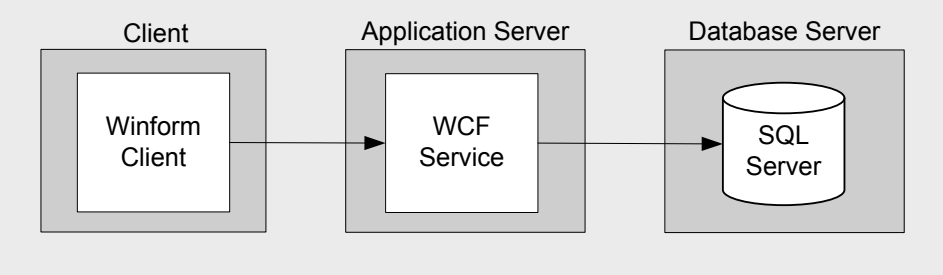


Figure 1. Windows Forms Client to Remote WCF Using Message Security (Original Caller, HTTP) – Model

Key Characteristics

This scenario applies to you if:

- Your users have Windows Forms clients.
- Your user accounts are stored in SQL.
- Your user roles are stored in SQL.
- Your application transmits sensitive data over the network that needs to be protected.
- The ability to host the WCF service in IIS is more important than a high performance connection between the ASP.NET application and the WCF service.

Solution

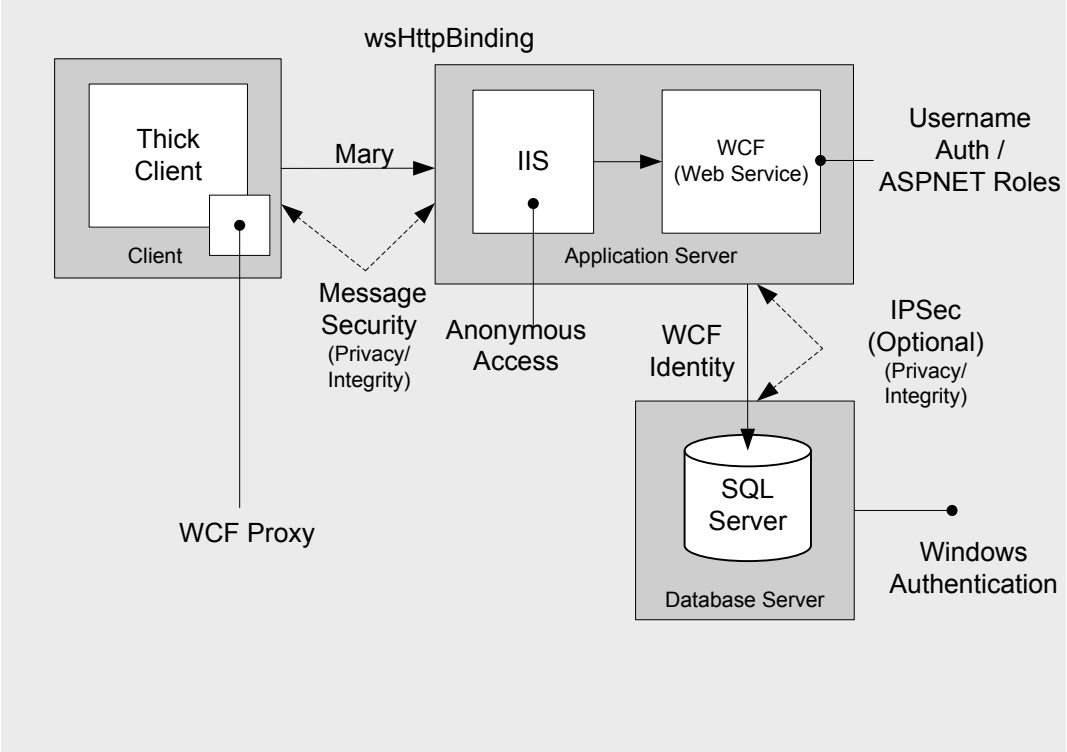


Figure 2. Windows Forms Client to Remote WCF Using Message Security (Original Caller, HTTP) – Solution

Solution Summary Table

In this solution you will:

- Use username and password to authenticate users against the SQL Server Membership Provider.
- Use a service account to call the SQL Server from WCF.
- Use message security to protect sensitive data between the ASP.NET application and the WCF service.
- Use **wsHttpBinding** to allow IIS to host the service.
- Host WCF in IIS.

Thick Client

Checks	Example
WCF Proxy	
Thick Client has a proxy reference to the WCF service. The application has access to the WCF metadata to create a service reference.	`WCFTestService.ServiceClient myService = new WCFTestService.ServiceClient();`

Checks	Example
Root CA certificate for the service is installed in "Trusted Root Certification Authorities." All certificates that are signed with this certificate will be trusted by the client machine.	

Pass user credentials to the WCF service when calling service operations.
 	
A proxy will invoke a WCF method within the service contained on the application server using the Service Accounts security context.
	`myService.ClientCredentials.UserName.UserName = "username";`
	`myService.ClientCredentials.UserName.Password = "p@ssw0rd";`
	`myService.GetData(123);`

Application Server

Checks	Example
IIS - Configuration	
A dedicated application pool is created and configured to run under a custom service account. Use a domain account if possible.	
The WCF Service is configured to run under the service account. Assign the WCF Service to the custom application pool.	
IIS - Authentication	
The IIS virtual directory is configured to use Anonymous access.	

Checks	Example
WCF Service - Configuration	
ASP.NET database is created for use with SQL Membership Provider and SQL Role provider. Aspnet_regsql.exe creates the SQL database to store the user and role information.	`aspnet_regsql -S .\SQLExpress -E -A r m`
Connection string is configured to point to the user and role stored in SQL Server. The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes for Windows Authentication.	`<add name="MyLocalSQLServer" connectionString="Initial Catalog=aspnetdb;data source=localhost;Integrated Security=SSPI;" />`

SqlMembershipProvider is configured to use with Membership.
 	
The membership feature helps protect credentials, can enforce strong passwords, and provides consistent APIs for user validation and secure user management.
	`<membership defaultProvider="MySqlMembershipProvider">`
	`  <providers>`
	`  <clear/>`
	`  <add name="MySqlMembershipProvider"`
	`  connectionStringName="MyLocalSQLServer"`
	`  applicationName="MyAppName"`
	`  type="System.Web.Security.SqlMembershipProvider"/>`
	`  </providers>`
	`</membership>`

Role Manager feature is enabled and SqlRoleProvider is configured for roles authorization.
 	
Role Manager allows you to look up users' roles without writing and maintaining custom code.
	`<roleManager enabled="true" defaultProvider="MySqlRoleProvider" >`
	`  <providers>`
	`  <clear/>`
	`  <add name="MySqlRoleProvider"`
	`  connectionStringName="MyLocalSQLServer"`
	`  applicationName="MyAppName"`
	`  type="System.Web.Security.SqlRoleProvider" />`
	`  </providers>`
	`</roleManager>`

Checks	Example

WCF Service process identity is given access permissions on the ASP.NET database.
 	
Your WCF service process identity requires access to the Aspnetdb database.
	`-- Create a SQL Server login for the Network Service account`
	`sp_grantlogin '<<Custom Service Account>>'`
	
	`-- Grant the login access to the membership database`
	`USE aspnetdb`
	`GO`
	`sp_grantdbaccess '<<Custom Service Account>>',`
	`'<<Custom Service Account>>'`
	
	`-- Add user to database role`
	`USE aspnetdb`
	`GO`
	`sp_addrolemember 'aspnet_Membership_FullAccess',`
	`'<<Custom Service Account>>'`
	
	`sp_addrolemember 'aspnet_Roles_FullAccess', '<<Custom Service Account >>'`

WCF Service is configured to use **wsHttpBinding** binding.
 	
wsHttpBinding uses the HTTP protocol and provides full support for SOAP security, transactions, and reliability. As clients are in the Internet, this is the only choice.
	`<endpoint address="" binding="wsHttpBinding"`
	`bindingConfiguration="BindingConfiguration"`
	`name="WsBinding" contract="IService"/>`

WCF Service - Authentication	

wsHttpBinding is configured to use Username Authentication and Message security.	`<wsHttpBinding>`
	`<binding name="BindingConfiguration">`
	`<security>`
	`<message clientCredentialType="UserName" />`
	`</security>`
	`</binding>`
	`</wsHttpBinding>`

SqlMembershipProvider is configured to provide user authentication.
 	
The membership feature automatically authenticates and creates the authentication ticket for you.
	`<membership defaultProvider="MySqlMembershipProvider">`
	`<providers>`
	`<clear/>`
	`<add name="MySqlMembershipProvider"`
	`connectionStringName="MyLocalSQLServer"`
	`applicationName="MyAppName"`
	`type="System.Web.Security.SqlMembershipProvider"/>`
	`</providers>`
	`</membership>`

Checks	Example
Service behavior is configured to use membership provider for using with username authentication.	`<userNameAuthentication userNamePasswordValidationMode="MembershipProvider" membershipProviderName="MySqlMembershipProvider" />`
Service behavior is configured to publish metadata.	`<serviceMetadata httpGetEnabled="true" />`
Service certificate is installed on the WCF Service machine. The service behavior is configured to use the service certificate. This is required for protecting the user credentials in the message.	`<serviceCertificate findValue="CN=machine.domain.com" />`
WCF Service - Authorization	
Service behavior is configured to use AspNetRoles with SqlRoleProvider.	`<serviceAuthorization principalPermissionMode="UseAspNetRoles" roleProviderName="MySqlRoleProvider" />`

WCF Operations are configured to do role checks at operation level, declaratively.
 	
Declarative role-checks on operations is the preferred mechanism.
	`[PrincipalPermission(SecurityAction.Demand, Role="Managers")]`
	`public string GetData(int value)`
	`{`
	`  return string.Format("You entered: {0}", value);`
	`}`

Roles APIs are used to do programmatic roles checks, for fine grained access control.
 	
If you need finer-grained authorization control, you can use imperative role checks in the code itself. Use call to Roles.IsUserInRole to perform the check.
	`If(Roles.IsUserInRole("Manager"))`
	`{`
	`  // do something for the manager`
	`}`
	`else`
	`{`
	`  // throw an error.`
	`}`

WCF Service - SQL	

Checks	Example
The connection string for database is configured to use Windows Authentication. The database connection string includes Integrated Security=SSPI or Trusted Connection=Yes.	`SqlConnection sqlcon = new SqlConnection("Server=10.3.19.11;Database=Northwind;IntegratedSecurity=SSPI");`
Open the database connection using the WCF process identity's security context. Service does not impersonate the original caller to benefit for connection pooling.	

Database Server

Check	Example
Configuration	
A SQL Server login is created for the WCF's service account (process identity). This grants access to the SQL Server.	`exec sp_grantlogin 'Custom Service Account'`

The login is mapped to a database user for the Web application.
 	
This grants access to the specified database.
	`use targetDatabase`
	`go`
	`exec sp_grantdbaccess ' Custom Service Account'`
	`go`

A database role is created in the target database.
 	
This allows access control and authorization to the DB.
	`use targetDatabase`
	`go`
	`exec sp_addrole 'DB Role Name'`
	`go`

The login is added to the database role.
 	
Grant minimum permissions. For example, grant execute permissions to selected stored procedures and provide no direct table access.
	`use targetDatabase`
	`go`
	`exec sp_addrolemember 'DB Role Name', 'Custom Service Account'`
	`go`

Authentication	

Check	Example
SQL Server is configured to use Windows Authentication.	

Communication Security

What	Check
App Server to Database Server	You can use IPSec or SSL between the App Server and Database Server to protect sensitive data on the wire.

Analysis

Thick Client

WCF Proxy

- Because original user's credentials are required in WCF for Authentication and Authorization, username credentials are set on the WCF proxy and all calls to the WCF service are made through that proxy instance.
- For validating the service certificate, the Root CA certificate is installed on the client machine in the "Trusted Root Certification Authorities" location.

Application Server

Authentication

- As the users are coming from the Internet and you cannot assume they have a Windows account, the user information is stored in SQL. For this reason, WCF is configured to use Username Authentication to authenticate its callers.
- The membership feature is a good choice as it allows you to enable user name authentication without writing and maintaining custom code.
- To protect the user credentials over the wire, a Service Certificate is installed and configured to be used as Service Credentials in WCF.

Authorization

- For coarse grained access control, authorization checks are performed in the WCF Service at the operation level, declaratively.
- For fine grained access control or implementing business logic, authorization checks are made within the operations programmatically.
- The Roles Manager is a good choice for this scenario because it allows you to look up users' roles without writing and maintaining custom code.

Data Access

- To reduce the chances of database credentials being stolen, the database connection string is configured to use Windows authentication. This choice avoids storing credentials in files and passing credentials over the network to the Database Server.
- The WCF service accesses the database using the WCF process identity. As a result, all calls are made using the single process account and database connection pooling to be used.

Configuration

- Since all of the clients are coming from the Internet, the best transport protocol for this scenario is HTTP. For this reason, **wsHttpBinding** is an ideal choice.
- Because **wsHttpBinding** is supported by IIS 6.0, Microsoft hosted the WCF service in IIS.
- In order to reduce attack surface and minimize the impact of a compromise, the WCF Service is running under the security context of the Service account using a least privileged account.

Database Server

- SQL Server database user roles are preferred to SQL server application roles to avoid the associated password management and connection pooling issues associated with the use of SQL application roles. Applications activate SQL application roles by calling a built-in stored procedure with a role name and a password. Therefore, the password must be stored securely. Database connection pooling must also be disabled when you use SQL application roles, which severely impacts application scalability.
- Creating a new user-defined database role and adding the database user to the role lets you give specific minimum permissions to the role. In this way, if the database account changes you don't have to change the permissions on all database objects.

Communication Security

- Message security protects sensitive data between the Thick Client and WCF Service.
- You can use IPSec or SSL between the WCF Service and the Database Server to protect sensitive data on the wire.

Example

Application Server

Code

- The service performs imperative authorization checks calling **Roles.IsUserInRole**.
- If auditing is required the service retrieves the identity of the caller.
- The service calls SQL using Windows Authentication.

```
using System.Data.SqlClient;
```

```
using System.Web.Security;


public string GetData(int value)
{
 if (Roles.IsUserInRole(@"accounting"))
   {
    SqlConnection sqlcon = new
SqlConnection("Server=10.3.19.60;Database=testdb;Integrated
Security=SSPI");
    sqlcon.Open();

    string identity = ServiceSecurityContext.Current.PrimaryIdentity.Name;
    return “data”
        }
    else return "not authorized";
   }
}
```

Configuration

- The service has a binding endpoint that uses **wsHttpbinding** with binding configuration that enables message security and username authentication.
- The service configuration file has an entry with a connection string pointing to the SQL store for authentication and authorization.
- The service configuration file has an entry for the **SqlRoleProvider** under system.web to define which role provider is being used.
- The service configuration file has an entry for the **SqlMemberShipProvider** under system.web to define the SQL provider for authentication.
- The service has a service behavior to use the **SqlMemberShipProvider**.
- The service behavior is configured with the element **serviceAuthorization** to allow **UseAspNetRoles** as the authorization provider.
- The service behavior is configured with the element **serviceMetadata** to allow publishing metadata.
- The service behavior is configured to use a certificate to encrypt the messages.

```
<configuration>
…
<connectionStrings>
    <add name="MyLocalSQLServer"
         connectionString="Initial Catalog=aspnetdb;data
source=10.3.19.60;Integrated Security=SSPI;"/>
  </connectionStrings>

    <system.web>


      <membership defaultProvider="MySqlMembershipProvider" >
        <providers>
          <clear/>
```

```
          <add name="MySqlMembershipProvider"
               connectionStringName="MyLocalSQLServer"
               applicationName="MyAppName"
               type="System.Web.Security.SqlMembershipProvider" />
        </providers>
      </membership>

      <roleManager enabled="true" defaultProvider="MySqlRoleProvider" >
        <providers>
          <clear/>
          <add name="MySqlRoleProvider"
               connectionStringName="MyLocalSQLServer"
               applicationName="MyAppName"
               type="System.Web.Security.SqlRoleProvider" />
        </providers>
      </roleManager>

          <assemblies>
            <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Xml.Linq, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0,
Culture=neutral,PublicKeyToken=B77A5C561934E089"/>
          </assemblies>

        </compilation>

      <pages>
        <controls>
          <add tagPrefix="asp" namespace="System.Web.UI"
assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
        </controls>
      </pages>

      <httpHandlers>
        <remove verb="*" path="*.asmx"/>
        <add verb="*" path="*.asmx" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
        <add verb="*" path="*_AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
        <add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
validate="false"/>
      </httpHandlers>
      <httpModules>
        <add name="ScriptModule" type="System.Web.Handlers.ScriptModule,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
```

```
      </httpModules>

    </system.web>

  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding name="BindingConfiguration">
          <security>
            <message clientCredentialType="UserName" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <services>
      <service behaviorConfiguration="BehaviorConfiguration"
name="Service">
        <endpoint address="" binding="wsHttpBinding"
bindingConfiguration="BindingConfiguration"
          name="WsBinding" contract="IService" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="BehaviorConfiguration">
          <serviceAuthorization principalPermissionMode="UseAspNetRoles"
            roleProviderName="MySqlRoleProvider" />
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true" />
          <serviceCredentials>
            <serviceCertificate findValue="CN=perfpres02.npscode.com" />
            <userNameAuthentication
userNamePasswordValidationMode="MembershipProvider"
              membershipProviderName="MySqlMembershipProvider" />
          </serviceCredentials>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Client

Code

- Client passes user credentials explicitly when making calls to the service.

```
WCFTestService.ServiceClient myService = new
WCFTestService.ServiceClient();
myService.ClientCredentials.UserName.UserName = "username";
myService.ClientCredentials.UserName.Password = "p@ssw0rd";
myService.GetData(123);
myService.Close();
```

Database Server

Configuration

- A SQL Server login is created for the WCF Service account.
- The WCF login name is given access to the application database.
- The role is created in the application database.
- The WCF login name is added to the role.

```
-- Create a SQL Server login  that matches the WCF machine name
EXEC SP_GRANTLOGIN 'npscode\perfpres02$'

-- Grant the login access to the application database
use testdb
go
exec sp_grantdbaccess 'npscode\perfpres02$'

-- Create the new database role
use testdb
go
exec sp_addrole 'myrole2','db_owner'

-- Add the new login to the role
use testdb
go
exec sp_addrolemember 'myrole2','npscode\perfpres02$'
```

Additional Resources

- For more information on Windows Communication Foundation Role Service Overview, see http://msdn2.microsoft.com/en-us/library/bb386424.aspx
- For more information on ASP.NET: Understanding Role Management, see http://msdn2.microsoft.com/en-us/library/5k850zwb.aspx
- For more information on Windows Authentication, see “Explained: Windows Authentication in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/aa480475.aspx
- For more information on debugging authentication errors, see “Debugging Windows Authentication Errors” at http://msdn2.microsoft.com/en-us/library/bb463274.aspx
- For more information on security authentication best practices, see “Best Practices for Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms731059.aspx
- For additional information on message security, see “Message Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx

WCF Security Checklist

Design Considerations

Check	Description
☐	Consider Exposing Different Endpoints
☐	If you need to support ASMX clients, use basicHttpBinding
☐	If you are migrating from DCOM then use netTcpBinding
☐	If you need to support legacy WSE clients then use a customBinding in WCF
☐	Consider transport security as your preferred security mode
☐	Know your Authentication options
☐	Know your Authorization options
☐	Know your binding options
☐	Choose the right binding for your scenario

Auditing and Logging

Check	Description
☐	Use WCF auditing to audit your service
☐	If non-repudiation is important, consider setting SuppressAuditFailure property to false
☐	Use message logging to log operations on your service
☐	Instrument for user management events
☐	Instrument for significant business operations
☐	Protect log files from unauthorized access
☐	Do not log sensitive information
☐	Protect information in log files
☐	Use a Custom Trace Listener only when message filtering is needed

Authentication

Check	Description
☐	Know your authentication options

☐	Use Windows Authentication when you can
☐	If your users are in AD, but you can't use windows authentication, consider using username authentication
☐	If you are using username authentication, use Membership Provider instead of custom authentication
☐	If your users are in a SQL membership store, use the SQL Membership Provider
☐	If your users are in a custom store, consider using username authentication with a custom validator
☐	If your clients have certificates, consider using client certificate authentication
☐	If your partner applications need to be authenticated when calling WCF services, use client certificate authentication.
☐	If you are using username authentication, validate user login information
☐	Do not store passwords directly in the user store
☐	Enforce strong passwords
☐	Protect access to your credential store
☐	If you are using client certificate authentication, consider reducing the attack surface by limiting the certificates in the certificate store

Authorization

Check	Description
☐	If you store role information in Windows Groups
☐	If You Use ASP.NET Roles
☐	If you use Windows groups for authorization
☐	If you store role information in SQL
☐	If you store role information in ADAM
☐	If you store role information in a custom store
☐	If you need to authorize access to WCF operations
☐	If you need to perform fine-grained authorization based on business logic

Bindings

Check	Description
☐	If you need to support clients over the internet

☐	If you need to expose your WCF service to legacy clients as an ASMX web service
☐	If you need to support WCF clients within an intranet
☐	If you need to support WCF clients on the same machine
☐	If you need to support disconnected queued calls
☐	If you need to support bidirectional communication between WCF Client and WCF service

Configuration Management

Check	Description
☐	Use Replay detection to protect against message replay attacks
☐	If you host your service in a Windows service, expose a metadata exchange (mex) binding
☐	If you don't want to expose your WSDL, turn off HttpGetEnabled and metadata exchange (mex)
☐	Encrypt configuration sections that contain sensitive data

Exception Management

Check	Description
☐	Use structured exception handling
☐	Do not divulge exception details to clients in production
☐	Use a fault contract to return error information to clients
☐	Use a Global Exception Handler with IErrorHandler to Catch Unhandled Exceptions

Hosting

Check	Description
☐	Run your service in a least privileged account
☐	Use IIS to host your service unless you need to use a transport that IIS does not support

Impersonation/Delegation

Check	Description
☐	Know Your Tradeoffs with Impersonation
☐	Know Your Impersonation Options
☐	Know Your Impersonation Methods
☐	Consider Using Programmatic Instead of Declarative Impersonation
☐	When Impersonating Programmatically be Sure to Revert to Original Context
☐	When Impersonating Declaratively, Only Impersonate on the Operations That Require It
☐	Consider Using S4U Feature for Impersonation and Delegation, When You Cannot do a Windows Mapping
☐	Consider Using LogonUser API, If Your WCF Service Cannot be Trusted for Delegation
☐	If You Have to Flow the Original Caller to the Backend Services, Use Constrained Delegation

Input/Data Validation

Check	Description
☐	If You Need To Validate Parameters, Use Parameter Inspectors
☐	Use Schemas to Validate Messages, Using Message Inspectors
☐	Use Regular Expressions in Schemas to Validate Format, Range or Length
☐	Implement AfterReceiveRequest Method to Validate Inbound Messages on the Service
☐	Implement BeforeSendReply Method to Validate Outbound Messages on the Service
☐	Implement AfterReceiveReply Method to Validate Inbound Messages on the Client
☐	Implement BeforeSendRequest Method to Validate Outbound Messages on the Client
☐	Validate Operation Parameters for Length, Range, Format and Type
☐	Do Not Rely on Client-side Validation
☐	Avoid User-supplied File Name and Path Input
☐	Do Not Echo Untrusted Input

Message Security

Check	Description
☐	If You Need to Support Clients Over the Internet, Consider Using Message Security
☐	If You There are Intermediaries between Client and Service, Consider Using Message Security
☐	If you Need to Support Selective Message Protection, Use Message Security

☐	If You Need to Support Multiple Transactions Per Session Using Secure Conversation, Use Message Security
☐	Do Not Pass Sensitive Information In SOAP Headers When Using Http Transport and Message Security
☐	If You Need to Support Interoperability, Consider Setting negotiateServiceCredentials to False
☐	If You Need to Streamline Certificate Distribution to Your Clients, Consider Negotiating the Service Credentials
☐	If You Need to Limit the Clients that Will Consume Your Service, Consider Setting negotiateServiceCredentials to False

Transport Security

Check	Description
☐	Use Transport Security When Possible
☐	If You Need to Support Clients in an Intranet, Use Transport Security
☐	If You need to Support Interoperability with Non-WCF Clients, Use Transport Security
☐	Use Hardware Accelerator When Using Transport Security

Proxy Considerations

Check	Description
☐	Publish Your WCF Service Metadata Only When Required
☐	If You Need to Publish Your WCF Service Metadata, Publish it Over HTTPS Protocol
☐	If You Need to Publish Your WCF Service Metadata, Publish it Using Secure Binding
☐	If You Turn Off Mutual Authentication, Be Aware of Service Spoofing

Sensitive Data

Check	Description
☐	Avoid Plain Text Passwords or Other Sensitive Data in Configuration Files
☐	Use Platform Features to Manage Keys Where Possible
☐	Protect Sensitive Data Over the Wire
☐	Do Not Cache Sensitive Data

☐	Minimize Exposure of Secrets in Memory
☐	Be Aware That basicHttpBinding Will Not Protect Sensitive Data by Default
☐	Use Appropriately Sized Keys

Deployment Considerations

Check	Description
☐	Do Not Use Temporary Certificates in Production
☐	If You are Using Kerberos Authentication or Delegation, Create an SPN
☐	Use IIS to Host Your WCF Service Wherever Possible
☐	Use a Least Privileged Account to Run Your WCF Service
☐	Protect sensitive data in your configuration files

WCF Security Guidelines

Index

Design Considerations

- Consider Exposing Different Endpoints
- If You Need to Support ASMX Clients, Use basicHttpBinding
- If You Are Migrating from DCOM, Use netTcpBinding
- If You Need to Support Legacy WSE Clients, Use a customBinding in WCF
- If You Require Interoperability with Non-Microsoft Clients, Use Bindings That Are Targeted for Interoperability
- If Your Non-Microsoft Clients Understand the WS* Stack, Use ws2007HttpBinding or wsHttpBinding
- Consider Transport Security as Your Preferred Security Mode
- Know Your Authentication Options
- Know Your Authorization Options
- Know Your Binding Options
- Choose the Right Binding for Your Scenario

Auditing and Logging

- Use WCF Auditing to Audit Your Service
- If Non-repudiation Is Important, Consider Setting the SuppressAuditFailure Property to false
- Use Message Logging for Debugging Purposes
- Instrument for User Management Events
- Instrument for Significant Business Operations
- Protect Log Files from Unauthorized Access
- Do Not Log Sensitive Information
- Protect Information in Log Files
- Use a Custom Trace Listener Only When Message Filtering Is Needed

Authentication

- Know Your Authentication Options
- Use Windows Authentication When You Can
- If Your Users Are in Active Directory but You Can’t Use Windows Authentication, Consider Using Username Authentication
- If You Are Using Username Authentication, Use a Membership Provider Instead of Custom Authentication
- If Your Users Are in a SQL Membership Store, Use the SQL Server Membership Provider
- If Your Users Are in a Custom Store, Consider Using Username Authentication with a Custom Validator
- If Your Clients Have Certificates, Consider Using Client Certificate Authentication

- If Your Partner Applications Need to Be Authenticated When Calling WCF Services, Use Client Certificate Authentication
- If You Are Using Username Authentication, Validate User Login Information
- Do Not Store Passwords Directly in the User Store
- Enforce Strong Passwords
- Protect Access to Your Credential Store
- If You Are Using Client Certificate Authentication, Limit the Certificates in the Certificate Store

Authorization

- If You Store Role Information in Windows Groups, Consider Using the WCF PrincipalPermissionAttribute Class for Role Authorization
- If You Use ASP.NET Roles, Use the ASP.NET Role Manager for Role Authorization
- If You Use Windows Groups for Authorization, Use the ASP.NET Role Provider with AspNetWindowsTokenRoleProvider
- If You Store Role Information in SQL Server, Consider Using the SQL Server Role Provider for Role Authorization
- If You Store Role Information in ADAM, Use the Authorization Manager Role Provider
- If You Store Role Information in a Custom Store, Create a Custom Authorization Policy
- If You Need to Authorize Access to WCF Operations, Use Declarative Authorization
- If You Need to Perform Fine-grained Authorization Based on Business Logic, Use Imperative Authorization

Bindings

- If You Need to Support Clients over the Internet, Consider Using wsHttpBinding
- If You Need to Expose Your WCF Service to Legacy Clients as an ASMX Web Service, Use basicHttpBinding
- If You Need to Support WCF Clients Within an Intranet, Consider Using netTcpBinding
- If You Need to Support WCF Clients on the Same Machine, Consider Using netNamedPipeBinding
- If You Need to Support Disconnected Queued Calls, Use netMsmqBinding
- If You Need to Support Bidirectional Communication Between a WCF Client and WCF Service, Use wsDualHttpBinding or netTcpBinding

Configuration Management

- Use Replay Detection to Protect Against Message Replay Attacks
- If You Host Your Service in a Windows Service, Expose a Metadata Exchange (mex) Binding
- If You Don’t Want to Expose Your WSDL, Turn Off HttpGetEnabled and Metadata Exchange (mex)
- Encrypt Configuration Sections That Contain Sensitive Data

Exception Management

- Use Structured Exception Handling
- Do Not Divulge Exception Details to Clients in Production
- Use a Fault Contract to Return Error Information to Clients
- Use a Global Exception Handler to Catch Unhandled Exceptions

Hosting

- Run Your Service in a Least-Privileged Account
- Use IIS to Host Your Service, Unless You Need to Use a Transport That IIS Does Not Support

Impersonation/Delegation

- Know the Tradeoffs Involved in Impersonation
- Know Your Impersonation Options
- Know Your Impersonation Methods
- Consider Using Programmatic Instead of Declarative Impersonation
- When Impersonating Programmatically, Be Sure to Revert to the Original Context
- When Impersonating Declaratively, Only Impersonate on the Operations That Require It
- Consider Using the S4U Feature for Impersonation and Delegation When You Cannot Do a Windows Mapping
- Consider Using the LogonUser API If Your WCF Service Cannot Be Trusted for Delegation
- Use Constrained Delegation if You Have to Flow the Original Caller to the Back-end Services

Message Validation

- If You Need to Validate Parameters, Use Parameter Inspectors
- Use Schemas with Message Inspectors to Validate Messages
- Use Regular Expressions in Schemas to Validate Format, Range, or Length
- Implement the AfterReceiveRequest Method to Validate Inbound Messages on the Service
- Implement the BeforeSendReply Method to Validate Outbound Messages on the Service
- Implement the AfterReceiveReply Method to Validate Inbound Messages on the Client
- Implement the BeforeSendRequest Method to Validate Outbound Messages on the Client
- Validate Operation Parameters for Length, Range, Format, and Type
- Do Not Rely on Client-side Validation
- Avoid User-supplied File Name and Path Input
- Do Not Echo Untrusted Input

Message Security

- If You Need to Support Clients over the Internet, Consider Using Message Security
- If There Are Intermediaries Between the Client and Service, Consider Using Message Security
- If You Need to Support Selective Message Protection, Use Message Security
- If You Need to Support Multiple Transactions per Session Using Secure Conversation, Use Message Security

- Do Not Pass Sensitive Information in SOAP Headers When Using HTTP Transport and Message Security
- If You Need to Support Interoperability, Consider Setting negotiateServiceCredentials to false
- If You Need to Streamline Certificate Distribution to Your Clients, Consider Negotiating the Service Credentials
- If You Need to Limit the Clients that Will Consume Your Service, Consider Setting negotiateServiceCredentials to false

Transport Security

- Use Transport Security When Possible
- If You Need to Support Clients in an Intranet, Use Transport Security
- If You Need to Support Interoperability with Non-WCF Clients, Use Transport Security
- Use a Hardware Accelerator When Using Transport Security

Proxy Considerations

- Publish Your WCF Service Metadata Only When Required
- If You Need to Publish Your WCF Service Metadata, Publish It over the HTTPS Protocol
- If You Need to Publish Your WCF Service Metadata, Publish It Using Secure Binding
- If You Turn Off Mutual Authentication, Be Aware of Service Spoofing

Sensitive Data

- Avoid Plain-Text Passwords or Other Sensitive Data in Configuration Files
- Use Platform Features to Manage Keys Where Possible
- Protect Sensitive Data Over the Network
- Do Not Cache Sensitive Data
- Minimize Exposure of Secrets in Memory
- Be Aware That basicHttpBinding Will Not Protect Sensitive Data by Default
- Use Appropriately Sized Keys

Deployment Considerations

- Do Not Use Temporary Certificates in Production
- If You Are Using Kerberos Authentication or Delegation, Create an SPN
- Use IIS to Host Your WCF Service Wherever Possible
- Use a Least-Privileged Account to Run Your WCF Service
- Protect Sensitive Data in Your Configuration Files

Design Considerations

The key issue to consider at design time is what binding you will choose for your particular scenario. Choosing an appropriate binding is important from a security perspective because it drives your security choices — for example, transfer security — which in turn determine the confidentiality, integrity, and authentication of your messages. Additionally, you need to

consider your authentication and authorization options and decide which option makes sense for your scenario.

When designing your WCF service, you should:

- **Consider exposing different endpoints.**
- **If you need to support ASMX clients, use basicHttpBinding.**
- **If you are migrating from DCOM, use netTcpBinding.**
- **If you need to support legacy WSE clients, use a customBinding in WCF.**
- **If you require interoperability with non-Microsoft clients, use bindings that are targeted for interoperability.**
- **If your non-Microsoft clients understand the WS* stack, use ws2007HttpBinding or wsHttpBinding.**
- **Consider transport security as your preferred security mode.**
- **Know your authentication options.**
- **Know your authorization options.**
- **Know your binding options.**
- **Choose the right binding for your scenario.**

Each of these guidelines is described in the following sections.

Consider Exposing Different Endpoints

Consider exposing multiple endpoints to support different authentication schemes or protocols, or to support ASP.NET Web Services (ASMX) clients. This approach gives you flexibility as you will only have to develop the service once, while still providing clients with the endpoint that matches their security requirements.

If you need to support ASMX clients, use basicHttpBinding

If you have an ASMX client base, consider using **basicHttpBinding** in your service because it can be consumed by both WCF and legacy clients. **basicHttpbinding** is a flexible way to provide support for existing legacy clients, as it does not require that they be upgraded to WCF. Additionally, **basicHttpbinding** provides a wide range of authentication schemes. Transport security supports Basic, Digest, Windows, and certificate authentication; message security supports username and certificate authentication. To use **basicHttpBinding**, create an endpoint with binding configuration set to use **basicHttpBinding**. Security mode is turned off by default. To enable security and authentication, you will need to create a binding configuration and then configure the endpoint to use the binding.

If You Are Migrating from DCOM, Consider Using netTcpBinding

If you are migrating from DCOM, consider using **netTcpBinding**. Because **netTcpBinding** uses binary encoding and the TCP protocol, it provides the best performance for .NET-to-.NET cross-machine communication. Consider the following when choosing **netTcpBinding**:

- It supports transactions and reliable messaging.

- It supports a wide range of authentication schemes.
- If you use transport security mode, you can use Windows and certificate authentication.
- If you use message or mixed-mode security, you can use Windows, issue token, username, and certificate authentication.
- You can host in Internet Information Services (IIS) 7.0 or a Windows service. You can also host in IIS 6.0, but you have to activate the host W3wp process before using the service.

If You Need to Support Legacy WSE Clients, Use a CustomBinding in WCF

If you have a Web Services Enhancements (WSE) client base, you need to use a custom binding in your service in order to support WSE legacy clients. This will provide interoperability for existing clients without requiring the clients to migrate to WCF. The following table shows the mapping of the WSE bindings to WCF binding elements.

WSE	WCF
anonymousForCertificateSecurity	anonymousForCertficate
kerberosSecurity or mutualCertificateSecurity11	MutualCertificate
usernameOverTransportSecurity	UserNameOverTransport
usernameForCertificateSecurity	usernameForCertificateSecurity

To create a custom binding to support WSE clients:

1. Add a **customBinding** to your WCF service.
2. Specify a name for your custom binding.
3. Specify an authentication mode that maps to the authentication used in your previous WSE service.
4. Specify that WCF uses the August 2004 version of the WS-Addressing specification.
5. Configure the WCF endpoint to use the custom binding.

Additional Resources

For detailed information on how to configure **customBinding**, see "How to: Configure WCF Services to Interoperate with WSE 3.0 Clients" at http://msdn2.microsoft.com/en-us/library/ms730049.aspx

If You Require Interoperability with Non-Microsoft Clients, Use Bindings That Are Targeted for Interoperability

The following bindings are targeted for interoperability:

- **basicHttpBinding**
- **wsHttpBinding**
- **ws2007HttpBinding**
- **ws2007FederationHttpBinding**
- **wsFederationHttpBinding**

If Your Non-Microsoft Clients Understand the WS Stack, Use ws2007HttpBinding or wsHttpBinding*

If you need to support non-Microsoft clients that understand the WS* stack, use **ws2007HttpBinding** or **wsHttpBinding**.

Consider Transport Security as Your Preferred Security Mode

Consider transport security as your preferred security mode, as it provides confidentiality, integrity, and authenticity and may result in better performance. If you use message security or mixed-mode security, consider profiling and testing your application to assess its performance and scalability characteristics, to confirm that it meets your business requirements.

Know Your Authentication Options

Understand the authentication options that map to your particular deployment scenario.

Internet

- **Username authentication with the SQL Server membership provider**. If your users are not in Active Directory, consider using the SQL Server membership provider. This will give you a store that can be easily created and deployed. Configure message security or mixed-mode security to protect your users' credentials.
- **Basic authentication with Windows**. If your users are already in Active Directory or in local machine accounts, consider using Basic authentication. Use transport security to secure the communication channel and protect your credentials.
- **Username authentication with a custom store**. If your users are in a custom store, consider using username authentication with a custom validator in order to validate user credentials against your custom store. Unlike the other scenarios, you will have to write custom code to validate your users' credentials. Use message or mixed-mode security to protect your users' credentials.
- **Certificate authentication with certificates**. If your clients are partners or mobile clients connecting over a virtual private network (VPN) in a peer-to-peer authentication scenario, consider using certificate authentication. If your users have Windows accounts in your domain, you can map the certificates to the Windows accounts and enable authorization checks based on Windows roles. Certificate authentication requires that you manage certificates; however, it allows seamless authentication for clients outside your firewall. Use transport security to secure the communication channel and protect your credentials.

Intranet

- **Username authentication with the SQL Server membership provider**. If your users are not in Active Directory, consider using the SQL Server membership provider. This will give you a store that can be easily created and deployed. Use transport security to secure the communication channel and protect your credentials.
- **Windows authentication with Windows**. If your users are already in Active Directory or in local machine accounts, consider using Windows authentication to leverage this

infrastructure. Windows authentication also gives you the benefits of using Windows roles for authorization checks. Use transport security to secure the communication channel and protect your credentials. Consider that local machine accounts configures authentication with the NTLM protocol, which is prone to brute-force attacks. For more secure peer-to-peer authentication, consider using certificate authentication.

- **Username authentication with a custom store**. If your users are in a custom store, consider using user name authentication with a custom validator in order to validate user credentials against your custom store. Unlike the other scenarios, you will have to write custom code to validate your users' credentials. Use message or mixed-mode security to protect your users' credentials.
- **Certificate authentication with certificates**. If your clients are partners or mobile clients connecting over a VPN in a peer-to-peer authentication scenario, consider using certificate authentication. If your users have Windows accounts in your domain, you can map the certificates to the Windows accounts and enable authorization checks based on Windows roles. Certificate authentication requires that you manage certificates; however, it allows seamless authentication for clients outside your firewall. Use transport security to secure the communication channel and protect your credentials.

Know Your Authorization Options

Know your authorization options and choose the most appropriate option for your particular scenario. First decide if you want to use resource-based or role-based authorization. Resource-based authorization uses access control lists (ACLs) on the resource to authorize the original caller. Role-based allows you to authorize access to service operations or resources based on the group a user is in.

- If you choose to use role-based authorization, you can store your roles in Windows groups or in ASP.NET roles.
- If you are using Active Directory, consider using Windows groups based on ease of maintenance and the fact that you maintain both roles and credentials in the Active Directory store. If you are not using Active Directory, consider using ASP.NET roles and the ASP.NET Role Provider.

Your authorization strategy may also be influenced by your choice of authentication as follows:

- Resource-based authorization
  - If you are using certificate authentication, you will need to map the certificates to Windows groups.
  - If you are using username authentication, you will need to perform protocol transition.
  - Windows authentication will work with resource-based authorization by default.
  - Basic authentication will work with resource-based authorization by default.

  Note: You need to impersonate for resource-based authorization.
- Role-based authorization
  - If you are using certificate authentication, you will need to map the certificates to Windows groups.

- If you are using username authentication with Windows groups, you will need to perform protocol transition.
- Username authentication will work with ASP.NET roles by default.
- Windows authentication will work with Windows groups by default.
- Basic authentication will work with Windows groups by default.

Know Your Binding Options

Know your binding options and choose the most appropriate option for your particular scenario. A thorough understanding of your binding options enables you to design more reliable and secure WCF applications.

The following table summarizes common bindings.

Binding	Description
basicHttpBinding	Represents a binding that configures and exposes endpoints that are able to communicate with ASMX-based Web services and clients and other services that conform to the WS-I Basic Profile 1.1 specification. By default, **basicHttpBinding** has security disabled.
wsHttpBinding	Defines a secure, reliable, interoperable binding suitable for non-duplex service contracts. The binding implements the following specifications: WS-Reliable Messaging for reliability, and WS-Security for message security and authentication. The transport is HTTP, and message encoding is text/XML encoding. By default it provides message security using Windows authentication.
ws2007HttpBinding	Defines a secure, reliable, interoperable binding suitable for non-duplex service contracts. The binding implements the following specifications: WS-Reliable Messaging for reliability, and WS-Security for message security and authentication. The transport is HTTP, and message encoding is text/XML encoding. The **ws2007HttpBinding** provides binding similar to **wsHttpBinding** but uses the standard for OASIS (Organization for the Advancement of Structured Information Standards). By default, **ws2007HttpBinding** provides message security using Windows authentication.
netTcpBinding	Specifies a secure, reliable, optimized binding suitable for cross-machine communication. By default, it generates a run-time communication stack with transport security and Windows authentication as default security settings. **netTcpBinding** uses the Transmission Control Protocol (TCP) for message delivery, and binary message encoding.
netNamedPipeBinding	Defines a binding that is secure, reliable, and optimized for

	cross process communication on the same machine. By default, **netNamedPipeBinding** generates a run-time communication stack with WS-ReliableMessaging for reliability, transport security for transfer security, named pipes for message delivery, and binary message encoding. **netNamedPipeBinding** is not secured by default.
netMsmqBinding	Defines a queued binding suitable for cross-machine communication.
wsFederationHttpBinding	Defines a binding that supports federated security. **wsFederationHttpBinding** helps in implementing federation, which is the ability to flow and share identities across multiple enterprises or trust domains for authentication and authorization. WCF implements federation over message and mixed-mode security but not over transport security. Services configured with this binding must use the HTTP protocol as transport.
ws2007FederationHttpBinding	Defines a binding that derives from **wsFederationHttpBinding** and supports federated security. **ws2007FederationHttpBinding** helps in implementing. WCF implements federation over message and mixed-mode security but not over transport security. Services configured with this binding must use the HTTP protocol as transport. The **ws2007FederationHttpBinding** provides binding similar to **ws2007FederationHttpBinding** but uses the OASIS standard.
wsDualHttpBinding	Defines a secure, reliable, and interoperable binding that is suitable for duplex service contracts or communication through Simple Object Access Protocol (SOAP) intermediaries.
customBinding	Allows you to create a custom binding with full control over the message stack.

For more information on bindings, see “Windows Communication Foundation Bindings” at http://msdn.microsoft.com/en-us/library/ms733027.aspx .

Choose the Right Binding for Your Scenario

Use the following recommendations as a rule of thumb when choosing a binding option. You can fine-tune your selection based on your unique needs and your infrastructure limitations.

- If your service needs to support legacy clients that expect an ASMX Web service, consider using **basicHttpBinding**. Because **basicHttpBinding** does not implement any security by default, if you require message or transport security, you should configure it explicitly on this binding. Use **basicHttpBinding** to expose endpoints that are able to communicate with ASMX-based Web services and clients and other services that conform to the WS-I Basic Profile 1.1 specification. When configuring transport security,

basicHttpBinding defaults to no credentials just like a classic ASMX Web service. **basicHttpBinding** allows you to host your service in IIS 5.0 or IIS 6.0.

- If your service will be called by clients over the Internet, consider using **wsHttpBinding**. **wsHttpBinding** is a good choice for Internet scenarios in which you do not have to support legacy clients that expect an ASMX Web service. If you do need to support legacy clients, consider using **basicHttpBinding** instead. **wsHttpBinding** allows you to host your service in IIS 5.0 or IIS 6.0.
- If you need to support clients within your intranet, consider using **netTcpBinding**. **netTcpBinding** is a good choice for an intranet scenario if transport performance is important to you and it is acceptable to host the service in a Windows service instead of in IIS. **netTcpBinding** uses the TCP protocol and provides full support for message security, transactions, and reliability. Use this binding when you want to provide a secure and reliable binding environment for .NET-to-.NET cross-machine communication. **netTcpBinding** allows you to host your service in IIS 7.0 or a Windows service. You can also host in IIS 6.0, but you must activate the host W3wp process before using the service.
- If you need to support WCF clients on the same machine as your service, consider using **netNamedPipeBinding**. **netNamedPipeBinding** provides a secure and reliable binding environment for cross-process communication on the same machine. Use this binding when you want to make use of the Named Pipe protocol and provide full support for SOAP security, transactions, and reliability. **netNamedPipeBinding** allows you to host your service in IIS 7.0 or a Windows service. You can also host in IIS 6.0, but you must activate the host W3wp process before using the service.
- If you need to support disconnected queuing, use **netMsmqBinding**. Queuing is provided by using Microsoft Message Queuing (MSMQ) as a transport, which enables support for disconnected operations, failure isolation, and load leveling. You can use **netMsmqBinding** when the client and the service do not have to be online at the same time. You can also manage any number of incoming messages by using load leveling. MSMQ supports failure isolation, meaning that messages can fail without affecting the processing of other messages. **netMsmqBinding** allows you to host your service in IIS 7.0 or a Windows service. You can also host in IIS 6.0, but you must activate the host W3wp process before using the service.
- If you need to support a duplex service, use **wsDualHttpBinding**. A *duplex service* is a service that uses duplex message patterns, which provides the ability for a service to communicate back to the client via a callback. You can also use this binding to support communication via SOAP intermediaries.

Auditing and Logging

It is important to audit and log activity across the tiers of your application in order to detect suspicious activity. Using logs frequently provides early indications of a full-blown attack and can help you address the threat of repudiation, where users deny their actions. Log files may be required in legal proceedings to prove the wrongdoing of individuals. Generally, auditing is

considered most authoritative if the audits are generated at the precise time of resource access and by the same routines that access the resource.

Use the following guidelines when implementing auditing and logging in WCF applications:

- **Use WCF auditing to audit your service**
- **If non-repudiation is important, consider setting the SuppressAuditFailure property to false**
- **Use message logging for debugging purposes**
- **Instrument for user management events**
- **Instrument for significant business operations**
- **Protect log files from unauthorized access**
- **Do not log sensitive information**
- **Protect information in log files**
- **Use a Custom Trace Listener only when message filtering is needed**

Each of these guidelines is described in the following sections.

Use WCF Auditing to Audit Your Service

Use the auditing feature in WCF to audit your service. WCF service auditing can allow you to detect an attack that has occurred or is in progress. In addition, auditing can help you debug security-related problems. For example, if an error in the configuration of the authorization or checking policy accidentally denies access to an authorized user, you can discover and isolate the cause of this error by examining the auditing events in the event log.

Configure your application to use the WCF auditing feature to log security events for success, failure, or both. The events are written to the Windows system event log, and you can view and examine them in the Event Viewer.

The following configuration snippet shows how to configure your WCF service to use auditing:

```
<configuration>
  <system.serviceModel>
    <behaviors>
      <behavior>
        <serviceSecurityAudit
             auditLogLocation="Application"
             suppressAuditFailure="true"
             serviceAuthorizationAuditLevel="Failure"
             messageAuthenticationAuditLevel=
                         "SuccessOrFailure" />
      </behavior>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Additional Resources

- For more information on WCF auditing, see "Auditing Security Events" at http://msdn2.microsoft.com/en-us/library/ms731669.aspx
- For auditing security concerns, see "Security Concerns for Message Logging" at http://msdn.microsoft.com/en-us/library/ms730318.aspx

If Non-repudiation is Important, Consider Setting the SuppressAuditFailure Property to false

If non-repudiation is important, consider setting the **SuppressAuditFailure** property to false. This setting will cause an exception to be thrown whenever an audit failure occurs. By default, your WCF service will ignore audit failures and allow the service to continue running. By setting **SuppressAuditFailure** to false, an exception can be thrown and handled by your WCF service.

```
// configuration snippet
<configuration>
  <system.serviceModel>
    <behaviors>
      <behavior>
        <serviceSecurityAudit
            auditLogLocation="Application"
            suppressAuditFailure="false"
            serviceAuthorizationAuditLevel="Failure"
            messageAuthenticationAuditLevel=
                        "SuccessOrFailure" />
      </behavior>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Use Message Logging for Debugging Purposes

Message logging can be used to diagnosed application errors and performance problems. Message logging is not turned on by default, because logging is an expensive operation that can consume disk space and processing time. Turn on message logging by setting attributes on the <messagelogging> element in your configuration file and adding a trace listener to log the events to a file.

Important: Make sure that you limit the number of messages that are written to disk for a particular service, as disk space could be a limiting factor. When the message limit is reached, a trace at the Information level is produced and all message-logging activities stop.

The following configuration code enables message logging by creating a **ServiceModelMessageLoggingListener** and **System.ServiceModel.MessageLogging** source:

```
…
<configuration>
<system.diagnostics>
  <sources>
   <source name="System.ServiceModel.MessageLogging" switchValue="Warning,
ActivityTracing">
```

```
    <listeners>
     <add type="System.Diagnostics.DefaultTraceListener" name="Default">
      <filter type="" />
     </add>
     <add name="ServiceModelMessageLoggingListener">
      <filter type="" />
     </add>
    </listeners>
   </source>
  </sources>
  <sharedListeners>
   <add initializeData="c:\inetpub\wwwroot\WCFService\web_messages.svclog"
    type="System.Diagnostics.XmlWriterTraceListener, System, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"
    name="ServiceModelMessageLoggingListener" traceOutputOptions="Timestamp">
    <filter type="" />
   </add>
  </sharedListeners>
</system.diagnostics>
</configuration>
…
…
<system.serviceModel>
  <diagnostics>
   <messageLogging logEntireMessage="false" logMalformedMessages="true"
    logMessagesAtServiceLevel="false" logMessagesAtTransportLevel="true" />
  </diagnostics>
…
```

Additional Resources

- For more information, see “How To - Audit and Log Security Events in WCF Calling from Windows Forms.”
- For more information on log throttling, see “Configuring Message Logging” at http://msdn2.microsoft.com/en-us/library/ms730064.aspx

Instrument for User Management Events

Use ASP.NET health monitoring to instrument your application and monitor user management events around authentication and authorization. This instrumentation can help you detect and react to potentially suspicious behavior. It also enables you to gather operations data; for example, to track who is accessing your application and when user account passwords need to be reset.

The following steps show how to instrument your application for user management events:

1. Create a custom user management Web event by creating a class library and then creating a class that inherits from **WebAuditEvent**.
2. Configure your WCF service for health monitoring.
3. Instrument the WCF service by raising the custom event in a service contract.
4. Verify the service events in the event log after calling the service method from a test client.

Additional Resources

- For more information, see "How To – Use Health Monitoring to Instrument a WCF Service for Security."
- For more information on health monitoring, see "How To: Use Health Monitoring in ASP.NET 2.0" at http://msdn2.microsoft.com/en-us/library/ms998306.aspx

Instrument for Significant Business Operations

Use ASP.NET health monitoring to instrument your application to track access to sensitive operations. This level of instrumentation can help you detect and react to potentially suspicious behavior. It also enables you to gather operations data; for example, to track what important business operations are being carried out and by whom, etc. For instance, you could track usage of methods that relate to financial transactions or access to sensitive data.

The following steps show how to instrument your application with health monitoring:

1. Create a class library and then create a class that inherits from **WebSuccessAuditEvent** that displays some business information, such as bank account transactions.
2. Configure your WCF service for health monitoring.
3. Instrument the WCF service by raising the custom event in a service contract.
4. Verify the service events in the event log after calling the service method from a test client.

Additional Resources

- For more information, see "How To – Use Health Monitoring to Instrument a WCF Service for Security."
- For more information on health monitoring, see "How To: Use Health Monitoring in ASP.NET 2.0" at http://msdn2.microsoft.com/en-us/library/ms998306.aspx

Protect Log Files from Unauthorized Access

Restrict access to log files and SQL Server records in order to make it more difficult for attackers to tamper with log files and cover their tracks.

Minimize the number of individuals who can manipulate the log files. Authorize access only to highly trusted accounts such as administrators.

Restrict access to audit and log files by using Windows ACLs. If you log events to SQL Server or to some custom event sink, use appropriate access controls to limit access to the event data. For example, grant write access to the account or accounts used by your application, grant full control to administrators, and grant read-only access to operators.

Do Not Log Sensitive Information

Do not log sensitive user or application data to your log files. Permissions on log files are often different than permissions on sensitive data in your data store and operations that access it.

The presence of sensitive data in your logs could allow users to gain access to information to which they would not otherwise have access.

Sensitive data includes, but is not limited to:

- **Personally identifiable information (PII)**. This is data that either contains personally identifiable information or can be used to derive personally identifiable information that should not be shared with users. Examples of personally identifiable information include credit card numbers and social security numbers.
- **User sensitive information.** This is information provided by a user that they would not want shared with other users of the application. This can include user credentials, preferences, or application usage information.
- **Application sensitive information.** This is information that comes from a trusted source that is not designed to be shared with users. Application sensitive information can include connection strings and service account credentials.

Additional Resources

- For more information on protecting sensitive data in logs, see "Security Concerns for Message Logging" at http://msdn2.microsoft.com/en-us/library/ms730318.aspx

Protect Information in Log Files

Protect the information in your log files because it may provide important insight into the internal workings of your application.

The following tips can help you prevent the contents of a log file from being exposed unintentionally:

- Ensure that the log files are protected by access control lists (ACLs) both in Web-hosted and self-hosted scenarios.
- Choose a file extension that cannot be easily served using a Web request. For example, the .xml file extension is not a safe choice. You can check the IIS administration guide to see a list of extensions that can be served.
- Specify an absolute path for the log file location, which should be outside of the Web host vroot public directory to prevent it from being accessed by an external party using a Web browser.

By default, when using message logging, keys and personally identifiable information (PII) (username, password, etc.) and application-specific headers (such as query string) and body information(such as a credit card number) are not logged in traces and logged messages.

Additional Resources

- For more information on protecting sensitive data in logs, see "Security Concerns for Message Logging" at http://msdn2.microsoft.com/en-us/library/ms730318.aspx

Use a Custom Trace Listener to Filter Sensitive Application Data in Messages

Use a custom trace listener only when you need a message filter for filtering application-specific personally identifiable information (PII) elements from messages before logging. Using a custom trace listener with additional options gives you more control over the messages to be logged.

Adding a custom trace listener on the message logging trace source is a privilege that should be restricted to the administrator. This is because malicious custom listeners can be configured to send messages remotely, which leads to disclosure of sensitive information. In addition, if you configure a custom listener to send messages on the network (for example, to a remote database), you should enforce proper access control on the message logs in the remote machine.

The following code example demonstrates a custom listener configuration:

```
<system.diagnostics>
   <sources>
     <source name="System.ServiceModel.MessageLogging">
           <listeners>
             <add name="MyListener"
                    type="YourCustomListener"
                    initializeData="c:\logs\messages.svclog"
                    maxDiskSpace="1000"/>
           </listeners>
     </source>
   </sources>
</system.diagnostics>
```

Additional Resources

- For more information, see “Security Concerns and Useful Tips for Tracing” at http://msdn2.microsoft.com/en-us/library/ms733053.aspx

Authentication

Authentication is one of the most important pillars of security. Where possible, you should use Windows authentication because this enables you to use an existing identity store such as your organization’s Active Directory and to enforce strong password policies. You do not need to build custom identity store management tools, and passwords are not transmitted over the network.

This section provides guidance on choosing the correct authentication option for your scenario:

- **Know your authentication options.**
- **Use Windows authentication when you can.**
- **If your users are in Active Directory, but you can’t use Windows authentication, consider using username authentication.**

- **If you are using username authentication, use a membership provider instead of custom authentication.**
- **If your users are in a SQL Server membership store, use the SQL Server membership provider.**
- **If your users are in a custom store, consider using username authentication with a custom validator.**
- **If your clients have certificates, consider using client certificate authentication.**
- **If your partner applications need to be authenticated when calling WCF services, use client certificate authentication.**
- **If you are using username authentication, validate user login information.**
- **Do not store passwords directly in the user store.**
- **Enforce strong passwords.**
- **Protect access to your credential store.**
- **If you are using client certificate authentication, limit the certificates in the certificate store.**

Each of these guidelines is described in the following sections.

Know Your Authentication Options

It is important to understand the authentication options that map to your deployment scenario.

Internet

- **Username authentication with the SQL Server membership provider.** If your users are not in Active Directory, consider using the SQL Server membership provider. This will give you a store that can be easily created and deployed. Both message security and mixed-mode security support username authentication, and the authentication in WCF can be configured to use this authentication mode. Additionally, username authentication can cross firewall boundaries. Consider using transport security with a custom HTTP module that is redirected to authenticate with the SQL Server membership provider.
- **Basic authentication with Active Directory.** If your users are already in Active Directory or in local machine accounts, consider using username password with Basic authentication. Transport security supports this authentication mode. The service can be hosted either in Windows service or in IIS.
- **Username authentication with a custom store.** If your users are already in a custom store, or you need to use a custom store because other platform applications might have access to it, consider using username authentication with a custom validator. Both message security and mixed-mode security support this authentication mode. Additionally, username authentication can cross firewall boundaries.
- **Certificate authentication with Windows.** If your clients are partners, or mobile clients connecting over a virtual private network (VPN), consider using certificate authentication. This configures a peer-to-peer authentication scenario. Optionally, you can map the certificates to Windows accounts, for authorization checks based on Windows roles. Certificate authentication has the benefits of providing a secure

infrastructure and the ability to cross firewall boundaries. All security modes, depending on the binding used, support this authentication mode. The tradeoff is that using this authentication mode requires you to manage certificates. Also, a public key infrastructure (PKI) can be expensive to maintain.

Intranet

- **Username authentication with the SQL Server membership provider**. If your users are not in Active Directory, consider using the SQL Server membership provider. This will give you a store that can be easily created and deployed. Both message security and mixed-mode security support username authentication, and the authentication in WCF can be configured to use this authentication mode. Additionally, username authentication can cross firewall boundaries. Consider using transport security with a custom HTTP module that is redirected to authenticate with the SQL Server membership provider.
- **Windows authentication with Active Directory**. If your users are already in Active Directory, consider using Windows authentication to leverage the deployment and investment in the infrastructure. This authentication mode also gives the benefits of using Windows roles for authorization checks. Both transport security and message security support Windows authentication. Windows authentication also supports message security without requiring you to install certificates.
- **Username authentication with a custom store.** If your users are already in a custom store, or you need to use a custom store because other platform applications might have access to it, consider using username authentication with a custom validator. Both message security and mixed-mode support this authentication mode. Username authentication can also cross firewall boundaries.
- **Certificate authentication with Windows.** Consider using certificate authentication when you cannot use Windows authentication because of a firewall between the client and the service. Certificate authentication has the benefits of providing a secure infrastructure and the ability to cross firewall boundaries. All security modes, depending on the binding used, support this authentication mode. Optionally, you can map the certificates to Windows accounts for authorization checks based on Windows roles. The tradeoff is that using certificate authentication requires you to manage certificates. Also, a PKI can be expensive to maintain.

Additional Resources

- For more information on authentication, see “Authentication” at http://msdn2.microsoft.com/en-us/library/ms733082.aspx

Use Windows Authentication When You Can

Use Windows authentication in the following scenarios:

- When both the client and service are in trusted domains, such as in an Intranet scenario. By using Windows authentication with Active Directory, you benefit from a unified

identity store, centralized account administration, enforceable account and password policies, and strong authentication that avoids sending passwords over the network.

- When the service uses a local machine account, the client can authenticate by using the NTLM protocol. However, because NTLM is not secured, this can expose the service to man-in-the-middle attacks, where the hash password sent over the network can be compromised by brute-force attacks. Although Windows authentication can be used without Active Directory, you should consider using more secure methods such as certificate authentication.

If Your Users Are in Active Directory but You Can't Use Windows Authentication, Consider Using Username Authentication

If your users are in Active Directory and you cannot use Windows authentication (e.g., in an Internet scenario), consider using username authentication. Your users will be mapped to a Windows account by default. By using username authentication with Windows accounts, you benefit from having a unified identity store, centralized account administration, and enforceable account and password policies.

If You Are Using Username Authentication, use a Membership Provider Instead of Custom Authentication

If you are using username authentication in WCF and your users are not in Active Directory, use a membership provider, in this case the SQL Server membership provider. Do not try to implement your own user authentication mechanism.

The membership feature is a good choice because it allows you to enable username authentication without writing and maintaining custom code. The membership provider can be integrated into a WCF application in order to authenticate consumers of your service. Use a WCF binding such as **wsHttpBinding** that supports username/password credentials, and set the client credential type to **UserName**. Configure the membership provider in your configuration file to authenticate users against the membership store.

The following configuration snippet shows how to configure the username authentication with a membership provider:

1. Set the membership provider configuration:

```
<connectionStrings>
    <add name="MyLocalSQLServer"
         connectionString="Initial Catalog=aspnetdb;data
source=10.3.19.60;Integrated Security=SSPI;"/>
  </connectionStrings>

  <system.web>

    <membership defaultProvider="MySqlMembershipProvider" >
      <providers>
        <clear/>
        <add name="MySqlMembershipProvider"
```

```
            connectionStringName="MyLocalSQLServer"
            applicationName="MyAppName"
            type="System.Web.Security.SqlMembershipProvider" />
      </providers>
    </membership>
  </system.web>
```

2. Set the client credentials to **UserName**:

```
<wsHttpBinding>
 <binding name="BindingConfiguration">
  <security>
  <message clientCredentialType="UserName" />
   </security>
 </binding>
</wsHttpBinding>
```

3. Set the Service Credentials configuration to use the membership provider:

```
….
<serviceBehaviors>
  <behavior name="BehaviorConfiguration">
    ….
    <serviceCredentials>
      <userNameAuthentication
userNamePasswordValidationMode="MembershipProvider"
        membershipProviderName=" MySqlMembershipProvider " />
    </serviceCredentials>
  </behavior>
</serviceBehaviors>
….
```

If Your Users Are in a SQL Server Membership Store, Use the SQL Server Membership Provider

If your user information is already stored in the **aspnetdb** database, or if you are building an Internet-facing WCF application from scratch, use the SQL Server membership provider to authenticate your WCF service clients. The SQL Server membership provider authenticates all incoming client credentials against the credentials stored in the SQL Membership database. The membership feature is a good choice because it allows you to enable username authentication without writing and maintaining custom code. The SQL Server membership provider is configured in the Service config file. The following example illustrates a service's SQL Server membership provider configuration:

1. Set the SQL membership provider configuration:

```
…
<connectionStrings>
    <add name="MyLocalSQLServer"
         connectionString="Initial Catalog=aspnetdb;data
source=10.3.19.60;Integrated Security=SSPI;"/>
</connectionStrings>
<system.web>
…
```

```
      <membership defaultProvider="MySqlMembershipProvider" >
        <providers>
          <clear/>
          <add name="MySqlMembershipProvider"
               connectionStringName="MyLocalSQLServer"
               applicationName="MyAppName"
               type="System.Web.Security.SqlMembershipProvider" />
        </providers>
      </membership>
…
</system.web>
…
```

2. Set your binding to use username authentication as follows:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="BindingConfiguration">
      <security>
        <message clientCredentialType="UserName" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
…
```

3. Set the service credentials configuration to use the SQL Server membership provider with username authentication:

```
…
<serviceBehaviors>
  <behavior name="BehaviorConfiguration">
    <serviceCredentials>
      <userNameAuthentication
userNamePasswordValidationMode="MembershipProvider"
        membershipProviderName="MySqlMembershipProvider" />
    </serviceCredentials>
  </behavior>
</serviceBehaviors>
…
```

If Your Users Are in a Custom Store, Consider Using Username Authentication with a Custom Validator

If you need to use a custom authentication store, consider using username authentication with a custom username and password validator. Configure the custom validator in a service behavior and implement it in a class library. The username and password validator is used by your service to authenticate your users based on your custom user store.

The following configuration snippet shows how to configure a custom validator for your WCF service:

```
<serviceCredentials>
   <userNameAuthentication
    userNamePasswordValidationMode="Custom"
   customUserNamePasswordValidatorType="MyUserNamePasswordValidator,Host"/>
</serviceCredentials>
```

The following code snippet shows how to implement a custom username and password validator:

```
using System;
using System.Collections.Generic;
using System.IdentityModel.Selectors;
using System.IdentityModel.Tokens;
using System.Text;

namespace DerivativesCalculator
{
    public class MyUserNamePasswordValidator : UserNamePasswordValidator
    {
        public override void Validate(string userName, string password)
        {
            Console.Write("\nValidating username, {0}, and password, {1} ...
", userName, password);
            if ((string.Compare(userName, "don", true) != 0) ||
                (string.Compare(password, "hall", false) != 0))
            {
               throw new SecurityTokenException("Unknown user.");
            }
            Console.Write("Done: Credentials accepted. \n");
        }
    }
}
```

If Your Clients Have Certificates, Consider Using Client Certificate Authentication

Client certificates can authenticate a client service account or multiple users to a WCF service. If you use a client certificate for each user, you can map each certificate to a Windows account.

Enable the Windows account mapping feature by setting the **mapClientCertificateToWindowsAccount** attribute to true as follows:

```
<serviceCredentials>
    <clientCertificate>
        <authentication mapClientCertificateToWindowsAccount="true" />
    </clientCertificate>
</serviceCredentials>
```

If Your Partner Applications Need to Be Authenticated When Calling WCF Services, Use Client Certificate Authentication

If you have partners who need to consume your services, consider using transport security with client certificate authentication. This type of authentication allows clients to authenticate without prompting for a username and password.

In order to support client certificate authentication, you will need to add a <clientCertificate> reference to the client configuration file. The following client configuration example links the client certificate to the binding:

```
<behaviors>
  <endpointBehaviors>
    <behavior name="ClientCertificateBehavior">
      <clientCredentials>
        <clientCertificate findValue="client.com"
                           storeLocation="CurrentUser"
                           storeName="My"
                           x509FindType="FindBySubjectName" />
      </clientCredentials>
    </behavior>
  </endpointBehaviors>
</behaviors>
```

If You Are Using Username Authentication, Validate User Login Information

If you are using username authentication, validate the user-provided usernames and passwords for type, length, format, and range. Input and data validation represents one line of defense in the protection of your WCF application. Use regular expressions to constrain the input at the server.

If you are not using the SQL Server membership provider and need to develop your own queries to access your user store database, do not use login details to dynamically construct SQL statements because this makes your code susceptible to SQL injection. Instead, validate the input and then use parameterized stored procedures.

The following code snippet shows how to validate the credentials of a new user by using regular expressions.

```
using System;
using System.Text.RegularExpressions;

public void CheckNewUserCredentials(string name, string password)
{
    // Check name contains only lower case or upper case letters,
    // the apostrophe, a dot, or white space. Also check it is
    // between 1 and 40 characters long
    if ( !Regex.IsMatch(userIDTxt.Text, @"^[a-zA-Z'./s]{1,40}$"))
      throw new FormatException("Invalid name format");

    // Check password contains at least one digit, one lower case
    // letter, one uppercase letter, and is between 8 and 10
    // characters long
    if ( !Regex.IsMatch(passwordTxt.Text,
                        @"^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,10}$" ))
      throw new FormatException("Invalid password format");

    // Perform data access logic (using type safe parameters)
```

```
    ...
}
```

Do Not Store Passwords Directly in the User Store

Do not store user passwords in plaintext or in encrypted format. Instead, store password hashes with a salt. By storing your password with hashes and salt, you help prevent an attacker that gains access to your user store from obtaining the user passwords. If you use encryption, you have the added problem of securing the encryption key. Use one of the membership providers to help protect credentials in storage and, where possible, specify a hashed password format on your provider configuration.

If you must implement your own user stores, store one-way password hashes with salt. Generate the hash from a combination of the password and a random salt value. Use an algorithm such as SHA256. If your credential store is compromised, the salt value helps to slow an attacker who is attempting to perform a dictionary attack or rainbow table. This gives you additional time to detect and react to the compromise.

Enforce Strong Passwords

Ensure that your passwords are complex enough to protect against brute-force or dictionary attacks against your user credential store.

When using username authentication with the ASP.NET membership provider, users are forced to use strong passwords by default. For example, the SQL Server membership provider and the Active Directory membership provider ensure that passwords are at least seven characters in length, with at least one non-alphanumeric character. Ensure that your membership provider configuration enforces passwords of at least this strength.

To configure the precise password complexity rules enforced by your provider, you can set the following additional attributes:

- **passwordStrengthRegularExpression**. The default is "".
- **minRequiredPasswordLength**. The default is 7.
- **minRequiredNonalphanumericCharacters**. The default is 1.

Note: The default values shown here apply to the SQL Server membership provider and the Active Directory membership provider. The Active Directory membership provider also verifies passwords against the default domain password policy.

Protect Access to Your Credential Store

Ensure that only those accounts that require access are granted access to your credential store. This helps to protect the credential store by limiting access to it. For example, consider limiting access to only your application's account. Ensure that the connection string used to identify your credential store is encrypted.

Also consider storing your credential database on a server that is physically separate from your WCF application server. This makes it more difficult for an attacker to compromise your credential store even if he or she manages to take control of your server.

If You Are Using Client Certificate Authentication, Limit the Certificates in the Certificate Store

If you are using certificate authentication, consider reducing the attack surface by limiting the certificates in the certificates store. Keep in mind the following considerations:

- Consider deleting all the root certificates from the **trusted root certification authorities** store that are not required in order to authenticate your clients.
- If your client base is large, consider using chain trust validation instead of peer trust so that you will have a smaller number of certificates to manage.
- If your client base is small, consider using peer trust validation authentication. This will require that you manage one certificate per user. Any users not installed in the **trusted people** store will be denied access to the service.

Authorization

Authorizing users of your WCF applications helps in reducing the attack surface. You control authorization by using either resource-based or role-based authorization.

Use the following guidelines to choose an authorization strategy when implementing role-base authorization:

- **If you store role information in Windows groups, consider using the WCF PrincipalPermissionAttribute class for role authorization.**
- **If you use ASP.NET roles, use the Role Manager for role authorization.**
- **If you use Windows groups for authorization, use the ASP.NET Role Provider with AspNetWindowsTokenRoleProvider.**
- **If you store role information in SQL Server, consider using the SQL Server role provider for role authorization.**
- **If you store role information in ADAM, use the Authorization Manager role provider.**
- **If you store role information in a custom store, create a custom authorization policy.**
- **If you need to authorize access to WCF operations, use declarative authorization.**
- **If you need to perform fine-grained authorization based on business logic, use imperative authorization.**

Each of these guidelines is described in the following sections.

If You Store Role Information in Windows Groups, Consider Using the WCF PrincipalPermissionAttribute Class for Role Authorization

If you are using Windows groups to store user roles, map your Windows groups to the WCF service methods by using the **PrincipalPermission** attribute. Incoming client username credentials will be mapped to associated Windows groups. Service method access will be

granted to the user if the user is a member of the group associated with the method being called.

The following example demonstrates how the WCF service's **Add** method will only run for users belonging to the CalculatorClients Windows group:

```
// Only members of the CalculatorClients group can call this method.
[PrincipalPermission(SecurityAction.Demand, Role = "CalculatorClients")]
public double Add(double a, double b)
{
    return a + b;
}
```

Additional Resources

- For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Use ASP.NET Roles, Use the ASP.NET Role Manager for Role Authorization

If you are using ASP.NET roles, leverage these roles in your WCF service by using the ASP.NET role manager. The role information can be stored in SQL Server, Windows Groups, or an Authorization Manager (AzMan) policy store in ADAM.

The ASP.NET role manager supports role creation and role managemen, and automatically performs role lookup for authenticated users.

Additional Resources

- For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Use Windows Groups for Authorization, Use the ASP.NET Role Provider with AspNetWindowsTokenRoleProvider

If you use Windows groups for authorization, consider using the ASP.NET role provider with the **AspNetWindowsTokenRoleProvider** name. This allows you to separate the design of the authorization from the implementation inside your service. If you decide to change the role provider, it will not affect the code needed to perform the authorization. Also consider doing

imperative checks by using the role manager API instead of performing authorization checks with **WindowsPrincipal.IsInrole**.

The following configuration example shows how to configure **AspNetWindowsTokenRoleProvider**:

```
<system.web>
…
<roleManager enabled="true"
             defaultProvider="AspNetWindowsTokenRoleProvider" />
…
</system.web>

Configure the service behavior to use ASPNetRoles and the role provider.
….
<behaviors>
    <serviceBehaviors>
        <behavior name="BehaviorConfiguration">
            <serviceAuthorization principalPermissionMode="UseAspNetRoles"
                roleProviderName=" AspNetWindowsTokenRoleProvider " />
            <serviceMetadata />
        </behavior>
    </serviceBehaviors>
</behaviors>
….
```

The following code example shows how to perform the authorization check in code, using the role manager API:

```
if (Roles.IsUserInRole(@"accounting"))
{
   //authorized
}
else
{
   //authorization failed
}
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For information on how to use the role manager , see “How To: Use Role Manager in ASP.NET 2.0”at http://msdn2.microsoft.com/en-us/library/ms998314.aspx
- For an authorization Q&A, see the Authorization section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Store Role Information in SQL Server, Consider Using the SQL Server Role Provider for Role Authorization

If you store role information in SQL Server, configure your application to use the SQL Server role provider for authorization. The role provider allows you to load the roles for users without writing and maintaining custom code.

The following steps show how to use SQL Server role provider to provide role-based authorization:

1. Enable the role provider as shown below and configure the connection string pointing to the role store in SQL Server:

```
…
<configuration>
…
<connectionStrings>
    <add name="MyLocalSQLServer"
         connectionString="Initial Catalog=aspnetdb;data
source=Sqlserver;Integrated Security=SSPI;"

<system.web>
<roleManager enabled="true" defaultProvider="MySqlRoleProvider" >
      <providers>
        <add name="MySqlRoleProvider"
             connectionStringName="MyLocalSQLServer"
             applicationName="MyAppName"
             type="System.Web.Security.SqlRoleProvider" />
      </providers>
    </roleManager>
…
<system.web>
```

2. Configure the service behavior. Set the **principalPermissionMode** attribute to **UseAspNetRoles** and the **roleProviderName** attribute to **MySqlRoleProvider**.

```
…
<system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="BehaviorConfiguration">
          <serviceAuthorization
principalPermissionMode="UseAspNetRoles"
            roleProviderName="MySqlRoleProvider" />
         </behavior>
      </serviceBehaviors>
    </behaviors>
<services>
    <service behaviorConfiguration=" BehaviorConfiguration "
name="MyService">
      <endpoint binding="wsHttpBinding" bindingConfiguration=""
        name="httpsendpoint" contract="IMyService2" />
     </service>
  </services>
  </system.serviceModel>
…
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Store Role Information in ADAM, Use the Authorization Manager Role Provider

If your application stores role information in an Authorization Manager (AzMan) policy store in Active Directory Application Mode (ADAM), use the Authorization Manager role provider. Authorization Manager provides a Microsoft Management Console (MMC) snap-in for creating and managing roles and managing role membership for users.

Additional Resources

- For more information on the Authorization Manager, see “How To: Use Authorization Manager (AzMan) with ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998336.aspx
- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Store Role Information in a Custom Store, Create a Custom Authorization Policy

If your application stores authorization data in a custom store such as a SQL Server database, create a custom authorization policy to authorize your users.

To create a custom authorization policy, you implement a class derived from **IAuthorizationPolicy** along with an **Evaluate** method that you can customize for your user authorization policy.

The Policy library is configured in the configuration file or in code. The following example configures the policy location in the configuration file:

```
<serviceAuthorization
serviceAuthorizationManagerType="Microsoft.ServiceModel.Samples.MyServiceAuth
orizationManager, service">
<!-- The serviceAuthorization behavior allows one to specify custom
authorization policies. -->
<authorizationPolicies>
```

```
      <add
policyType="Microsoft.ServiceModel.Samples.CustomAuthorizationPolicy.MyAuthor
izationPolicy, PolicyLibrary" />
</authorizationPolicies>
</serviceAuthorization>
```

Additional Resources

- For more information on custom authorization policies, see "How to: Create a Custom Authorization Policy" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx

If You Need to Authorize Access to WCF Operations, Use Declarative Authorization

You can add declarative authorization by specifying required access for a particular method declared as an attribute on the operation. Declarative authorization checks will work if you are using the ASP.NET role provider or Windows groups. Use imperative authorization if you need finer-grained authorization decisions based on business logic.

The following code example shows how to use the **PrinciplePermission** attribute to perform declarative authorization:

```
[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]
public double Add(double a, double b)
{
    return a + b;
}
```

Additional Resources

- For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Perform Fine-Grained Authorization Based on Business Logic, Use Imperative Authorization

Use imperative role-based authorization when you need to make fine-grained authorization choices based on business logic, or when you require finer-grained access control beyond the level of a code method.

Imperative check using a Windows principal:

```
WindowsPrincipal myPrincipal = new
WindowsPrincipal(ServiceSecurityContext.Current.WindowsIdentity);
if(myPrincipal.IsInRole(@"domain\Accounting"))
```

```
{
//authorized
}
else
{
//not authorized
}
```

Imperative check using the ASP.NET role provider:

```
if (Roles.IsUserInRole(@"accounting"))
{
//authorized
}
else
{
//authorization failed
}
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Bindings

Choosing the right binding for your particular scenario is important from both a security and performance perspective. One general rule of thumb you can follow is o use **netTcpbinding** in an intranet scenario and **wsHttpBinding** in an Internet scenario. You can then fine-tune your selection based on your unique needs and your infrastructure limitations.

Consider the following recommendations when choosing a binding option:

- **If you need to support clients over the Internet, consider using wsHttpBinding.**
- **If you need to expose your WCF service to legacy clients as an ASMX Web service, use basicHttpBinding.**
- **If you need to support WCF clients within an intranet, consider using netTcpBinding.**
- **If you need to support WCF clients on the same machine, consider using netNamedPipeBinding.**
- **If you need to support disconnected queued calls, use netMsmqBinding.**
- **If you need to support bidirectional communication between a WCF client and WCF service, use wsDualHttpBinding or netTcpBinding.**

Each of these guidelines is described in the following sections.

If You Need to Support Clients Over the Internet, Consider Using wsHttpBinding

If your service will be called by clients over the Internet, consider using **wsHttpBinding**. **wsHttpBinding** is a good choice for Internet scenarios in which you do not have to support legacy clients that expect an ASMX Web service. If you do need to support legacy clients, consider using **basicHttpBinding** instead.

wsHttpBinding has the following characteristics:

- It provides interoperability with non-WCF clients that support the WS* stack.
- It supports the WS* stack, including reliable messaging, message security, and secure transactions.
- Message security is turned on by default. Transport security is also available.
- It allows the service to be hosted in IIS 5.0 or IIS 6.0.
- If you choose transport security, you can use certificate, Windows, or token authentication.
- If you choose message security, you can use certificate, username, Windows, or issue token authentication (Windows CardSpace).

The following example shows how to configure wsHttpBinding :

```
<system.serviceModel>
....
    <services>
        <service behaviorConfiguration="" name="WCFServicecHost.MyService">
            ...
            <endpoint address="" binding="wsHttpBinding"
bindingConfiguration=""
                name="wsBinding" contract="WCFServicecHost.IMyService" />
            ...
        </service>
    </services>
</system.serviceModel>
```

Additional Resources

- For more information on bindings, see “Windows Communication Foundation Bindings” at http://msdn2.microsoft.com/en-us/library/ms733027.aspx
- For a binding Q&A, see the Bindings section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Expose Your WCF Service to Legacy Clients as an ASMX Web Service, Use basicHttpBinding

If your service needs to support legacy clients that expect an ASMX Web service, consider using **basicHttpBinding**. **basicHttpBinding** does not implement any security by default. If you require

message or transport security, configure it explicitly using this binding. Use **basicHttpBinding** to expose endpoints that are able to communicate with ASMX-based Web services and clients as well as other services that conform to the WS-I Basic Profile 1.1 specification. When configuring transport security, **basicHttpBinding** defaults to no credentials just like a classic ASMX Web service.

basicHttpBinding has the following characteristics:

- Because it does not support the WS* stack, it does not provide reliable messaging and secure transactions.
- Neither transport nor message security is turned on by default.
- It allows interoperability with legacy clients that expect to consume an ASMX Web service.
- It allows the service to be hosted in IIS 5.0 or IIS 6.0.
- If you choose to use message security, you can only use Basic or certificate authentication.
- If you choose to use transport security, you can use certificate, Windows, or issue token authentication (Windows CardSpace).

The following example shows how to configure **basicHttpBinding**:

```
<system.serviceModel>
....
    <services>
        <service behaviorConfiguration="" name="WCFServicecHost.MyService">
            ...
            <endpoint address="" binding="basicHttpBinding"
bindingConfiguration=""
                name="basicBinding" contract="WCFServicecHost.IMyService" />
            ...
        </service>
    </services>
</system.serviceModel>
```

Additional Resources

- For more information on bindings, see “Windows Communication Foundation Bindings” at http://msdn2.microsoft.com/en-us/library/ms733027.aspx
- For a binding Q&A, see the Bindings section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Support WCF Clients Within an Intranet, Consider Using netTcpBinding

If you need to support clients within your intranet, consider using **netTcpBinding**. **netTcpBinding** is a good choice for the intranet scenario if transport performance is important and it is acceptable to host the service as a Windows service instead of in IIS. **netTcpBinding** uses the

Transmission Control Protocol (TCP) and provides full support for Simple Object Access Protocol (SOAP) security, transactions, and reliability. Use this binding when you want to provide a secure and reliable binding environment for .NET-to-.NET cross-machine communication.

netTcpBinding has the following characteristics:

- It can only be consumed by WCF-enabled clients.
- It supports the WS* stack, including reliable messaging, message security, and secure transactions.
- Transport security is turned on by default. Message security is also available.
- The service can be hosted in IIS 5.0 or IIS 6.0, but as it is not message-activated, you can consider hosting in a Windows service or IIS 7.0 instead.
- If you choose to use message security, you can use certificate, username, Windows, or issue token authentication (Windows CardSpace).
- If you choose to use transport security, you can only use certificate or Windows authentication.

The following example shows how to configure **netTcpBinding**:

```
<system.serviceModel>
....
    <services>
        <service behaviorConfiguration="" name="WCFServicecHost.MyService">
            ...
            <endpoint address="" binding="netTcpBinding"
bindingConfiguration=""
                name="TcpBinding" contract="WCFServicecHost.IMyService" />
            ...
        </service>
    </services>
</system.serviceModel>
```

Additional Resources

- For more information on bindings, see “Windows Communication Foundation Bindings” at http://msdn2.microsoft.com/en-us/library/ms733027.aspx
- For a binding Q&A, see the Bindings section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Support WCF Clients on the Same Machine, Consider Using netNamedPipeBinding

If you need to support WCF clients on the same machine as your service, consider using **netNamedPipeBinding**. Use this binding when you want to make use of the NamedPipe protocol and provide full support for SOAP security, transactions, and reliability.

netNamedPipeBinding has the following characteristics:

- It can only be consumed by WCF-enabled clients.
- It supports the WS* stack, including reliable messaging and secure transactions.
- It only supports transport security; you cannot turn on message security.
- The service can be hosted in IIS 5.0 or IIS 6.0, but as it is not message activated, you can consider hosting in a Windows service or IIS 7.0 instead.
- Your only authentication option is Windows.

The following example shows how to configure **netNamedPipeBinding**:

```
<system.serviceModel>
....
    <services>
        <service behaviorConfiguration="" name="WCFServicecHost.MyService">
            ...
            <endpoint address="" binding="netNamedPipeBinding"
bindingConfiguration=""
                name="namedPipeBinding" contract="WCFServicecHost.IMyService"
/>
            ...
        </service>
    </services>
</system.serviceModel>
```

Additional Resources

- For more information on bindings, see “Windows Communication Foundation Bindings” at http://msdn2.microsoft.com/en-us/library/ms733027.aspx
- For a binding Q&A, see the Bindings section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Support Disconnected Queued Calls, Use netMsmqBinding

If you need to support disconnected queuing, use **netMsmqBinding**. Queuing is provided by using Microsoft Message Queuing (MSMQ) as a transport, which enables support for disconnected operations, failure isolation, and load leveling. You can use **netMsmqBinding** when the client and service do not have to be online at the same time. You can also manage any number of incoming messages by using load leveling. MSMQ supports failure isolation, where messages can fail without affecting the processing of other messages.

netMsmqBinding has the following characteristics:

- It supports asynchronous, disconnected operations.
- It can only be consumed by WCF-enabled clients.
- Transport security is turned on by default. Message security is also available.
- The service can be hosted in IIS 5.0 or IIS 6.0, but as it is not message activated, you can consider hosting in a Windows service or IIS 7.0 instead.

- If you choose to use message security, you can use certificate, username, Windows, or issued token authentication
- If you choose to use transport security, you can only use certificate or Windows authentication.

The following example shows how to configure **netMsmqBinding**:

```
<system.serviceModel>
....
    <services>
        <service behaviorConfiguration="" name="WCFServicecHost.MyService">
            ...
            <endpoint address="" binding="netMsmqBinding"
bindingConfiguration=""
                name="MsmqBinding" contract="WCFServicecHost.IMyService" />
            ...
        </service>
    </services>
</system.serviceModel>
```

Additional Resources

- For more information on bindings, see "Windows Communication Foundation Bindings" at http://msdn2.microsoft.com/en-us/library/ms733027.aspx
- For a binding Q&A, see the Bindings section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Support Bidirectional Communication Between a WCF Client and WCF Service, Use wsDualHttpBinding or netTcpBinding

If you need to support a duplex service, use **wsDualHttpBinding** or **netTcpBinding**. A *duplex service* is a service that uses duplex message patterns, which provides the ability for a service to communicate back to the client via a callback. You can also use **wsDualHttpBinding** binding to support communication via SOAP intermediaries.

wsDualHttpBinding has the following characteristics:

- It supports two-way communication between the client and the service.
- It provides interoperability with non-WCF clients that support the WS* stack.
- It supports the WS* stack, including reliable messaging and secure transactions.
- It only supports message security; you cannot turn on transport security.
- It allows the service to be hosted in IIS 5.0 or IIS 6.0.
- If you choose to use message security, you can use certificate, username, Windows, or Issue token authentication (Windows CardSpace).

The following example shows how to configure **wsDualHttpBinding**:

```
<system.serviceModel>
```

```
....
    <services>
        <service behaviorConfiguration="" name="WCFServicecHost.MyService">
            ...
            <endpoint address="" binding="wsDualHttpBinding"
bindingConfiguration=""
                name="DualBinding" contract="WCFServicecHost.IMyService" />
            ...
        </service>
    </services>
</system.serviceModel>
```

Additional Resources

- For more information on bindings, see "Windows Communication Foundation Bindings" at http://msdn2.microsoft.com/en-us/library/ms733027.aspx
- For a binding Q&A, see the Bindings section of "WCF 3.5 Security Questions and Answers " at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Configuration Management

Follow these guidelines to avoid introducing vulnerabilities when you configure your WCF application:

- **Use replay detection to protect against message replay attacks.**
- **If you host your service in a Windows service, expose a metadata exchange (mex) binding.**
- **If you don't want to expose your WSDL, turn off HttpGetEnabled and metadata exchange (mex).**
- **Encrypt configuration sections that contain sensitive data.**

Each of these guidelines is described in the following sections.

Use Replay Detection to Protect Against Message Replay Attacks

Use the WCF replay detection feature to protect your service against message replay attacks. A message replay attack occurs when an attacker copies a stream of messages between two parties and replays the stream to one or more of the parties. Unless mitigated, the computers subject to the attack will process the stream as legitimate messages, resulting in a range of harmful consequences including unauthorized access to the service.

To enable replay detection in your service:

1. Create a customBinding Element.
2. Create a <security> element.
3. Create a localClientSettings element or localServiceSettings element.

4. Set the following attribute values, as appropriate: **detectReplays**, **maxClockSkew**, **replayWindow**, and **replayCacheSize**. The following example sets the attributes of both a <localServiceSettings> and a <localClientSettings> element:

```
<customBinding>
  <binding name="NewBinding0">
   <textMessageEncoding />
    <security>
     <localClientSettings
      replayCacheSize="800000"
      maxClockSkew="00:03:00"
      replayWindow="00:03:00" />
     <localServiceSettings
      replayCacheSize="800000"
      maxClockSkew="00:03:00"
      replayWindow="00:03:00" />
    <secureConversationBootstrap />
   </security>
  <httpTransport />
 </binding>
</customBinding>
```

Additional Resources

- For more information on replay detection, see "How to: Enable Message Replay Detection" at http://msdn2.microsoft.com/en-us/library/ms733063.aspx
- For a configuration management Q&A, see the Configuration Management section of "WCF 3.5 Security Questions and Answers " at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Host Your Service in a Windows Service, Expose a Metadata Exchange (mex) Binding

If you are hosting your service as a Windows service and are exposing the service by using **netTcpBinding**, publish the service metadata by creating a **mexTcpBinding** endpoint so that your clients can discover and use the service. Clients will be able to generate a proxy file by using the ServiceModel Metadata Utility Tool (Svcutil.exe).

Additional Resources

- For more information on publishing metadata endpoints, see "Publishing Metadata" at http://msdn2.microsoft.com/en-us/library/aa751951.aspx
- For a configuration management Q&A, see the Configuration Management section of "WCF 3.5 Security Questions and Answers " at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Don't Want to Expose Your WSDL, Turn Off HttpGetEnabled and Remove Metadata Exchange (mex) Endpoints

If you want to block clients from accessing your service's Web Services Description Language (WSDL), you should remove all metadata exchange endpoints and set the **httpGetEnabled** and **httpsGetEnabled** attributes to false.

This is potentially important after your clients are built and deployed, if you do not want other clients to discover and use the WCF service. If the metadata is exposed, unwanted clients will be able to generate proxy files (e.g., by using Svcutil.exe) and inspect potentially sensitive methods and parameters offered by the service. If your client programs already have access to the service proxy, set the **httpGetEnabled** attribute to false.

The following configuration disables sharing service metadata:

```
<serviceMetadata httpGetEnabled="False" httpsGetEnabled="False"/>
```

Additional Resources

- For more information on publishing metadata endpoints, see "Publishing Metadata" at http://msdn2.microsoft.com/en-us/library/aa751951.aspx
- For a configuration management Q&A, see the Configuration Management section of "WCF 3.5 Security Questions and Answers " at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Encrypt Configuration Sections That Contain Sensitive Data

Encrypt configuration sections that contain sensitive data such as SQL connection strings. Use the Data Protection API (DPAPI) to encrypt the sensitive data in the configuration file on your WCF server machine.

To encrypt the <connectionStrings> section by using the DPAPI provider with the machine-key store (the default configuration), run the following command from a command window:

```
aspnet_regiis -pe "connectionStrings" -app "/MachineDPAPI" -prov
"DataProtectionConfigurationProvider"
```

The aspnet_regiis options are:

- **-pe** – Specifies the configuration section to encrypt.
- **-app** – Specifies your Web application's virtual path. If your application is nested, you need to specify the nested path from the root directory; for example, "/test/aspnet/MachineDPAPI".
- **-prov** – Specifies the provider name.

Note: If you need to encrypt configuration file data on multiple servers in a Web farm, use the RSA protected configuration provider because of the ease with which you can export RSA key containers.

Additional Resources

- For more information on using DPAPI, see "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI" at http://msdn2.microsoft.com/en-us/library/ms998280.aspx
- For more information on using RSA, see "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA" at http://msdn2.microsoft.com/en-us/library/ms998283.aspx
- For a configuration management Q&A, see the Configuration Management section of "WCF 3.5 Security Questions and Answers " at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Exception Management

Correct exception handling in your WCF application prevents sensitive exception details from being revealed to the user, improves application robustness, and helps avoid leaving your application in an inconsistent state in the event of errors.

Consider the following guidelines:

- **Use structured exception handling.**
- **Do not divulge exception details to clients in production.**
- **Use a fault contract to return error information to clients.**
- **Use a global exception handler to catch unhandled exceptions.**

Each of these guidelines is described in the following sections.

Use Structured Exception Handling

Use structured exception handling and catch exception conditions. Doing this improves robustness and avoids leaving your application in an inconsistent state that may lead to information disclosure. It also helps protect your application from denial of service (DoS) attacks.

In C#, you can use the **try/catch** and **finally** construct to implement structured exception handling. You can protect code by placing it inside **try** blocks, and implement **catch** blocks to log and process exceptions. Also, use the **finally** construct to ensure that critical system resources such as connections are closed, whether an exception condition occurs or not.

Additional Resources

- For more information about exceptions, see the Exceptions Reference at http://msdn2.microsoft.com/en-us/library/ms733763.aspx
- For an exception management Q&A, see the Exception Management section of "WCF 3.5 Security Questions and Answers " at

http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Do Not Divulge Exception Details to Clients in Production

Do not divulge exception error details to clients in production. Instead, develop a fault contract and return it to your client. When exceptions occur, return concise error messages to the client and log specific details on the server. Do not reveal internal system or application details — such as stack traces, SQL statement fragments, and table or database names — to the client. Ensure that this type of information is not allowed to propagate to the end user or beyond your current trust boundary. If you expose any detailed exception information to the client, a malicious user could use the system-level diagnostic information to learn about your application and probe for weaknesses to exploit in future attacks.

By using the **FaultContract** attribute in a service contract you can specify the possible faults that can occur in your WCF service. This prevents exposing any other exception details to the clients. To define a Fault contract, apply the FaultContract attribute directly on a contract operation, specifying the error detailing type as shown below:

```
[ServiceContract]
interface ICalculator
{
   [OperationContract]
   [FaultContract(typeof(DivideByZeroException))]
   double Divide(double number1,double number2);
}
```

In the following example, the **FaultContract** attribute is limited to the **Divide** method. Only that method can throw that fault and have it propagated to the client.

```
class MyService : ICalculator
{
   public double Divide(double number1,double number2)
   {

      throw new FaultException<DivideByZeroException>(new
DivideByZeroException());
   }
}
```

Additional Resources

- For more information about exceptions, see the Exceptions Reference at http://msdn2.microsoft.com/en-us/library/ms733763.aspx
- For an exception management Q&A, see the Exception Management section of “WCF 3.5 Security Questions and Answers “ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Use a Fault Contract to Return Error Information to Clients

Use a fault contract to define the exceptions in your service and return error information to clients. By using the **FaultContract** attribute in a service contract, you can specify the possible faults that can occur in your WCF service. If there is an exception in your WCF service operation, use the **FaultContract** attribute to generate a specific SOAP fault message that will be sent back to the client application. The **FaultContract** attribute can only be used in operations that return a response. You cannot use this attribute on a one-way operation.

The following code snippet shows how to use the **FaultContract** attribute to return error information.

1. Define the type to pass the details of SOAP faults as exceptions from a service back to a client.

```
[DataContract]
public class DatabaseFault
{
    [DataMember]
    public string DbOperation;
    [DataMember]
    public string DbReason
    [DataMember]
    public string DbMessage;
}
```

2. Use the **FaultContract** attribute in the **ListCustomers** method to generate SOAP faults.

```
[ServiceContract]
public interface ICustomerService
{
    // Get the list of customers
    [FaultContract(typeof(DatabaseFault))]
    [OperationContract]
    List<string> ListCustomers();
    …
}
```

3. Create and populate the **DatabaseFault** object with the details of the exception in the **Service implementation** class and then throw a **FaultException** object with the **DatabaseFault** object details.

```
catch(Exception e)
{    DatabaseFault df = new DatabaseFault();
    df.DbOperation = "ExecuteReader";
    df.DbReason = "Exception in querying the Northwind database.";
    df.DbMessage = e.Message;
    throw new FaultException<DatabaseFault>(df);
}
```

Use a Global Exception Handler to Catch Unhandled Exceptions

Use a global exception handler to catch unhandled exceptions and prevent them from being propagated to the client. You can handle the unhandled exceptions in a WCF service by subscribing to the **Faulted** event of a service host object. By subscribing to this event, you can determine the cause of a failure, and then perform the necessary actions to abort or restart the service.

The following code snippet shows how to subscribe to the **Faulted** event:

```
// hosting a WCF service
ServiceHost customerServiceHost;
customerServiceHost = new ServiceHost(…);
…
// Subscribe to the Faulted event of the customerServiceHost object
customerServiceHost.Faulted += new EventHandler(faultHandler);
…
// FaultHandler method - invoked when customerServiceHost enters the Faulted
state
void faultHandler(object sender, EventArgs e)
{     // log the reasons for the fault…
}
```

Additional Resources

- For more information about exceptions, see the Exceptions Reference at http://msdn2.microsoft.com/en-us/library/ms733763.aspx
- For an exception management Q&A, see the Exception Management section of "WCF 3.5 Security Questions and Answers " at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Hosting

Choosing the correct host and process identity to run your WCF application is very important from a security perspective.

Consider the following guidelines when choosing a host for your WCF application:

- **Run your service in a least-privileged account.**
- **Use IIS to host your service unless you need to use a transport that IIS does not support.**

Each of these guidelines is described in the following sections.

Run Your Service in a Least-Privileged Account

If you are hosting your WCF service in a Windows service or in IIS, run your service using a least-privileged service account. By default, IIS is run under the ASPNET (in IIS 5.0) or NetworkService account (in IIS 6.0).

By using a custom domain account, you can audit and authorize your service individually, and your service is protected from changes made to the privileges and permissions within the System account. Configure your account to use the least number of privileges necessary to allow your service to run. This will reduce the attack surface and constrain the impact of any malicious attack.

The following steps outline how to use a least-privileged custom domain account:

1. Create a Windows account.
2. Run the following aspnet_regiis.exe command to assign the relevant ASP.NET permissions to the account:

   ```
   aspnet_regiis.exe -ga machineName\userName
   ```

   Note: This step is needed only if your application needs to run in ASP.NET compatibility mode.
3. Use the Local Security Policy tool to grant the Windows account the **Deny logon locally** user right.
   This reduces the privileges of the account and prevents anyone from logging on to Windows locally with this account.
4. If your service is hosted in a Windows service, configure the Windows service to run using the account identity.
   The WCF service will run under the security context of the Windows service.
5. If your service is hosted in IIS 6.0, use IIS Manager to create an application pool running as an account identity. Use IIS Manager to assign your WCF service to that application pool.

Additional Resources

- For more information, see “Hosting” at http://msdn2.microsoft.com/en-us/library/ms729846.aspx
- For more information on running IIS under a least-privileged service account, see “How To: Create a Service Account for an ASP.NET 2.0 Application” at http://msdn2.microsoft.com/en-us/library/ms998297.aspx
- For a hosting Q&A, see the Hosting section of “WCF 3.5 Security Questions and Answers “ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Use IIS to Host Your Service Unless You Need to Use a Transport That IIS Does Not Support

Use IIS to host your WCF service because it provides a large number of features for efficient service management and scalability. IIS 6.0 only supports bindings over HTTP so if you need to use TCP, Microsoft Message Queuing (MSMQ), or named pipes, you should host in a Windows service instead. IIS 7.0 supports all of the commonly used transport protocols such as HTTP, TCP, MSMQ, and named pipes.

By using IIS as your WCF service host, you can take full advantage of IIS features, such as process recycling, idle shutdown, process health monitoring, and message-based activation.

Additional Resources

- For more information, see "Hosting" at http://msdn2.microsoft.com/en-us/library/ms729846.aspx
- For a hosting Q&A, see the Hosting section of "WCF 3.5 Security Questions and Answers " at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Impersonation/Delegation

By default, WCF applications do not impersonate, although in some scenarios you need to use impersonation to perform operations or to access resources using the authenticated user's identity.

If you use impersonation, consider the following guidelines:

- **Know the tradeoffs involved in impersonation.**
- **Know your impersonation options.**
- **Know your impersonation methods.**
- **Consider using programmatic instead of declarative impersonation.**
- **When impersonating programmatically, be sure to revert to the original context.**
- **When impersonating declaratively, only impersonate on the operations that require it.**
- **Consider using the S4U feature for impersonation and delegation when you cannot do a Windows mapping.**
- **Consider using the LogonUser API if your WCF service cannot be trusted for delegation.**
- **Use constrained delegation if you have to flow the original caller to the back-end services.**

Each of these guidelines is described in the following sections:

Know the Tradeoffs Involved in Impersonation

Be aware that impersonation prevents the efficient use of connection pooling if you access downstream databases by using the impersonated identity. This impacts the ability of your application to scale. Also, using impersonation can introduce other security vulnerabilities, particularly in multi-threaded applications.

You might need impersonation if you need to:

- Flow the original caller's security context to the middle tier and/or data tier of your Web application in order to support fine-grained (per-user) authorization.

- Flow the original caller's security context to the downstream tiers in order to support operating system–level auditing.
- Access a particular network resource by using a specific identity.

Know Your Impersonation Options

Impersonation is used to restrict or authorize the original caller's access to a WCF service's local resources, such as files. There are three options available for impersonation:

- Impersonate using Windows authentication.
- Impersonate using S4U Kerberos extensions.
- Impersonate using the LogonUser API.

Each of these options is described in the following sections.

Impersonate Using Windows Authentication

With this option, you impersonate by using the Windows token, which is obtained from the Security Support Provider Interface (SSPI) or Kerberos authentication, or any other authentication type that cam map to Windows, such as username or certificate authentication. The Windows identity token obtained by this method is then cached on the service.

This impersonation option supports programmatic and declarative impersonation in WCF.

Impersonate Using S4U Kerberos Extensions

With this option, you impersonate by using a Windows token obtained from the Kerberos extensions, collectively called Service-for-User (S4U). You can use this option when your clients are authenticated using non-Windows authentication types such as client certificates but have mapping to Windows accounts, or when you want to impersonate a service account. This impersonation option supports programmatic impersonation in WCF.

Note: To impersonate at the impersonation level, you must grant your process account the "Act as part of the operating system" user right.

Impersonate Using the LogonUser API

With this option, you impersonate by using a Windows token obtained from the **LogonUser** Windows API. You can use this option when you want to access network resources (delegation) but do not have trust for delegation, or if you want to access local resources but do not want to grant higher privileges to your WCF process identity. This option adds the responsibility of maintaining the user credentials on the WCF service. This impersonation option supports programmatic impersonation in WCF.

Additional Resources

- For more information on the S4U extensions and protocol transition, see "Using Protocol Transition—Tips from the Trenches" by Keith Brown at http://msdn2.microsoft.com/en-us/magazine/cc163500.aspx

- For more information, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

Know Your Impersonation Methods

Impersonation is used to restrict or authorize the original caller’s access to a WCF service’s local resources, such as files. There are three methods of impersonation:

- Impersonate the original caller declaratively on specific operations.
- Impersonate the original caller declaratively on the entire service.
- Impersonate the original caller programmatically within an operation.

Impersonate the Original Caller Declaratively on Specific Operations

Use this option when you want to impersonate the original caller for the entire duration of a specific operation. Impersonation is a costly operation and is usually used for higher-privileged original callers, hence you would use impersonation selectively only on the operations that need a reduced potential attack surface. You can impersonate declaratively by applying the **OperationBehaviorAttribute** attribute on any operation that requires client impersonation, as shown in the following code example:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
   return “test”;
}
```

Impersonate the Original Caller Declaratively on the Entire Service

Use this option when you want to impersonate the original caller for the entire duration of all the operations. Impersonation is a costly operation and is usually used for higher-privileged original callers. You need to be careful when using this option because it potentially increases the attack surface. To impersonate the entire service, set the **impersonateCallerForAllOperations** attribute to "true" in the WCF configuration file, as shown in the following example:

```
...
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceAuthorization impersonateCallerForAllOperations="true" />
    </behavior>
  </serviceBehaviors>
</behaviors>
...
```

When impersonating for all operations, the **Impersonation** property of the **OperationBehaviorAttribute** applied to each method must also be set to either **Allowed** or **Required**.

Note: When a service has higher credentials than the remote client, the credentials of the service are used if the **Impersonation** property is set to **Allowed**. That is, if a low-privileged user provides its credentials, a higher-privileged service executes the method with the credentials of the service, and can use resources that the low-privileged user would otherwise not be able to use.

Impersonating the original caller programmatically within an operation

Use this option when you want to impersonate the original caller for a short duration in a service operation. Impersonation is a costly operation and is usually used for higher-privileged original callers, hence you would use impersonation only when needed to reduce the potential attack surface. Perform programmatic impersonation as shown in the following example:

```
public string GetData(int value)
{
 using (ServiceSecurityContext.Current.WindowsIdentity.Impersonate())
 {
     // return the impersonated user (original users identity)
     return string.Format("Hi, {0}, you have entered: {1}",
          WindowsIdentity.GetCurrent().Name, value);
 }
}
```

Note: It is important to revert to impersonation. Failure to do so can form the basis for denial of service and elevation of privilege attacks. In the example above, the **using** statement ensures that the impersonation is reverted after execution of the **using** block.

Additional Resources

- For more information, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Consider Using Programmatic Instead of Declarative Impersonation

Use programmatic impersonation to impersonate the original caller or the ASP.NET service account calling into your service. Programmatic impersonation allows you to impersonate on specific lines of code rather than the entire operation. Although this finer-grained approach to impersonation can reduce security risk, be aware that it is easier to make a mistake during implementation that could leave your code impersonating at higher privilege in the event of an error. Use the **using** statement to revert impersonation automatically.

The following code snippet shows how to impersonate programmatically:

```
public string GetData(int value)
{
 using (ServiceSecurityContext.Current.WindowsIdentity.Impersonate())
 {
```

```
     // return the impersonated user (original users identity)
     return string.Format("Hi, {0}, you have entered: {1}",
          WindowsIdentity.GetCurrent().Name, value);
 }
}
```

Additional Resources

- For more information, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of "WCF 3.5 security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

When Impersonating Programmatically, Be Sure to Revert to the Original Context

When using programmatic impersonation, revert to the original security context as soon as possible. If you do not remember to revert, your application's attack surface will be increased because it will be running under higher privileges than necessary. Use the **using** statement to revert impersonation automatically.

The following code snippet shows how to impersonate programmatically:

```
public string GetData(int value)
{
 using (ServiceSecurityContext.Current.WindowsIdentity.Impersonate())
 {
     // return the impersonated user (original users identity)
     return string.Format("Hi, {0}, you have entered: {1}",
          WindowsIdentity.GetCurrent().Name, value);
 }
}
```

Additional Resources

- For more information, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

When Impersonating Declaratively, Only Impersonate on the Operations That Require It

Only impersonate on specific operations that require it. If you impersonate on operations that do not require the additional privileges, you will increase your attack surface as well as the potential impact of an exploit.

Impersonation is a costly operation and is usually used for higher-privileged original callers. Use impersonation selectively only on the operations that need to reduce the potential attack surface. You can impersonate declaratively by applying the **OperationBehaviorAttribute** attribute on any operation that requires client impersonation, as shown in the following code example:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
   return “test”;
}
```

Additional Resources

- For more information, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Consider Using the S4U Feature for Impersonation and Delegation When You Cannot Do a Windows Mapping

In many situations — for example, if your users access a WCF service over the Internet — you cannot use Kerberos authentication because firewalls prevent the client computer from directly communicating with the domain controller. Instead, your service must authenticate the client by using another approach, such as username authentication, or in an extranet scenario, client certificate authentication.

In such situations where you cannot map the username or certificate authentication directly to Windows accounts, you can consider using the protocol transition (S4U) feature that permits applications to use a non-Windows authentication mechanism to authenticate users, while still using Kerberos authentication and delegation to access downstream network resources. This allows your application to access downstream servers that require Windows authentication, and it allows you to use Windows auditing to track user access to back-end resources.

Use the **WindowsIdentity** constructor to create a Windows token giving only an account’s user principal name (UPN).

Important: To impersonate at the impersonation level, you must grant your process account the "Act as part of the operating system" user right. To impersonate at the delegation level, you must enable protocol transition in Active Directory in order to access network resources.

The following code snippet shows how to use this constructor to obtain a Windows token for a given user:

```
using System;
using System.Security.Principal;
```

```
public void ConstructToken(string upn, out WindowsPrincipal p)
{
  WindowsIdentity id = new WindowsIdentity(upn);
  p = new WindowsPrincipal(id);
}
```

Additional Resources

- For more information on the S4U feature and Protocol Transition, see "Using Protocol Transition—Tips from the Trenches" at http://msdn2.microsoft.com/en-us/magazine/cc163500.aspx
- For more information, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Consider Using the LogonUser API if Your WCF Service Cannot Be Trusted for Delegation

Use the Win32 **LogonUser** API (via **P/Invoke**) to create delegation-level impersonation tokens, but only when your WCF service cannot be trusted for delegation, because this option forces you to store usernames and passwords on your WCF service.

Use the Basic authentication mode to get the original user's impersonation token so that you will have access to the username and password. You can then get the impersonation token by using the LogonUser API. For service accounts, you will have to store the username and password securely, and then use the LogonUser API to get the impersonation token.

The following code example shows how the LogonUser API is used for impersonation:

```
using System.Runtime.InteropServices;
…
// Declare the logon types as constants
const long LOGON32_LOGON_NETWORK = 3;

// Declare the logon providers as constants
const long LOGON32_PROVIDER_DEFAULT = 0;

[DllImport("advapi32.dll",EntryPoint = "LogonUser")]
private static extern bool LogonUser(
           string lpszUsername,
           string lpszDomain,
           string lpszPassword,
           int dwLogonType,
           int dwLogonProvider,
           ref IntPtr phToken);
[DllImport("kernel32.dll", CharSet=CharSet.Auto)]
public extern static bool CloseHandle(IntPtr handle);

private void ImpersonateAndUse(string Username,
```

```
                                        string Password,
                                        string Domain)
{
  IntPtr token = new IntPtr(0);
  token = IntPtr.Zero;
  // Call LogonUser to obtain a handle to an access token.
  bool returnValue = LogonUser(Username, Domain,Password,
                               (int)LOGON32_LOGON_NETWORK,
                               (int)LOGON32_PROVIDER_DEFAULT,
                               ref token);
  if (false == returnValue)
  {
     int ret = Marshal.GetLastWin32Error();
     string strErr = String.Format("LogonUser failed with error code : {0}",
ret);
     throw new ApplicationException(strErr, null);
  }
  WindowsIdentity newId = new WindowsIdentity(token);
  WindowsImpersonationContext impersonatedUser = newId.Impersonate();
  try
  {
     // do the operations using original user security context
  }
  finally
  {
     // stop impersonating
     impersonatedUser.Undo();
     CloseHandle(token);
  }
}
```

Additional Resources

- For more information on the LogonUser API, see “How to validate user credentials on Microsoft operating systems” at http://support.microsoft.com/kb/q180548/
- For more information, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Use Constrained Delegation if You Have to Flow the Original Caller to the Back-end Services

Use delegation for flowing the impersonated original user’s security context (Windows identity) to the remote back-end service. On the remote back-end service, the original user’s Windows identity can be used to authenticate or impersonate the original caller, in order to restrict or authorize the original caller’s access to local resources.

When using delegation on Microsoft Windows Server® 2003 or later, use constrained delegation. This allows administrators to specify exactly which services on a downstream server or a domain account can be accessed when using an impersonated user's security context.

Additional Resources

- For more information, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

Message Validation

If you make unfounded assumptions about the type, length, format, or range of input, your application is unlikely to be robust. Input validation can become a security issue if an attacker discovers that you have made unfounded assumptions. The attacker can then supply carefully crafted input that compromises your application. Misplaced trust of user input is one of the most common and serious vulnerabilities in WCF applications.

Consider these guidelines to help avoid input data validation vulnerabilities:

- **If you need to validate parameters, use parameter inspectors.**
- **Use schemas with message inspectors to validate messages.**
- **Use regular expressions in schemas to validate format, range, or length.**
- **Implement the AfterReceiveRequest method to validate inbound messages on the service.**
- **Implement the BeforeSendReply method to validate outbound messages on the service.**
- **Implement the AfterReceiveReply method to validate inbound messages on the client.**
- **Implement the BeforeSendRequest method to validate outbound messages on the client.**
- **Validate operation parameters for length, range, format, and type.**
- **Do not rely on client-side validation.**
- **Avoid user-supplied file name and path input.**
- **Do not echo untrusted input.**

Each of these guidelines is described in the following sections.

If You Need to Validate Parameters, Use Parameter Inspectors

If you need to validate parameters passed to operations, use parameter inspectors. *Parameter inspectors* are extensibility points that are plugged into the WCF behaviors that allow the inspecting of parameter values during message exchange. Parameter validation can happen on both the client and the service.

You should validate all parameters exposed in WCF service operations to protect the service from attack by a malicious client. Conversely, you should also validate all return values received by the client to protect the client from attack by a malicious service.

You can use parameter inspectors to inspect simple types or types with fewer fields, passed to operations that will not result in complex validation logic. If you need to validate complex types, or data/message contracts with several fields to be validated, use schema validation with message inspectors.

Additional Resources

- For more information, see "How to: Inspect or Modify Parameters" at http://msdn2.microsoft.com/en-us/library/ms733747.aspx
- For more information on how to use parameter inspectors, see "How To – Perform Input Validation in WCF."

Use Schemas with Message Inspectors to Validate Messages

If you need to validate parameters, message contracts, or data contracts passed to operations, use schema validation with message inspectors. *Message inspectors* are extensibility points that are plugged into the WCF behaviors that allow the inspecting of messages during message exchange. Schema validation can happen on both the client and the service. Use schema validation with message inspectors to validate messages without needing to write validation code. Additionally, schema validation will allow validating of complex types, message contracts, and data contracts, passed to operations that will not result in complex validation logic.

Additional Resources

- For information on configuring and extending the runtime with behaviors, see "Configuring and Extending the Runtime with Behaviors" at http://msdn2.microsoft.com/en-us/library/ms730137.aspx
- For additional information about the **IEndPoint Behavior** interface, see "IEndpointBehavior Interface" at http://msdn2.microsoft.com/en-us/library/system.servicemodel.description.iendpointbehavior.aspx
- For additional information about the **IDispatchMessageInspector** interface, see "IDispatchMessageInspector Interface" at http://msdn2.microsoft.com/en-us/library/system.servicemodel.dispatcher.idispatchmessageinspector.aspx
- For additional information about the **IClientMessageInspector** interface, see "IClientMessageInspector Interface" at http://msdn2.microsoft.com/en-us/library/system.servicemodel.dispatcher.iclientmessageinspector.aspx
- For additional information about the **BehaviorExtensionElement** class, see "BehaviorExtensionElement Class" at http://msdn2.microsoft.com/en-us/library/system.servicemodel.configuration.behaviorextensionelement.aspx
- For more information on message inspectors, see "Message Inspectors" at http://msdn2.microsoft.com/en-us/library/aa717047.aspx

- For more information on how to use message inspectors, see "How To – Perform Message Validation with Schemas in WCF."

Use Regular Expressions in Schemas to Validate Format, Range, or Length

Consider using regular expressions in schemas to validate format, range, or length. This allows you to use complex validation logic without the need for implementing the complex validation code. Using regular expressions also allows you to decouple the validation logic from the business logic. Schema validation provides a way to create validation for data types using XML rules, for validity checks regard to format, length, and type. Schema validation also supports the use of regular expressions. Without regular expressions, complex validation code would be required.

The following example schema exemplifies the validation of integer with values between 1 and 5, the string of length 5, and a social security number and ZIP code with valid formats:

```
<xs:schema elementFormDefault="qualified"
targetNamespace="http://Microsoft.PatternPractices.WCFGuide"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://Microsoft.PatternPractices.WCFGuide">
  <xs:element name="GetData">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="CustomerInfo" nillable="false"
type="tns:CustomerData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="CustomerData">
    <xs:sequence>
      <xs:element name="CustomerID" type="tns:CustIDLimitor">
      </xs:element>
      <xs:element name="text" type="tns:CustomerN">
      </xs:element>
      <xs:element name="ssn" type="tns:SSN">
      </xs:element>
      <xs:element name="zip" type="tns:us-zipcode">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="SSN">
    <xs:restriction base="xs:token">
      <xs:pattern value="[0-9]{3}-[0-9]{2}-[0-9]{4}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="us-zipcode">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CustomerN">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
      <xs:maxLength value="5" />
```

```
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CustIDLimitor">
    <xs:restriction base="xs:int">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="GetDataResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="GetDataResult" nillable="false"
type="tns:CustomerData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Implement the AfterReceiveRequest Method to Validate Inbound Messages on the Service

If you need to perform validation of inbound messages on your service, implement the **AfterReceiveRequest** method of the message inspector's **IDispatchMessageInspector** interface. This will validate the message after the client request has arrived and before service operation invocation and deserialization.

Inbound message validation will be required if you want to validate the message when it is received by the client and before invoking the operations in the service. Input data validation represents one line of defense in the protection of your WCF application. Validate all parameters exposed in WCF service operations to protect the service from attack by a malicious client.

The following example shows how to implement the **AfterReceiveRequest** method:

```
object IDispatchMessageInspector.AfterReceiveRequest(ref
System.ServiceModel.Channels.Message request,
System.ServiceModel.IClientChannel channel,
System.ServiceModel.InstanceContext instanceContext)
{
    try
    {
        validateMessage(ref request);
    }
    catch (FaultException e)
    {
        throw new FaultException<string>(e.Message);
    }
    return null;

}
```

Additional Resources

- For more information, see “How to: Inspect and Modify Messages on the Service” at http://msdn2.microsoft.com/en-us/library/ms733104.aspx
- For more information, see “How to: Inspect or Modify Parameters” at http://msdn2.microsoft.com/en-us/library/ms733747.aspx

Implement the BeforeSendReply Method to Validate Outbound Messages on the Service

If you need to perform validation of outbound messages on your service, implement the **BeforeSendReply** method of the message inspector’s **IDispatchMessageInspector** interface . This will validate the message before sending the response to the client and the service operation invocation and serialization.

Outbound message validation is required if you want to validate the message before sending the response to the client, so that you can validate output parameters and message and data contracts.

The following code example shows how to implement the **BeforeSendReply** method:

```
void IDispatchMessageInspector.BeforeSendReply(ref
System.ServiceModel.Channels.Message reply, object correlationState)
{
    try
    {
        validateMessage(ref reply);
    }
    catch (FaultException fault)
    {
        // if a validation error occurred, the message is replaced
        // with the validation fault.
        reply = Message.CreateMessage(reply.Version, new
FaultException("validation error in reply message").CreateMessageFault() ,
reply.Headers.Action);
    }
}
```

Additional Resources

- For more information, see “How to: Inspect and Modify Messages on the Service” at http://msdn2.microsoft.com/en-us/library/ms733104.aspx
- For more information, see “How to: Inspect or Modify Parameters” at http://msdn2.microsoft.com/en-us/library/ms733747.aspx

Implement the AfterReceiveReply Method to Validate Inbound Messages on the Client

If you need to perform validation of inbound messages on the client, implement the **AfterReceiveReply** method of the message inspector’s **IClientMessageInspector** interface . This will

validate the message after the client response has arrived and before deserialization and the returning of the data to the client application.

The following example shows how to implement the **AfterReceiveReply** method:

```
void IClientMessageInspector.AfterReceiveReply(ref
System.ServiceModel.Channels.Message reply, object correlationState)
 {
     validateMessage(ref reply);
 }
```

Additional Resources

- For more information, see "How to: Inspect or Modify Messages on the Client" at http://msdn2.microsoft.com/en-us/library/ms733786.aspx
- For more information, see "How to: Inspect or Modify Parameters" at http://msdn2.microsoft.com/en-us/library/ms733747.aspx

Implement the BeforeSendRequest Method to Validate Outbound Messages on the Client

If you need to perform validation of outbound messages on client, implement the **BeforeSendRequest** method of the message inspector "s **IClientMessageInspector** interface . This will validate the message before sending the request to the service and after serialization.

Outbound message validation is required if you want to validate the message before sending the request to the service, so that you can validate input parameters and message and data contracts on the client.

The following example shows how to implement the **BeforeSendRequest** method:

```
object IClientMessageInspector.BeforeSendRequest(ref
System.ServiceModel.Channels.Message request,
System.ServiceModel.IClientChannel channel)
{
    validateMessage(ref request);
    return null;
}
```

Additional Resources

- For more information, see "How to: Inspect or Modify Messages on the Client" at http://msdn2.microsoft.com/en-us/library/ms733786.aspx
- For more information, see "How to: Inspect or Modify Parameters" at http://msdn2.microsoft.com/en-us/library/ms733747.aspx

Validate Operation Parameters for Length, Range, Format, and Type

Check for known good data and constrain input by validating it for type, length, format, and range. Do not trust *any* input. An attacker passing malicious input can attempt SQL injection,

cross-site scripting, and other injection attacks that aim to exploit your application's vulnerabilities.

If the client application that consumes your WCF service is a Web-based application, use the ASP.NET validator controls, such as the **RegularExpressionValidator**, **RangeValidator**, and **CustomValidator**, to validate and constrain input. Check all numeric fields for type and range. If you are not using server controls, you can use regular expressions and the **Regex** class. You can validate numeric ranges by converting the input value to an integer or double and then performing a range check.

Do Not Rely on Client-side Validation

Do not rely on client-side validation because it can be easily bypassed. While you may have control over the source code for the clients that call your service, clients can be reverse-engineered or built from scratch to attack your service. Use client-side validation to reduce round-trips to the server and to improve the user experience, but always use validation in the service itself to perform security checks.

Avoid User-supplied File Name and Path Input

Wherever possible, avoid writing code that accepts user-supplied file or path input. Failure to do this can result in attackers coercing your application into accessing arbitrary files and resources. If your application must accept input file names, file paths, or URL paths, validate that the path is in the correct format and that it points to a valid location within the context of your application.

File Names

Ensure that file paths only refer to files within your application's virtual directory hierarchy if that is appropriate. When checking file names, obtain the full name of the file by using the **System.IO.Path.GetFullPath** method.

File Paths

If you use **MapPath** to map a supplied virtual path to a physical path on the server, use the overloaded **Request.MapPath** method that accepts a **bool** parameter so that you can prevent cross-application mapping. The following code example shows this technique:

```
try
{
 string mappedPath = Request.MapPath( inputPath.Text,
                                      Request.ApplicationPath, false);
}
catch (HttpException)
{
 // Cross-application mapping attempted
}
```

Note: The final false parameter in **Request.MapPath()** prevents cross-application mapping. This means that a user cannot successfully supply a path that contains ".." to traverse outside of your application's virtual directory hierarchy.

Do Not Echo Untrusted Input

Do not echo input back to the user without first validating and/or encoding the data. Echoing input directly back to the user can make client applications that rely on your service susceptible to malicious input attacks, such as cross-site scripting.

Message Security

If your WCF application passes sensitive data over networks, consider the threats of eavesdropping, tampering, and unauthorized callers accessing your endpoint. In an Internet scenario where you do not have control over the intermediate systems, consider using message security.

Consider the following guidelines for choosing message security:

- **If you need to support clients over the Internet, consider using message security.**
- **If there are intermediaries between the client and service, consider using message security.**
- **If you need to support selective message protection, use message security.**
- **If you need to support multiple transactions per session using Secure Conversation, use message security.**
- **Do not pass sensitive information in SOAP headers when using HTTP transport and message security.**
- **If you need to support interoperability, consider setting negotiateServiceCredentials to false.**
- **If you need to streamline certificate distribution to your clients, consider negotiating the service credentials.**
- **If you need to limit the clients that will consume your service, consider setting negotiateServiceCredentials to false.**

Each of these guidelines is described in the following sections.

If You Need to Support Clients over the Internet, Consider Using Message Security

Use message security when your clients are deployed over the Internet and you cannot rely on transport security (SSL). Message security provides end-to-end security in the following ways:

- Because SSL does not provide protection for the initial client-server handshake, a man-in-the-middle attack can go undetected.
- You have less control of the communication between the client and service across the Internet. There is a chance of having intermediaries, which might break transport security.

The downside of using message security is potentially decreased performance due to the fact that each message much be encrypted individually. Large message packets especially can create lag. You can use **wsHttpBinding**, which by default uses message security and also supports interoperability because it uses text encoding.

Additional Resources

- For more information on message protection, see "Message Security in WCF" at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For a message security Q&A, see the Message Protection section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If There Are Intermediaries Between the Client and Service, Consider Using Message Security

Use message security in scenarios where there may be intermediaries inspecting the message before the final delivery. You can protect your messages by using message security to encrypt and sign your messages. By encrypting your messages, you protect your sensitive data from being compromised. By signing your messages, you protect the client and service from tampering and man-in-the-middle attacks by protecting message integrity.

The following configuration snippet shows how to use message security to protect the credentials when using **wsHttpBinding**:

```
<wsHttpBinding>
  <binding name="MessageAndUserName">
    <security mode="Message">
      <message clientCredentialType="UserName" algorithmSuite="Default" />
    </security>
  </binding>
</wsHttpBinding>
```

Additional Resources

- For more information on message protection, see "Message Security" at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For a message security Q&A, see the Message Protection section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Support Selective Message Protection, Use Message Security

If you need signatures but not encryption on your messages, use message security to allow selective reduction of the protection level. This gives you more flexibility than transport

security, especially if you do not need to protect specific bigger message payloads over the network.

Be aware that turning off encryption will allow an attacker to view the content of your messages, including credentials or other sensitive information.

You can set the protection level to signatures only on the entire service as follows:

```
[ServiceContract(ProtectionLevel=ProtectionLevel.Sign]
public interface IService
{
      string GetData(int value);
}
```

You can set the protection level to signatures only on a single method at a time as follows:

```
[OperationContract(ProtectionLevel=ProtectionLevel.Sign]
string GetData(int value);
```

Additional Resources

- For more information on message protection, see “Message Security” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For a message security Q&A, see the Message Protection section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Support Multiple Transactions per Session Using Secure Conversation, Use Message Security

If your WCF clients need to exchange multiple messages with the WCF service, you can use the Secure Conversation feature for improved performance. Using Secure Conversation means the token negotiation and authentication happens only once for all the requests in a session.

Secure Conversation is enabled with message security on all the standard bindings that support the WS-Security specification (for example, **WsHttpBinding**, **NetTcpBinding**, and **netMsmqBinding**).

The following configuration example shows the secure conversation turned on:

```
...
<wsHttpBinding>
  <binding name="NewBinding0">
    <security>
      <message establishSecurityContext="true" />
    </security>
  </binding>
</wsHttpBinding>
...
```

Additional Resources

- For more information on message protection, see “Message Security” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx

Do Not Pass Sensitive Information in SOAP Headers When Using HTTP Transport and Message Security

Do not use message security if you need to pass sensitive information in Simple Object Access Protocol (SOAP) headers over the HTTP protocol. Instead, use transport security to protect sensitive data passed in SOAP headers, such as user identities passed for auditing purposes.

Information contained in SOAP headers is sent in plain text format and can be stolen if you use message security. SOAP header information is signed by default using message security, so the information can be read but cannot be spoofed.

Additional Resources

- For more information on message protection, see “Message Security” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For a message security Q&A, see the Message Protection section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Support Interoperability, Consider Setting negotiateServiceCredentials to False

If you need to support clients that do not understand the WS-Trust and WS-SecureConversation specifications, set the **negotiateServiceCredentials** attribute to false.

For Anonymous, Username, or Certificate client credential types, setting this property to false implies that the service certificate must be made available to the client out of band, and that the client must specify the service certificate to be used. In the case of Windows credentials, setting this property to false causes an authentication based on KerberosToken. This requires that the client and service both be part of a Kerberos domain.

The following is a configuration example:

```
<wsHttpBinding>
  <binding name="MessageAndUserName">
    <security mode="Message">
      <message clientCredentialType="UserName" negotiateCredentials=”false”
algorithmSuite="Default" />
    </security>
  </binding>
</wsHttpBinding>
```

Additional Resources

- For more information on message protection, see “Message Security” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For a message security Q&A, see the Message Protection section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Streamline Certificate Distribution to Your Clients, Consider Negotiating the Service Credentials

Consider enabling **negotiateServiceCredential** if you need to streamline certificate distribution to your clients for message encryption. This option is only available with **wsHttpbinding**. Keep in mind that non-Microsoft clients will not be able to consume your service if you enable this option. Also consider that there is a performance penalty of negotiating credentials, due to message exchange. Additionally, consider that allowing negotiation of service credentials is less secure, thereby allowing any client to consume your service.

The following binding configuration shows how to set this option:

```
<binding name="BindingMessage">
   <security mode="Message">
      <message clientCredentialType="Windows"
negotiateServiceCredential="true" />
   </security>
</binding>
```

Additional Resources

- For more information on message protection, see “Message Security” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For message security Q&A, see the Message Protection section of “WCF 3.5 Questions and Answers“ at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=Questions%20and%20Answers&referringTitle=Home

If You Need to Limit the Clients That Will Consume Your Service, Consider Setting negotiateServiceCredentials to false

If you want to limit the clients that can consume your service, consider setting **negotiateServiceCredentials** to false. This option will force you to install a certificate on the client, in addition to a service certificate with a public key. On the service, you will need to install a certificate plus the client certificate with the public key. Negotiation of service credentials is only available with **wsHttpBinding**.

The following binding configuration shows how to set this option:

```
<binding name="BindingMessage">
   <security mode="Message">
      <message clientCredentialType="Windows"
negotiateServiceCredential="false" />
   </security>
```

Transport Security

If your WCF application passes sensitive data over networks, consider the threats of eavesdropping, tampering, and unauthorized callers accessing your endpoint. In an Intranet scenario where you have control over the intermediate systems, consider using transport security.

Consider the following guidelines when choosing transport security:

- **Use transport security when possible.**
- **If you need to support clients in an intranet, use transport security.**
- **If you need to support interoperability with non-WCF clients, use transport security.**
- **Use a hardware accelerator when using transport security.**

Each of these guidelines is described in the following sections.

Use Transport Security When Possible

Transport security secures the client-server communication channel by using Secure Sockets Layer (SSL) over HTTP and Transport Layer Security (TLS) over TCP. Transport security is transport-dependent and does not require that the communicating parties understand XML-level security concepts. This can improve interoperability.

Consider that, if you are using transport security, you cannot use Service Credential Negotiation or authentication types such as username or issue token (CardSpace).

Use the following criteria to decide whether or not to use transport security:

- **Point-to-point**. Transport security supports point-to-point communication and does not support intermediary scenarios or protocol transition.
- **Streaming**. Transport security can support streaming data scenarios.
- **Binding limitations**. Transport security does not work with **wsDualHttpBinding**.
- **Authentication limitations**. Transport security does not work with negotiation, username, issue token (CardSpace), or Kerberos direct authentication.

Additional Resources

- For more information on choosing a transport, see “Choosing a Transport” at http://msdn2.microsoft.com/en-us/library/ms733769.aspx

If You Need to Support Clients in an Intranet, Use Transport Security

Use transport security when your clients are deployed within an intranet because it provides point-to-point security and better performance compared to message security.

In an intranet, you have control over the communication between client and service, and very few chances of having any intermediaries that might break the transport security. You can use **netTCPbinding** for better performance than HTTP bindings. By default, **netTcpbinding** uses binary encoding and transport security.

Additional Resources

- For more information on choosing a transport, see "Choosing a Transport" at http://msdn2.microsoft.com/en-us/library/ms733769.aspx

If You Need to Support Interoperability with Non-WCF Clients, Use Transport Security

If you have non-WCF clients and they do not support the WS-Security specification, use transport security. Because message security requires the client to understand and support WS-Security specifications, it will not work with non-WCF clients.

Additional Resources

- For more information on message security, see "Message Security in WCF" at http://msdn.microsoft.com/en-us/library/ms733137.aspx
- For more information on transport security, see "Transport Security Overview" at http://msdn.microsoft.com/en-us/library/ms729700.aspx

Use a Hardware Accelerator When Using Transport Security

Transport security can benefit from SSL hardware acceleration that is performed on a network card in order to avoid burdening the host machine CPU with the encryption and decryption of the messages.

Transport security requires both the client and service to negotiate the details of the encryption. This is done automatically as part of the communication protocol in the respective binding. Hardware acceleration provides high throughput and may even make the security overhead unnoticeable.

Proxy Considerations

When creating a WCF service proxy, clients needs to access metadata that might consist of sensitive data such as service location, etc. It is important to secure the metadata because attackers can leverage this information and exploit your WCF services.

Consider the following guidelines when exposing your service metadata for client proxy creation:

- **Publish your WCF service metadata only when required.**
- **If you need to publish your WCF service metadata, publish it over the HTTPS protocol.**
- **If you need to publish your WCF service metadata, publish it using secure binding.**
- **If you turn off mutual authentication, be aware of service spoofing.**

Publish Your WCF Service Metadata Only When Required

Set the **httpGetEnabled** and **httpsGetEnabled** attributes to false on the serviceMetadata element, and remove any endpoints configured on your service that implement IMetadataExchange contracts.

This is especially important after your clients are built and deployed, and if you do not need other clients to discover and use the WCF service. If the metadata is exposed, unwanted clients will be able to generate proxy files (e.g., by using Svcutil.exe) and inspect potentially sensitive methods and parameters offered by the service. If your client programs already have access to the service proxy, set the **httpGetEnabled** attribute to false.

The following configuration disables sharing of service metadata:

```
<serviceMetadata httpGetEnabled="False" httpsGetEnabled="False"/>
```

Additional Resources

- For more information on publishing metadata endpoints, see “Publishing Metadata” at http://msdn2.microsoft.com/en-us/library/aa751951.aspx
- For more information, see “Security Considerations with Metadata” at http://msdn.microsoft.com/en-us/library/ms734741.aspx

If You Need to Publish Your WCF Service Metadata, Publish It over the HTTPS Protocol

Publish your service metadata over Secure HTTP (HTTPS) to protect clients from being spoofed when adding a service reference. Clients cannot be certain that they have added a reference to the right service if you expose your service metadata over HTTP. The service may have been spoofed through Domain Name System (DNS) poisoning or a man-in-the-middle attack.

To publish your service metadata over HTTPS, use **mexHttpsBinding** and configure a server certificate for the service.

Additional Resources

- For more detailed steps, see “How to: Secure Metadata Endpoints” at http://msdn.microsoft.com/en-us/library/ms733114.aspx
- For more information on publishing metadata, see “Publishing Metadata“ at http://msdn2.microsoft.com/en-us/library/aa751951.aspx
- For more information, see “Security Considerations with Metadata” at http://msdn.microsoft.com/en-us/library/ms734741.aspx

If You Need to Publish Your WCF Service Metadata, Publish It Using Secure Binding

To protect service metadata from unauthorized access, you can use a secure binding for your metadata endpoint. The service metadata that a WCF service publishes contains a detailed description of the service and may intentionally or unintentionally contain sensitive

information. For example, service metadata may contain information about infrastructure operations that was not intended to be broadcast publicly.

You can use any standard binding (which has security features) you want for the mex service endpoint. The only requirement is to use the **IMetadataExchange** contract.

Additional Resources

- For more information on using secure bindings, see Nicholas Allen's blog at http://blogs.msdn.com/drnick/archive/2006/08/31/733173.aspx
- For more information, see "Security Considerations with Metadata" at http://msdn.microsoft.com/en-us/library/ms734741.aspx

If You Turn Off Mutual Authentication, Be Aware of Service Spoofing

Be aware that your service may be spoofed by a malicious attacker if you are running your service in a scenario in which mutual authentication has been turned off. Without mutual authentication, calls to your service could be diverted to a malicious service through DNS poisoning or a man-in-the-middle attack.

The follow scenarios will result in mutual authentication being turned off:

- If you turn off message and transport security on your binding
- If you use **basicHttpBinding**, which has message and transport security turned off by default
- If you use NTLM authentication

Additional Resources

- For more information on authentication, see "Authentication" at http://msdn2.microsoft.com/en-us/library/ms733082.aspx
- For more information on choosing a transport, see "Choosing a Transport" at http://msdn2.microsoft.com/en-us/library/ms733769.aspx

Sensitive Data

Sensitive data usually needs to be protected in persistent storage, in memory, and while it is on the network. Wherever possible, look for opportunities to avoid storing sensitive data. Use encryption to make sure that sensitive data cannot be viewed.

Follow these guidelines to help protect sensitive data:

- **Avoid plain-text passwords or other sensitive data in configuration files.**
- **Use platform features to manage keys where possible.**
- **Protect sensitive data over the network.**
- **Do not cache sensitive data.**
- **Minimize exposure of secrets in memory.**
- **Be aware that basicHttpBinding will not protect sensitive data by default.**

- **Use appropriately sized keys.**

Avoid Plain-text Passwords or Other Sensitive Data in Configuration Files

Avoid putting any sensitive information in the configuration files or within code. If you have to store user credentials or any other sensitive information in the configuration sections, encrypt the configuration sections by using one of the protected configuration providers. The sensitive Information should not be stored in plain text, because an attacker that can compromise a server will be able to read those credentials unless they are adequately protected.

In the .NET Framework version 2.0 and later, there are two libraries that provide encryption facilities for connection strings: DPAPI and RSA. If your application is deployed in a Web farm, you should use the RSA protected configuration provider because of the ease with which RSA keys can be exported. Otherwise, you should use the DPAPI protected configuration provider.

Additional Resources

- For more information on using DPAPI, see "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI" at http://msdn2.microsoft.com/en-us/library/ms998280.aspx
- For more information on using RSA, see "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA" at http://msdn2.microsoft.com/en-us/library/ms998283.aspx

Use Platform Features to Manage Keys Where Possible

Use platform features where possible to avoid managing keys yourself. For example, by using DPAPI, the encryption key is derived from an account's password, so Windows handles this for you.

Protect Sensitive Data over the NetworkNetwork

Protect sensitive data when it is transmitted over the network. Consider where items of sensitive data, such as credentials and application-specific data, are transmitted over a network link. Using a safe protocol to transmit information is important because it adds protection against inadvertent eavesdropping and modification during transport.

If you need to send sensitive data between the client and WCF service, consider using message or transport security. If you need to protect server-to-server communication, such as between your WCF service and a Microsoft SQL Server® database, consider Internet Protocol Security (IPSec) or SSL.

Do Not Cache Sensitive Data

If your service method contains data that is sensitive, such as a password, credit card number, or account status, it should not be cached. If sensitive data is cached on the client machine, it has serious security implications because it leaves interesting data available to attackers.

Perform the following steps to ensure that sensitive data is not cached:

1. **Review operations for sensitive data**. Review all of your operations for usage of sensitive data. This could include but is not limited to:
    - Information that either contains personally identifiable information (PII) or can be used to derive PII that should not be shared with users
    - Information that a user provides that they would not want shared with other users of the application
    - Information that comes from an external trusted source that is not designed to be shared with users
2. **Review the operations for caching of sensitive data**. Review how each operation manages sensitive data and ensure that it is not cached. There are three patterns of sensitive data caching that you can review for:
    - Custom caching code such as use of a **Dictionary** or **SortedList** object
    - Use of the ASP.NET cache via System.Web.Caching.Cache.
    - Use of an Enterprise Library caching block

Minimize Exposure of Secrets in Memory

When manipulating secrets, consider how the secret data is stored in memory. How long is the secret data retained in clear-text format? Clear-text secrets held in your process address space are vulnerable if an attacker is able to probe your application's address space. Also, if the page of memory containing the secret is swapped out to the page file, the secret data is vulnerable if someone gains access to the page file. Similarly, clear text secrets held in memory appear in the crash dump file if a process crashes.

To minimize the exposure of secrets in memory, consider the following measures:

- **Avoid creating multiple copies of the secret.** Having multiple copies of the secret data increases your attack surface. Pass references to secret data instead of making copies of the data. Also realize that if you store secrets in immutable **System.String** objects, a new copy is created after each string manipulatio.
- **Keep the secret encrypted for as long as possible.** Decrypt the data at the last possible moment before you need to use the secret.
- **Clean the clear-text version of the secret as soon as you can.** Replace the clear-text copy of the secret data with zeros as soon as you have finished with it.

Prior to .NET Framework 2.0, the use of byte arrays was recommended to help implement these guidelines. Byte arrays can be pinned in memory, encrypted, and replaced with zeros. In .NET Framework 2.0, use **SecureString** instead.

Be Aware That basicHttpBinding Will Not Protect Sensitive Data by Default

If you use **basicHttpBinding**, be aware that message security and transport security are turned off by default. All of the bindings, except for **basicHttpBinding**, have either message or transport security turned on by default. If you want to secure your WCF messages on the

network, you will need to explicitly turn on either message or transport security for **basicHttpBinding**.

Additional Resources

- For more information on choosing a transport, see "Choosing a Transport" at http://msdn2.microsoft.com/en-us/library/ms733769.aspx

Use Appropriately Sized Keys

Choosing a key size represents a trade-off between performance and security. If you choose a key that is too small, the data that you thought was well-protected can be vulnerable to attack. If you choose a key that is too large, your system will be subject to a performance impact without a commensurate real-world improvement in security. The appropriate key size changes based on the cryptographic algorithm in use, and also changes over time as machine processing speeds increase and attack techniques become more sophisticated.

The following recommendations will give you a margin of safety without sacrificing too much performance:

- When you use an asymmetric algorithm (RSA), choose a 2048-bit key
- When you use a symmetric algorithm (AES), choose a 128-bit key

Deployment Considerations

To avoid introducing vulnerabilities when you deploy your WCF application into a production environment, follow these guidleines:

- **Do not use temporary certificates in production.**
- **If you are using Kerberos authentication or delegation, create an SPN.**
- **Use IIS to host your WCF service wherever possible.**
- **Use a least-privileged account to run your WCF service.**
- **Protect sensitive data in your configuration files.**

Do Not Use Temporary Certificates in Production

Use temporary certificates when you are developing and testing your WCF service. Temporary certificates are less expensive than production certificates and are easier to create. Once you are ready to deploy, replace your temporary certificate with a production certificate provided by a certificate authority (CA).

Temporary certificates can be created by using the **makecert** utility. If you are deploying a WCF service for a real-world production environment, use a certificate provided by a CA such as Microsoft Windows Server 2003 Certificate Server or a third party.

Additional Resources

- For more information on how to create a certificate, see "Certificate Creation Tool" at http://msdn2.microsoft.com/en-us/library/bfsktky3(VS.80).aspx

If You Are Using Kerberos Authentication or Delegation, Create an SPN

If you are using Kerberos authentication, create a service principle name (SPN). Without an SPN, the Kerberos authentication will stop working when you switch from a machine account, such as Network Service, to a domain account.

Only SPNs can be configured for delegation in the Active Directory. In a production scenario using delegation, where you want to run the WCF service using a low-privileged custom domain account, you need to create an SPN for that account in order to enable delegation

To create an SPN for a domain account, run the Setspn tool from a command prompt as follows:

```
setspn -A HTTP/webservername domain\customAccountName
setspn -A HTTP/webservername.fullyqualifieddomainname
domain\customAccountName
```

This creates an SPN for the custom domain account (domain\customAccountName) and associates the account with the HTTP service on the specified WCF server. By running the command twice as shown above, you can associate the account with the NetBIOS server name and the fully qualified domain name (FQDN) of the server. This ensures that the SPN is established correctly even if your environment does not consistently use FQDNs.

Additional Resources

- For more information on SPNs, see “Setspn.exe: Manipulate Service Principal Names for Accounts” at http://technet2.microsoft.com/windowsserver/en/library/b3a029a1-7ff0-4f6f-87d2-f2e70294a5761033.mspx?mfr=true

Use IIS to Host Your WCF Service Wherever Possible

Use Internet Information Services (IIS) to host your WCF service because it provides a large number of features for efficient service management and scalability. IIS 6.0 only supports bindings over HTTP, so if you need to use TCP, MSMQ, or named pipes, you should host in a Windows service instead. IIS 7.0 supports all of the commonly used transport protocols such as HTTP, TCP, MSMQ, and named pipes. By using IIS as your WCF service host, you can take full advantage of IIS features such as process recycling, idle shutdown, process health monitoring, and message-based activation.

Perform the following steps when you want to develop and deploy a WCF service that is hosted in IIS:

1. Ensure that IIS, WCF, and the WCF activation component are correctly installed and registered.
2. Create a new IIS application, or reuse an existing ASP.NET application.
3. Create a .svc file for the WCF service.
4. Deploy the service implementation to the IIS application.
5. Configure the WCF service.

Additional Resources

- For more information, see "How to: Host a WCF Service in IIS" at http://msdn.microsoft.com/en-us/library/ms733766.aspx

Use a Least-Privileged Account to Run Your WCF Service

Use a least-privileged account to host your WCF service because it will reduce your application's attack surface and reduce the potential damage if you are attacked. If the service account requires additional access rights on infrastructure resources such as MSMQ, the event log, performance counters, and the file Ssstem, appropriate permissions should be given to these resources so that the WCF service can run successfully. If your service needs to access specific resources on behalf of the original caller, use impersonation and delegation to flow the caller's identity for a downstream authorization check.

In a development scenario, use the local network service account, which is a special built-in account that has reduced privileges. In a production scenario, create a least-privileged custom domain service account.

Additional Resources

- For more information on how to create a custom service account, see "How To - Create a Service Account for an ASP.NET 2.0 Application" at http://msdn2.microsoft.com/en-us/library/ms998297.aspx

Protect Sensitive Data in Your Configuration Files

Protect the sensitive data, such as SQL connection strings, in your configuration files by encrypting it. Connection strings stored in plaintext are dangerous, because an attacker that can compromise a server will be able to read those connection strings. Even if a machine is not compromised, connection strings stored in plaintext are accessible to administrators and any other users with sufficient privileges on the host machine and/or Windows domain.

Use DPAPI to encrypt the sensitive data in the configuration file on your WCF server machine. To encrypt the <connectionStrings> section by using the DPAPI provider with the machine-key store (the default configuration), run the following command from a command window:

```
aspnet_regiis -pe "connectionStrings" -app "/MachineDPAPI" -prov
"DataProtectionConfigurationProvider"
```

The aspnet_regiis options are:

- **-pe** – Specifies the configuration section to encrypt.
- **-app** – Specifies your Web application's virtual path. If your application is nested, you need to specify the nested path from the root directory; for example, "/test/aspnet/MachineDPAPI".
- **-prov** – Specifies the provider name.

If you need to encrypt configuration file data on multiple servers in a Web farm, use the RSA protected configuration provider because of the ease with which you can export RSA key containers.

Additional Resources

- For more information on using DPAPI, see “How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI” at http://msdn2.microsoft.com/en-us/library/ms998280.aspx
- For more information on using RSA, see “How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA” at http://msdn2.microsoft.com/en-us/library/ms998283.aspx

Practices at a Glance – WCF Security

Index

Auditing and Logging

Authentication

Authorization

Configuration Management

- How to Create a Service Account for Your WCF Service
- How to Stop Clients from Referencing Your Service
- How to Protect Against Message Replay Attacks

Deployment Considerations

- How to Configure Certificates to Enable SSL In IIS
- How to Map Windows Accounts with Certificates
- How to Create a Service Principle Name (SPN)
- How to Configure WCF For NATs and Firewalls
- How to Create an X.509 Certificate

Exception Management

- How to Shield Exception Information with Fault Contracts
- How to Check the State of a Channel in WCF Proxy Client
- How to Avoid Faulting the Channels with Fault Contracts
- How to Create an Error Handler to Log Details of Faults for Auditing Purposes
- How to Handle Unhandled Exceptions in Downstream Services
- How to Throw an Exception with Complex Types or Data Contracts with a Fault Exception
- How to Handle Unknown Faults in a Service
- How to Implement a Data Contract to Propagate Exception Details for Debugging Purposes
- How to Implement Fault Contracts in Callback Functions

Hosting

- How to Host WCF in IIS
- How to Host WCF in a Windows Service
- How to Self-host WCF
- How to Configure a Least-privileged Account to Host your Service

Impersonation/Delegation

- How to Choose Between a Trusted Subsystem and Impersonation/Delegation
- How to Impersonate the Original Caller When Using Windows Authentication
- How to Impersonate Programmatically in WCF
- How to Impersonate Declaratively in WCF
- How to Delegate the Original Caller to Call Back-end Services When Using Windows Authentication
- How to Impersonate the Original Caller Without Windows Authentication
- How to Impersonate the Original Caller Using S4U Kerberos Extensions
- How to Delegate the Original Caller Using S4U Kerberos Extensions
- How to Impersonate and Delegate Using the LogonUser Windows API
- How to Flow the Original Caller from an ASP.NET Client to WCF
- How to Control Access to a Remote Resource Based on the Original Caller's Identity

Message Validation

- How to Protect Your Service from Malicious Messages
- How to Protect Your Service from Malicious Input
- How to Protect Your Service from Denial of Service Attacks
- How to Validate Parameters with Parameter Inspectors
- How to Validate Parameters with Message Inspectors Using Schemas
- How to Validate Data Contracts with Message Inspectors Using Schemas
- How to Validate Message Contracts with Message Inspectors Using Schemas
- How to Use Regular Expressions to Validate Format, Range, and Length in Schemas
- How to Validate Inbound Messages on a Service
- How to Validate Outbound Messages on a Service
- How to Validate Outbound Messages on the Client
- How to Validate Inbound Messages on the Client
- How to Validate Input Parameters
- How to Validate Output Parameters

Message Security

- How to Use Message Security
- How to Control the Level of Message Encryption
- How to Use Out-of-Band Credentials with Message Security

Proxy Considerations

- How to Avoid Proxy Spoofing
- How to Publish Service Metadata for Your Clients
- How to Create a Proxy for an IIS-hosted Service with Certificate Authentication and Transport Security

Sensitive Data

- How to Encrypt Sensitive Data in Configuration Files
- How to Protect Sensitive Data in Memory
- How to Protect Sensitive Data on the Network

Transport Security

- How to Use Transport Security
- How to Use Secure Conversations in WCF

X.509 Certificates

- How to Create a Temporary X.509 Certificate for Transport Security
- How to Create a Temporary X.509 Certificate for Message Security
- How to Create a Temporary X.509 Certificate For Certificate Authentication

Auditing and Logging

- **How to Audit Security Events**
- **How to Enable WCF Message Logging**
- **How to Enable WCF Tracing**
- **How to Use Health Monitoring in WCF**
- **How to Filter Sensitive Data from Your Logs**
- **How to View Log Information**
- **How to View Trace Information**
- **How to Log Traces to a WMI Provider**
- **How to Turn Off Audit Failure Suppression**

How to Audit Security Events

You can use the auditing feature in WCF to audit security events such as authentication and authorization failures. WCF service auditing can allow you to detect an attack that has occurred or is in progress. In addition, auditing can help you debug security-related problems. For example, if an error in the configuration of the authorization or checking policy accidentally denies access to an authorized user, you can discover and isolate the cause of this error by examining the auditing events in the event log.

Perform the following steps to enable auditing of authentication and authorization for your WCF service:

1. Open the web.config file of the WCF service by using the Configuration Editor tool (SvcConfigEditor.exe).
2. In the Configuration Editor, navigate to the **Advanced** node.
3. Select the **Behavior: ServiceBehavior** section and add a new service behavior extension element.
4. In the **Adding Behavior Element Extension Sections** dialog box, select **serviceSecurityAudit** and then click **Add**.
5. In the **Configuration** section, under **Service Behaviors**, select the **serviceSecurityAudit** option.
6. Set the **MessageAuthenticationAuditLevel** attribute to **SuccessOrFailure** by choosing this option from the drop-down list.
7. Set the **ServiceAuthorizationAuditLevel** attribute to **SuccessOrFailure** by choosing this option from the drop-down list.
8. In the **Configuration Editor**, on the **File** menu, click **Save**.
9. In Microsoft Visual Studio®, verify your configuration. The configuration should look as follows.

```
…
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
```

```
        <serviceSecurityAudit
messageAuthenticationAuditLevel="SuccessOrFailure" />
        <serviceSecurityAudit
serviceAuthorizationAuditLevel="SuccessOrFailure" />
    </behavior>
  </serviceBehaviors>
</behaviors>
…
```

Additional Resources

- For more information on auditing, see “Auditing Security Events” at http://msdn2.microsoft.com/en-us/library/ms731669.aspx
- For more information on auditing in WCF, see “How to: Audit Windows Communication Foundation Security Events” at http://msdn2.microsoft.com/en-us/library/ms734737.aspx
- For auditing guidelines, see the Auditing and Logging section of “WCF 3.5 Security Guidelines“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Enable WCF Message Logging

You can enable message logging to log the messages processed by your service. Message logging can be used to diagnose your applications and analyze the root cause of problems. Message logging is not turned on by default; turn on message logging by setting attributes on the <messagelogging> element in your configuration file and then add a trace listener to log the events to a file.

Enabling WCF Message Logging

Perform the following steps to enable WCF message logging:

1. Open the web.config file of the WCF service by using the Configuration Editor tool (SvcConfigEditor.exe).
2. In the Configuration Editor, navigate to the **Diagnostics** node and then click the **Enable Message Logging** link.
   This enables message logging for your service and also creates a listener (**ServiceModelMessageLoggingListener**) and a source (**System.ServiceModel.MessageLogging**) under the Listeners and Sources folders, respectively.

Configuring Message Logging Levels

You can configure message logging levels at both the service and transport levels. Perform the following steps to configure the message logging levels:

1. In the left pane of the Configuration editor, select **MessageLogging** under the **Diagnostics** node.

2. Set the **LogMessagesAtServiceLevel** attribute to **True** by choosing this option from the drop-down list.
   The **LogMessagesAtTransportLevel** attribute is **True** by default.

Determining Where Messages Will Be Logged

Perform the following step to determine where the messages will be logged:

- Select **ServiceModelMessageLoggingListener** under the **Listeners** node and note the value of the **InitData** attribute. The default location where messages are logged is **c:\inetpub\wwwroot\WCFService\web_messages.svclog**.

  The configuration file should look as follows:

```
    <system.diagnostics>
        <sources>
            <source name="System.ServiceModel.MessageLogging"
                    switchValue="Warning, ActivityTracing">
                <listeners>
                    <add type="System.Diagnostics.DefaultTraceListener"
                         name="Default">
                        <filter type="" />
                    </add>
                    <add name="ServiceModelMessageLoggingListener">
                        <filter type="" />
                    </add>
                </listeners>
            </source>
        </sources>
        <sharedListeners>
            <add
initializeData="c:\inetpub\wwwroot\auditingwcf\web_messages.svclog"
type="System.Diagnostics.XmlWriterTraceListener, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
                name="ServiceModelMessageLoggingListener"
                traceOutputOptions="Timestamp">
                  <filter type="" />
            </add>
        </sharedListeners>
    </system.diagnostics>

 <system.serviceModel>
    <diagnostics>
      <messageLogging logMalformedMessages="true"
                      logMessagesAtServiceLevel="true"
                      logMessagesAtTransportLevel="true" />
    </diagnostics>
```

Additional Resources

- For more information on auditing, see “Auditing Security Events” at http://msdn2.microsoft.com/en-us/library/ms731669.aspx
- For message logging information, see “Message Logging” at http://msdn2.microsoft.com/en-us/library/ms731859.aspx

- For more information on auditing in WCF, see **"**How to: Audit Windows Communication Foundation Security Events" at http://msdn2.microsoft.com/en-us/library/ms734737.aspx
- For auditing guidelines, see the Auditing and Logging section of "WCF 3.5 Security Guidelines" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Enable WCF Tracing

Use WCF tracing to help debug your WCF service by logging all operations on your service.

Enabling Tracing

Perform the following steps to enable tracing:

1. Open the web.config file of the WCF service by using the Configuration Editor tool (SvcConfigEditor.exe).
2. In the Configuration Editor, navigate to the **Diagnostics** node and then click the **Enable Tracing** link.
   This enables tracing of your WCF service and also creates a listener (ServiceModelTraceListener) and a source (SystemServiceModel) under the Listeners and Sources folders, respectively.

Determining Where Traces Will Be Written

Perform the following step to determine where the traces will be written:

- Select **ServiceModelTraceListener** under the **Listeners** node and note the value of the **InitData** attribute. The default location where trace messages are written is **c:\inetpub\wwwroot\auditingwcf\web_tracelog.svclog**.

  The configuration file should look as follows:

```
<system.diagnostics>
    <sources>
        <source name="System.ServiceModel"
                switchValue="Warning, ActivityTracing"
                propagateActivity="true">
            <listeners>
                <add type="System.Diagnostics.DefaultTraceListener"
                     name="Default">
                    <filter type="" />
                </add>
                <add name="ServiceModelTraceListener">
                    <filter type="" />
                </add>
            </listeners>
        </source>
    </sources>
```

```
        <sharedListeners>
            <add
          initializeData="c:\inetpub\wwwroot\auditingwcf\web_tracelog.svclog"
                type="System.Diagnostics.XmlWriterTraceListener, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
                name="ServiceModelTraceListener"
                traceOutputOptions="Timestamp">
                <filter type="" />
            </add>
        </sharedListeners>
    </system.diagnostics>
```

Additional Resources

- For more information on tracing, see “Tracing” at http://msdn.microsoft.com/en-us/library/ms730342.aspx
- For more information on using the WCF Service Trace Viewer Tool, see “Service Trace Viewer Tool” at http://msdn.microsoft.com/en-us/library/ms732023.aspx and “Examining WCF Diagnostic Traces Using Service Trace Viewer Tool (SvcTraceViewer.exe)” at http://blogs.msdn.com/alikl/archive/2007/10/23/examining-wcf-diagnostic-traces-using-service-trace-viewer-tool-svctraceviewer-exe.aspx
- For auditing guidelines, see the Auditing and Logging section of “WCF 3.5 Security Guidelines“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Use Health Monitoring in WCF

You can use the health monitoring feature in WCF to log custom events in your service based on business logic. You can use health monitoring to instrument your application and monitor user-management events around authentication and authorization. This instrumentation can help you to detect and react to potentially suspicious behavior. It also enables you to gather data on operations; for example, to track who is accessing your application and when user account passwords need to be reset.

Perform the following high-level steps to configure your WCF service to use health monitoring:

1. **Create a custom health monitoring event.**
2. **Configure your WCF service for health monitoring.**
3. **Instrument your application to raise a custom event.**

Each of these steps is detailed below.

1. **Create a custom health monitoring event.**
   Create a custom user management Web event by first creating a class library and then creating a class that inherits from **WebAuditEvent**, as follows:

```
using System.Web.Management;

 public class MyEvent : WebAuditEvent
    {

        public MyEvent(string msg, object eventSource, int eventCode)
            : base(msg, eventSource, eventCode)
        {
        }

        public MyEvent(string msg, object eventSource, int eventCode,
                       int eventDetailCode)
            : base(msg, eventSource, eventCode, eventDetailCode)
        {
        }

       public override void FormatCustomEventDetails(WebEventFormatter
            formatter)
        {
            base.FormatCustomEventDetails(formatter);

            // Display some custom event message
            formatter.AppendLine("Some Critical Event Fired");
        }
   }
```

2. Configure your WCF Service for health monitoring.

Add a health monitoring element to your configuration file as follows:

```
…
<system.web>
   <healthMonitoring>
         <eventMappings>
               <add name="Some Custom Event"
                    type="MyEventLibrary.MyEvent, MyEventLibrary"/>
         </eventMappings>
         <rules>
               <add name="Custom event"
                    eventName="Some Custom Event"
                    provider="EventLogProvider"
                    minInterval="00:00:01"/>
         </rules>
   </healthMonitoring>
</system.web>
…
```

3. Instrument your application to raise a custom event.

Instrument the WCF service by raising the custom event in a service contract as follows.

```
[OperationContract]
string InvokeCriticalEvent();

public string InvokeCriticalEvent()
    {
        MyEvent obj = new MyEvent("Invoking Some Custom Event",
                          this, WebEventCodes.WebExtendedBase + 1);
```

```
            obj.Raise();
            return "Critical event invoked";
        }
```

After completing these steps, you can verify that the custom events are in the system event log after calling the service method from a test client.

Additional Resources

- For more information on health monitoring, see "How To: Use Health Monitoring in ASP.NET 2.0" at http://msdn2.microsoft.com/en-us/library/ms998306.aspx
- For additional information on health monitoring, see "ASP.NET Health Monitoring Overview" at http://msdn.microsoft.com/en-us/library/bb398933.aspx
- For auditing guidelines, see the Auditing and Logging section of "WCF 3.5 Security Guidelines" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Filter Sensitive Data from Your Logs

You can use message filters to log messages that match the filter criteria. For example, you could use a message filter to remove personally identifiable information (PII) before it can get into log files.

Filters support the full XPath syntax. The following code shows how to configure a filter that records only messages that have a Simple Object Access Protocol (SOAP) header section:

```
<messageLogging logEntireMessage="true"
    logMalformedMessages="true"
    logMessagesAtServiceLevel="true"
    logMessagesAtTransportLevel="true"
    maxMessagesToLog="420">
    <filters>
        <add nodeQuota="10"
             xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
                 /soap:Envelope/soap:Header
        </add>
     </filters>
</messageLogging>
```

Filters provide a safety feature using the **nodeQuota** attribute, which limits the maximum number of nodes in the XPath Document Object Model (DOM) that can be examined to match the filter.

Additional Resources

- For more information on message logging, see "Configuring Message Logging" at http://msdn.microsoft.com/en-us/library/ms730064.aspx

How to View Log Information

You can use the SvcTraceViewer.exe utility to view both message log files and trace files. You can find this tool at <*Drive Name*>:\Program Files\Microsoft SDKs\Windows\v6.0\Bin.

This tool gives you a comprehensive analysis of the step-by-step process of the WCF service, showing each interaction with the WCF run time and the clients. It shows the object activities, messages, and all errors that occurred in the host's life. It also provides you a graphical view of the log or trace data.

Additional Resources

- For more information on using the Service Trace Viewer Tool, see "Service Trace Viewer Tool" at http://msdn.microsoft.com/en-us/library/ms732023.aspx and "Examining WCF Diagnostic Traces Using Service Trace Viewer Tool (SvcTraceViewer.exe)" at http://blogs.msdn.com/alikl/archive/2007/10/23/examining-wcf-diagnostic-traces-using-service-trace-viewer-tool-svctraceviewer-exe.aspx
- For more information on authentication, see "Authentication" at http://msdn2.microsoft.com/en-us/library/ms733082.aspx

How to View Trace Information

Perform the following steps to view trace information:

1. Enable tracing by adding configuration information to the application web.config or app.config file; for example:

```
<system.diagnostics>
    <trace autoflush="true" />
    <sources>
            <source name="System.ServiceModel"
                    switchValue="Information, ActivityTracing"
                    propagateActivity="true">
            <listeners>
               <add name="sdt"
                   type="System.Diagnostics.XmlWriterTraceListener"
                   initializeData= "WCFTraceLog.svclog" />
            </listeners>
         </source>
    </sources>
</system.diagnostics>
```

2. Navigate to the SvcTraceViewer.exe installation location (C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin) and run SvcTraceViewer.exe.
3. On the **File** menu, click **Open** and then navigate to the location where your trace files are stored.
4. Double-click the trace log file to open it.

Additional Resources

- For more information on tracing, see "Tracing" at http://msdn.microsoft.com/en-us/library/ms730342.aspx
- For more information on the Service Trace Viewer Tool, see "Service Trace Viewer Tool (SvcTraceViewer.exe)" at http://msdn.microsoft.com/en-us/library/ms732023.aspx
- For more information on authentication, see "Authentication" at http://msdn2.microsoft.com/en-us/library/ms733082.aspx
- For auditing guidelines, see the Auditing and Logging section of "WCF 3.5 Security Guidelines" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Log Traces to a WMI Provider

Perform the following steps to enable a Windows Management Instrumentation (WMI) provider for your service:

1. Open the web.config file of the WCF service by using the Configuration Editor tool (SvcConfigEditor.exe).
2. In the Configuration Editor, navigate to the **Diagnostics** node and then click the **Enable WMI Provider** link. The configuration file should look as follows.

```
<system.serviceModel>
  <diagnostics wmiProviderEnabled="true">
  ...
  </diagnostics>
  ...
</system.serviceModel>
```

3. To view the WMI trace information, you need to install WMI CIM Studio so that you can view the WMI interactions. WMI CIM Studio is a Microsoft ActiveX® component that plugs into Microsoft Internet Explorer®. You can get this as a free download available from Microsoft.

Additional Resources

- To download the WMI CIM Studio tool, see "WMI Administrative Tools" at http://www.microsoft.com/downloads/details.aspx?familyid=6430F853-1120-48DB-8CC5-F2ABDC3ED314&displaylang=en

How to Turn Off Audit Failure Suppression

By default, WCF will ignore audit failures and allow the service to continue running by setting the **SuppressAuditFailure** property to **true**. You can set this property to **false**, which will turn off audit failure suppression, thereby throwing an exception when there has been an auditing failure.

Perform the following step to turn off audit failure suppression:

- Set the **suppressAuditFailure** property to **false** as follows:

```
<configuration>
  <system.serviceModel>
    <behaviors>
      <behavior>
        <serviceSecurityAudit
                  auditLogLocation="Application"
                  suppressAuditFailure="false"
                  serviceAuthorizationAuditLevel="Failure"
                  messageAuthenticationAuditLevel="SuccessOrFailure" />
      </behavior>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Additional Resources

- For more information on auditing in WCF, see "How to: Audit Windows Communication Foundation Security Events" at http://msdn.microsoft.com/en-us/library/ms734737.aspx

Authentication

- **How to Authenticate Users Against the SQL Server Membership Provider**
- **How to Authenticate Users Against Active Directory**
- **How to Authenticate Users Against Active Directory Without Windows Authentication**
- **How to Authenticate Users with Certificates**
- **How to Map Certificates with Windows Accounts**
- **How to Authenticate Users Against a Custom User Store**

How to Authenticate Users Against the SQL Server Membership Provider

If your user information is already stored in a Microsoft SQL Server® Membership database, or if you are building an Internet-facing WCF application from scratch, you can use the SQL Server membership provider to authenticate your WCF service clients. The SQL Server membership provider authenticates all incoming client credentials against the credentials stored in the SQL Server Membership database. The membership feature is a good choice becauseit allows you to enable username authentication without writing and maintaining custom code.

Perform the following steps to configure the SQL Server membership provider to work with username authentication in your WCF application:

1. Configure your SQL Server database for membership. From a Visual Studio 2008 command prompt, run the following command:

   ```
   aspnet_regsql -S .\SQLExpress -E -A m -d <<YourDatabaseName>>
   ```

   In this command:

   - **-S** specifies the server, which is (.\SQLExpress) in this example.

- **-E** specifies to use Windows authentication to connect to SQL Server.
- **-A m** specifies to add only the membership feature. For simple authentication against a SQL Server user store, only the membership feature is required.
- **-d** specifies the SQL server database name. If this option is not used, a default aspnetdb database will be created.

For a complete list of the commands, run Aspnet_regsql /?

2. Modify your web.config file in your WCF service application by adding the following sections

```
<connectionStrings>
  <add name="MyLocalSQLServer"
       connectionString="Initial Catalog=<<YourDatabaseName>>;
      data source=.\sqlexpress;Integrated Security=SSPI;" />
</connectionStrings>

…
<system.web>
  ...
  <membership defaultProvider="MySqlMembershipProvider" >
    <providers>
      <clear/>
      <add name="MySqlMembershipProvider"
           connectionStringName="MyLocalSQLServer"
           applicationName="MyAppName"
           type="System.Web.Security.SqlMembershipProvider" />
    </providers>
  </membership>
</system.web>
…
```

3. Configure the service to use username authentication:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security>
        <message clientCredentialType="UserName" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

4. Configure the service to use the SQL Server membership provider:

```
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">

      <serviceCredentials>
        <userNameAuthentication
userNamePasswordValidationMode="MembershipProvider"
```

```
            membershipProviderName="MySqlMembershipProvider" />
        </serviceCredentials>

      </behavior>
    </serviceBehaviors>
  </behaviors>
  …
```

Additional Resources

- For more information, see "How To – Use Username Authentication with the SQL Server Membership Provider and Message Security in WCF from Windows Forms" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Use%20Username%20Authentication%20with%20the%20SQL%20Membership%20Provider%20and%20Message%20Security%20in%20WCF%20from%20Windows%20Forms&referringTitle=How%20Tos
- For more information, see "How to: Use the ASP.NET Membership Provider" at http://msdn.microsoft.com/en-us/library/ms731049.aspx

How to Authenticate Users against Active Directory

Use Windows authentication when both the client and service are in trusted domains, or when users are stored in local machine accounts, such as in an intranet scenario. By using Windows authentication with the Microsoft Active Directory® directory service, you benefit from a unified identity store, centralized account administration, enforceable account and password policies, and strong authentication that avoids sending passwords over the network.

If Windows authentication is not possible because of infrastructure limitations such as a firewall between clients and Active Directory, consider using username authentication instead. If you are using username authentication, the username/password for the user will be automatically mapped to a Windows account.

The following example shows how to configure the client credentials in WCF to use Windows authentication:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security>
        <message clientCredentialType="Windows" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
…
```

Additional Resources

- For more information, see "Selecting a Credential Type" at http://msdn.microsoft.com/en-us/library/ms733836.aspx

How to Authenticate Users Against Active Directory Without Windows Authentication

Use username authentication to authenticate users against Active Directory or local machine accounts, when you cannot use Windows authentication. By default, username authentication will map your user's credentials to Windows accounts and authenticate the users against Active Directory.

The following code snippet configures a WCF service to use username authentication:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security>
        <message clientCredentialType="UserName" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
…
```

Note: Use message security to protect user credentials passed over the network.

Additional Resources

- For more information, see "Selecting a Credential Type" at http://msdn.microsoft.com/en-us/library/ms733836.aspx

How to Authenticate Users with Certificates

Client certificates can authenticate a client service account or multiple users to a WCF service. If you use a client certificate for each user, you can map each certificate to a Windows account.

Perform the following steps to authenticate users by using a client-side certificate:

1. Install the service certificate on the WCF service machine:
   - If you are using message security, configure service credentials to set the name and location of the service certificate.
   - If you are using transport security with **wsHttpBinding**, install the service certificate on Internet Information Services (IIS) and configure the virtual directory to require Secure Sockets Layer (SSL) and client certificates.
2. Configure the service to use certificates for the client credentials type, as shown in the following example:

```
<wsHttpBinding>
  <binding name="WSHttpBinding_ICalculator">
```

```
            <security mode="Message">
              <message clientCredentialType="Certificate" />
            </security>
          </binding>
        </wsHttpBinding>
```

3. Install the service certificate on the client machine.
4. Configure the endpoint behavior to set the name and location of the client certificate.

Note: Make sure that the root CA certificate is in the Trusted Root Certification Authorities location on both the server and client machines.

Additional Resources

- For more information on working with WCF and certificates, see "Working with Certificates" at http://msdn.microsoft.com/en-us/library/ms731899.aspx
- For more information on using certificates with WCF, see "How To – Use Certificate Authentication and Message Security in WCF Calling from Windows Forms" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Use%20Certificate%20Authentication%20and%20Message%20Security%20in%20WCF%20calling%20from%20Windows%20Forms&referringTitle=How%20Tos

How to Map Certificates with Windows Accounts

Client certificates are not mapped to Windows accounts by default. To do so, you set the **mapClientCertificateToWindowsAccount** property to **true**.

Perform the following steps to map certificates to Windows accounts:

1. Decide between the IIS certificate mapping versus Active Directory certificate mapping.
    a. IIS certificate mapping is useful if you need only a limited number of mappings, or a different mapping on each WCF service.
    b. Use Active Directory certificate mapping when the account mappings are identical on all IIS servers. Active Directory mapping is easier to maintain than IIS mapping because you only have to create the mapping in one location.
2. Configure IIS / Active Directory for mapping the certificates.
3. After you have enabled the client certificate mapping feature, set the **mapClientCertificateToWindowsAccount** property to **true** as follows:

```
<serviceBehaviors>
  <behavior name="MyServiceBehaviorForWebHttp">

     <serviceCredentials>
      <clientCertificate>
       <authentication mapClientCertificateToWindowsAccount="true" />
      </clientCertificate>
     </serviceCredentials>

  </behavior>
</serviceBehaviors>
```

Additional Resources

- For more information on using certificates with WCF, see "Working with Certificates" at http://msdn.microsoft.com/en-us/library/ms731899.aspx
- For more information on mapping certificates to Windows accounts, see "Map certificates to user accounts" at http://technet2.microsoft.com/WindowsServer/f/?en/library/0539dcf5-82c5-48e6-be8a-57bca16c7e171033.mspx
- For more information on mapping certificates to Active Directory, see "Mapping Client Certificates with Directory Service Mapping" at http://technet2.microsoft.com/windowsserver/en/library/7cce4299-28f2-45fa-8730-4e0cbe3be8561033.mspx?mfr=true
- For more information on certificate mapping strategies, see "Mapping Strategies" at http://technet2.microsoft.com/windowsserver/en/library/aa61c564-1599-4414-a12d-2f64786f6ec31033.mspx?mfr=true

How to Authenticate Users Against a Custom User Store

To authenticate users against a custom user store, configure your application to use username authentication with a custom username and password validator. Configure the custom validator in a service behavior and implement it in a class library. Your service uses the username and password validator to authenticate your users based on your custom user store.

Configuring a custom validator for your WCF service

The following configuration snippet shows how to configure a custom validator for your WCF service:

```
<system.serviceModel>
 ...
  <behaviors>
    <serviceBehaviors>
      <behavior name="ServiceBehavior">
        ...
        <serviceCredentials>
          <serviceCertificate findValue=" CN=FabrikamEnterprises " />
          <userNameAuthentication userNamePasswordValidationMode="Custom"
                      customUserNamePasswordValidatorType=
                             "MyUserNamePasswordValidator,Host" />
        </serviceCredentials>
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

Implementing a custom username and password validator

The following code snippet shows how to implement a custom username and password validator:

```
using System;
using System.Collections.Generic;
```

```
using System.IdentityModel.Selectors;
using System.IdentityModel.Tokens;
using System.Text;

namespace DerivativesCalculator
{
 public class MyUserNamePasswordValidator : UserNamePasswordValidator
 {
    public override void Validate(string userName, string password)
    {
      Console.Write("\nValidating username, {0}, and password, {1} ... ",
userName, password);
       if ((string.Compare(userName, "don", true) != 0) ||
          (string.Compare(password, "hall", false) != 0))
          {
           throw new SecurityTokenException("Unknown user.");
          }
       Console.Write("Done: Credentials accepted. \n");
     }
  }
}
```

Additional Resources

- For more information on password validators, see "User Name Password Validator" at http://msdn.microsoft.com/en-us/library/aa354513.aspx

Authorization

- **How to Authorize Declaratively**
- **How to Authorize Imperatively if You Use a Role Provider**
- **How to Authorize Imperatively**
- **How to Perform Resource-based Authorization**
- **How to Perform Role-based Authorization**
- **How to Authorize Users Against Windows Groups**
- **How to Authorize Users Against Windows Groups Using Aspnetwindowstokenroleprovider**
- **How to Authorize Users Against the SQL Server Role Provider**
- **How to Authorize Users Against the ASP.NET Role Provider**
- **How to Assign the Current Principal with Iauthorizationpolicy to Allow Authorization Using Custom Authentication**
- **How to Authorize Users Against ADAM Using the Authorization Manager Role Provider**
- **How to Map Roles to Certificates**

How to Authorize Declaratively

Declarative authorization can be added to application code at design time by specifying required access for a particular method or class declared as an attribute on the operation. Declarative role-based authorization is best for authorizing access to WCF at the operation

level. Declarative authorization can be added to application code at design time by specifying required access for a particular method or class declared as an attribute on the operation.

Authorize Windows groups declaratively by adding the **PrincipalPermission** attribute above each service method that requires authorization. Specify the Windows user group required to access the method in the Role field as shown in the following example:

```
[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]
public double Add(double a, double b)
{
    return a + b;
}
```

The username/password combination supplied by the client will be mapped to a Windows user account by the WCF service. If the user is successfully authorized, the system will next check to see if the user belongs to the group declared with the **PrinciplePermission** role. Method access will be granted if the user belongs to the role.

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For information on the **Roles.IsUserInRole** method, see “Roles.IsUserInRole Method (String)” at http://msdn.microsoft.com/en-us/library/4z6b5d42.aspx

How to Authorize Imperatively if You Use a Role Provider

If you are using a role provider, you can do imperative checks by calling **Roles.isUserInRole**. If you are using the AspNetWindowsToken Role Provider, you can use imperative authorization against Windows roles. Imperative security is useful when the resource to be accessed or action to be performed is not known until run time, or when you require finer-grained access control beyond the level of a code method.

Authorize Windows groups or roles that can be SQL or custom roles imperatively by using the **Roles.IsUserInRole** method to authorize the client. The role can be contained in a variable and changed dynamically if needed, as shown in the following example:

```
  string RequiredGroup = "Administrators";
  try
  {
    if (!Roles.IsUserInRole(User.Identity.Name, "RequiredGroup"))
    {
      Msg.Text = "You are not authorized to view user roles.";
      UsersListBox.Visible = false;
      return;
    }
  }
  catch (HttpException e)
  {
    Msg.Text = "There is no current logged on user. Role membership cannot be
verified.";
```

```
    return;
}
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For information on the **Roles.IsUserInRole** method, see “Roles.IsUserInRole Method (String)” at http://msdn.microsoft.com/en-us/library/4z6b5d42.aspx
- For authorization guidelines, see the Authorization section of “WCF 3.5 Security Guidelines“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Authorize Imperatively

Imperative role-based authorization is written into your code and processed at run time. Imperative security is useful when the resource to be accessed or action to be performed is not known until run time, or when you require finer-grained access control beyond the level of a code method.

Authorize Windows groups imperatively by using the **principal.InInRole** method. The following code snippet exemplifies the authorization check:

```
string RequiredGroup = "Administrators";
IPrincipal principal=System.Threading.Thread.CurrentPrincipal;
  if (principal.IsInRole(RequiredGroup))
      return string.Format("You entered: {0}", value);
  else
      return "not authorized";
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For information on the **Roles.IsUserInRole** method, see “Roles.IsUserInRole Method (String)” at http://msdn.microsoft.com/en-us/library/4z6b5d42.aspx
- For authorization guidelines, see the Authorization section of “WCF 3.5 Security Guidelines“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Perform Resource-based Authorization

Resource-based authorization sets permissions on the resource itself. For example, you would set an access control list (ACL)on a Windows resource and then use the identity of the original caller to determine access rights to the resource. If you use resource-based authorization in WCF, you will need to impersonate the original caller through the application layer (e.g.,

ASP.NET application), through the WCF service layer, and to the business logic code that is accessing the file resource.

To use resource-based authorization, you need to set permissions on the resource itself by setting an ACL and then impersonating the original caller.

The following code example impersonates a specific (fixed) identity:

```
using System.Security.Principal;
…
WindowsIdentity wi = new
WindowsIdentity("userName@fullyqualifieddomainName");
WindowsImpersonationContext ctx = null;

try
{
  ctx = wi.Impersonate();

  // Thread is now impersonating you can access resource needed…
}
catch
{
  // Prevent exceptions propagating.
}
finally
{
  // Ensure impersonation is reverted
  ctx.Undo();
}
```

Additional Resources

- For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Perform Role-based Authorization

Use role-based authorization to group users into groups (roles) and then set permissions on the role rather than on individual users. This eases management by allowing you to administer a smaller set of roles rather than a larger set of users.

The following are the different option for creating role-based authorization based on your authentication choice:

- If you are using Windows or Basic authentication, you can use Windows groups for role-based authorization.
- If you are using username authentication, you can use ASP.NET roles for role-based authorization.

- If you are using certificate authentication, you can map certificates to Windows groups for role-based authorization.

The following example configures the service to enable the SQL Server role provider for using ASP.NET roles:

1. Configure the SQL Server role provider:

```
<!-- Configure the Sql Role Provider -->
<roleManager enabled ="true"
             defaultProvider ="SqlRoleProvider" >
  <providers>
    <add name ="SqlRoleProvider"
         type="System.Web.Security.SqlRoleProvider"
         connectionStringName="SqlConn"
         applicationName="MembershipAndRoleProviderSample"/>
  </providers>
</roleManager>
<!-- Configure role based authorization to use the Role Provider -->
<serviceAuthorization principalPermissionMode ="UseAspNetRoles"
                      roleProviderName ="SqlRoleProvider" />
```

2. Include a **PrincipalPermission** attribute in the service method that specifies the required authorization access role:

```
    [PrincipalPermission(SecurityAction.Demand, Role = "Registered
Users")]
    public double Multiply(double n1, double n2)
      {
         double result = n1 * n2;
         return result;
      }
```

3. The following code shows how to create the authorization check in code:

```
if (Roles.IsUserInRole(@"accounting"))
{
//authorized
}
else
{
//authorization failed

}
```

4. The following client connection supplies a username and password to call the method:

```
      // Set credentials to Alice
      client.ClientCredentials.UserName.UserName = "Alice";
      client.ClientCredentials.UserName.Password = "ecilA-123";

      // Call the Add service operation.
      double value1 = 100.00D;
      double value2 = 15.99D;
```

```
        double result = client.Multiply(value1, value2);
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Authorize Users Against Windows Groups

Map Windows groups to WCF service methods by using the WCF **PrincipalPermission** attribute. Incoming client username credentials will be mapped to the associated Windows group. Service method access will be granted to a user only if they are a member of the group associated with the service method being called.

The following example demonstrates how the WCF service “Add” will only run for users belonging to the “CalculatorClients” Windows group.

```
// Only members of the CalculatorClients group can call this method.
[PrincipalPermission(SecurityAction.Demand, Role = "CalculatorClients")]
public double Add(double a, double b)
{
    return a + b;
}
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For authorization guidelines, see the Authorization section of “WCF 3.5 Security Guidelines“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Authorize Users Against Windows Groups Using AspNetWindowsTokenRoleProvider

If you use Windows groups for authorization, consider using the ASP.NET role provider with the **AspNetWindowsTokenRoleProvider** name. This allows you to separate the design of the authorization from the implementation inside your service. If you decide to change the role provider, it will not affect the code needed to perform the authorization. Also, when doing imperative checks, consider using the role manager API instead of performing authorization checks with **WindowsPrincipal.IsInrole**.

The following configuration example shows how to configure **AspNetWindowsTokenRoleProvider**:

1. Enable the role manager and configure it to use the default **AspNetWindowsTokenRoleProvider** as follows:

```
<system.web>
…
<roleManager enabled="true"
defaultProvider="AspNetWindowsTokenRoleProvider" />
…
</system.web>
```

2. Configure the service behavior to use ASPNetRoles and the role provider as follows:

```
<behaviors>
    <serviceBehaviors>
        <behavior name="BehaviorConfiguration">
            <serviceAuthorization
principalPermissionMode="UseAspNetRoles"
                roleProviderName=" AspNetWindowsTokenRoleProvider " />
            <serviceMetadata />
        </behavior>
    </serviceBehaviors>
</behaviors>
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For information on the **Roles.IsUserInRole** method, see “Roles.IsUserInRole Method (String)” at http://msdn.microsoft.com/en-us/library/4z6b5d42.aspx
- For authorization guidelines, see the Authorization section of “WCF 3.5 Security Guidelines“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Authorize Users Against the SQL Server Role Provider

If you store role information in SQL Server, configure your application to use **SqlRoleProvider** for authorization. The role provider allows you to load the roles for users without writing and maintaining custom code.

Perform the following steps to enable SQL Server role authorization to provide role-based authorization:

1. Enable the role provider as shown below and configure the connection string pointing to the role store in SQL Server:

```
…
<configuration>
…
<connectionStrings>
    <add name="MyLocalSQLServer"
```

```
         connectionString="Initial Catalog=aspnetdb;data
source=Sqlserver;Integrated Security=SSPI;"

<system.web>
<roleManager enabled="true" defaultProvider="MySqlRoleProvider" >
      <providers>
        <add name="MySqlRoleProvider"
             connectionStringName="MyLocalSQLServer"
             applicationName="MyAppName"
             type="System.Web.Security.SqlRoleProvider" />
      </providers>
    </roleManager>
<system.web>
```

2. Configure the service behavior. Set the **principalPermissionMode** attribute to **UseAspNetRoles** and the **roleProviderName** attribute to **MySqlRoleProvider**:

```
…
<system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="BehaviorConfiguration">
          <serviceAuthorization
principalPermissionMode="UseAspNetRoles"
            roleProviderName="MySqlRoleProvider" />
         </behavior>
      </serviceBehaviors>
    </behaviors>
<services>
    <service behaviorConfiguration=" BehaviorConfiguration "
name="MyService">
      <endpoint binding="wsHttpBinding" bindingConfiguration=""
        name="httpsendpoint" contract="IMyService2" />
     </service>
  </services>
  </system.serviceModel>
```

3. Authorize Windows groups declaratively by adding the **PrincipalPermission** attribute above each service method that requires authorization. Specify the Windows user group required to access the method in the Role field:

```
[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]
public double Add(double a, double b)
{
    return a + b;
}
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of “WCF 3.5 Security Questions and Answers“ at

http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Authorize Users Against the ASP.NET Role Provider

Using ASP.NET Role provider allows you to separate the design of the authorization from the implementation inside your service. If you decide to change the role provider, it will not affect the code needed to perform the authorization.

Declaratively authorize users with the ASP.NET role provider

Perform the following steps to declaratively authorize users with the ASP.NET role provider:

1. Configure the ASP.NET role provider in the service app.config or web.config file as follows:

```
<system.web>
  <!-- Configure the ASP.NET Role Provider -->
  <roleManager enabled ="true"
               defaultProvider ="MyRoleProvider" >
    <providers>
      <add name ="MyRoleProvider"
           type="System.Web.Security.<RoleProviderTobeUsed>"
           connectionStringName="ProviderConn"
           applicationName="MembershipAndRoleProviderSample"/>
    </providers>
  </roleManager>
</system.web>
```

2. Configure the WCF service to use the ASP.NET role:

```
<behaviors>
  <serviceBehaviors>
    <behavior name="CalculatorServiceBehavior">
       <!-- Configure role based authorization to use -->
       <!-- the Role Provider -->
      <serviceAuthorization
           principalPermissionMode ="UseAspNetRoles"
           roleProviderName ="MyRoleProvider" />
    </behavior>
  </serviceBehaviors>
</behaviors>
```

3. Authorize users declaratively by adding the **PrincipalPermission** attribute above each service method that requires authorization. Specify the user role required to access the method in the Role field.

```
[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]
public double Add(double a, double b)
{
    return a + b;
}
```

Imperatively authorize users with the ASP.NET role provider

Perform the following steps to imperatively authorize users with the ASP.NET role provider:

1. Configure the ASP.NET role provider in the service app.config or web.config file as follows:

```
<system.web>
  <!-- Configure the ASP.NET Role Provider -->
  <roleManager enabled ="true"
               defaultProvider ="MyRoleProvider" >
    <providers>
      <add name ="SqlRoleProvider"
           type="System.Web.Security.<RoleProviderToBeUsed>"
           connectionStringName="ProviderConn"
           applicationName="MembershipAndRoleProviderSample"/>
    </providers>
  </roleManager>
</system.web>
```

2. Configure the WCF Service to use the ASP.NET role provider:

```
<behaviors>
  <serviceBehaviors>
    <behavior name="CalculatorServiceBehavior">
      <!-- Configure role based authorization -->
      <!-- to use the Role Provider -->
      <serviceAuthorization principalPermissionMode ="UseAspNetRoles"
                            roleProviderName ="MyRoleProvider" />
    </behavior>
  </serviceBehaviors>
</behaviors>
```

3. Authorize users imperatively by using the **Roles.IsUserInRole** method. The role can be contained in a variable and changed dynamically if needed, as shown in the following example:

```
  string RequiredGroup = “Administrators”;
  try
  {
    if (!Roles.IsUserInRole(User.Identity.Name, “RequiredGroup”))
    {
      Msg.Text = "You are not authorized to view user roles.";
      UsersListBox.Visible = false;
      return;
    }
  }
  catch (HttpException e)
  {
    Msg.Text = "There is no current logged on user. Role membership
cannot be verified.";
    return;
}
```

Additional Resources

- For more information on the ASP.NET role provider, see “How to: Use the ASP.NET Role Provider with a Service” at http://msdn.microsoft.com/en-us/library/aa702542.aspx
- For information on the **Roles.IsUserInRole** method, see “Roles.IsUserInRole Method (String)” at http://msdn.microsoft.com/en-us/library/4z6b5d42.aspx
- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Assign the Current Principal with IAuthorizationPolicy to Allow Authorization Using Custom Authentication

If your application uses custom authentication, you will need to create a class that derives from **IAuthorizationPolicy**. In this class, you will retrieve the principal from the cache that was created by the custom authentication, or from the store based on the username, so that WCF can authorize the user. After you get the principal, you assign it to EvaluationContext.Properties[“principal”] and the identity to EvaluationContext.Properties["Identities"] as shown in the following example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IdentityModel.Claims;
using System.IdentityModel.Policy;
using System.Web;
using System.Security.Principal;
using CustomAuthenticator;



namespace AuthorizationPolicy
{
   public class AuthorizationPrincipalPolicy : IAuthorizationPolicy
   {
      public bool Evaluate(EvaluationContext evaluationContext, ref
        object state)
        {
            object obj;
            if (!evaluationContext.Properties.TryGetValue("Identities",
                out obj))
                return false;
            IList<IIdentity> identities = obj as IList<IIdentity>;

            // make sure there is already a default identity
            if (identities == null || identities.Count <= 0)
                return false;
```

```
            string username = identities[0].Name;

            //get the principal from the cache or build another one

            IPrincipal principal =
            UserNameAuthenticator.GetUser(username);

            if (principal == null)
            {
                string[] roles =
                UserNameAuthenticator.GetRolesForUser(username);
                principal = new GenericPrincipal(new
                GenericIdentity(username, "Custom Provider"), roles);

            }

            evaluationContext.Properties["Principal"] = principal;
            evaluationContext.Properties["Identities"] =
                   new List<IIdentity>() { principal.Identity };


            return true;
        }

        public System.IdentityModel.Claims.ClaimSet Issuer
        {
            get { return ClaimSet.System; }
        }

        public string Id
        {
            get { return "ContextPrincipalPolicy"; }
        }
    }
}
```

The Policy library is configured in the web.config or app.config configuration file or in code. The following example configures the policy location in the configuration file. Define the custom authorization policy type in the **add** element **policyType** attribute.

```
<serviceAuthorization
    serviceAuthorizationManagerType
    ="Microsoft.ServiceModel.Samples.MyServiceAuthorizationManager, service">
    <!-- The serviceAuthorization behavior allows one to specify custom
authorization policies. -->
      <authorizationPolicies>

        <add policyType
="Microsoft.ServiceModel.Samples.CustomAuthorizationPolicy.MyAuthorizationPol
icy, PolicyLibrary" />
      </authorizationPolicies>

</serviceAuthorization>
```

Additional Resources

- For more information on custom authorization policies, see "How to: Create a Custom Authorization Policy" at http://msdn.microsoft.com/en-us/library/ms729794(VS.85).aspx
- For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx

How to Authorize Users Against ADAM Using the Authorization Manager Role Provider

If your application stores role information in an Authorization Manager (AzMan) policy store in Active Directory Application Mode (ADAM), use the Authorization Manager Role Provider. Authorization Manager provides a Microsoft Management Console (MMC) snap-in, to create and manage roles, and to manage role membership for users.

Perform the following steps to authenticate a directory service with ADAM:

1. Use AzMan to store roles in an ADAM policy store as follows.
   Note: You can only currently create an ADAM store only within Microsoft Windows Server® 2003.
   a. At the command prompt, type **azman.msc** to open the Authorization Manager snap-in.
   b. In AzMan, right-click **Authorization Manager** and then click **New Authorization Store**. Select **Active Directory** and enter a name to create the ADAM store.
   c. Right-click the Groups folder of the Active Directory store you just created, and then click **New Application Group**. Enter a name for the group you want to create. Repeat this step to create as many groups as needed.
   d. Add Windows users to the AzMan groups(s) you created. Double-click each group you created and use the **Members** tab to add the users.
2. Configure the web.config or app.config file to use the ADAM store. Define a connection string to the AzMan policy store in ADAM and configure the WCF service to use the role provider as follows.

```
<ConnectionStrings>

   <add name="AzManADAMServer" connectionString=
"msldap://servername:port/CN=AzManADAMStore,OU=SecNetPartition,O=Sec
Net,C=US"/>
</ConnectionStrings>

<system.web>
<roleManager
    enabled="true"
    defaultProvider="RoleManagerAzManADAMProvider"
    <providers>
        <add name="RoleManagerAzManADAMProvider"
     type="System.Web.Security.AuthorizationStoreRoleProvider,
System.Web, Version=2.0.0.0, Culture=neutral,
```

```
                publicKeyToken=b03f5f7f11d50a3a"
                connectionStringName="AzManADAMServer"
                applicationName="AzManDemo"/>
        </providers>
    </roleManager>
    </system.web>
```

3. Configure the service behavior. Set the **principalPermissionMode** attribute to **UseAspNetRoles** and the **roleProviderName** attribute to **RoleManagerAzManADAMProvider**:

```
…
<system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="BehaviorConfiguration">
          <serviceAuthorization
principalPermissionMode="UseAspNetRoles"
            roleProviderName="RoleManagerAzManADAMProvider" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
….
  </system.serviceModel>
```

4. Authenticate the users declaratively by adding the **PrincipalPermission** attribute above each service method that requires authorization. Specify the Windows user group required to access the method in the Role field.

```
[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]
public double Add(double a, double b)
{
   return a + b;
}
```

The username/password combination supplied by the client will be mapped by the WCF service to a Windows user account. If the user is successfully authorized, the system will next check to see if the user belongs to the group declared with the **PrinciplePermission** role. Method access will be granted if the user belongs to the role.

Additional Resources

- For more information on AzMan, see “How To: Use Authorization Manager (AzMan) with ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998336.aspx
- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For an authorization Q&A, see the Authorization section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Map Roles to Certificates

If you are using certificate authentication, you can map the certificate to a Windows account and authorize based on this account.

Perform the following steps to associate roles with a certificate:

1. Configure IIS to enable client certificate mapping.
    a. Open the IIS service manager.
    b. Right-click the Web site you will be using for your service and then click **Properties**.
    c. Click the **Directory Security** tab, and then in the **Secure Communications** section, click **Edit**.
    d. Select the **Enable Client Certificate Mapping** check box. Click the **Edit** button and then enter **1-1** or **Many-to-1**, depending on your configuration.
2. Configure the service to require **ClientCredentialType = "Certificate"**.
   This will require clients to connect by using certificate authentication.

```
<message clientCredentialType="Certificate" />
```

3. Configure the service to map certificates to user accounts in the web.config or app.config file. Set the **mapClientCertificateToWindowsAccount** to **"true"** as follows:

```
<serviceBehaviors>
 <behavior name="MappingBehavior">
  <serviceCredentials>
   <clientCertificate>
    <authentication certificateValidationMode="None"
                    mapClientCertificateToWindowsAccount="true" />
   </clientCertificate>
  </serviceCredentials>
 </behavior>
</serviceBehaviors>
```

4. Configure clients to supply a certificate as shown below.
   The incoming client requests will contain a certificate name and thumbprint ID. IIS will map the client certificates to a Windows user account.

```
<message clientCredentialType="Certificate" />
```

5. Authorize the required Windows group by adding the **PrincipalPermission** attribute above each service method that requires authorization. Specify the Windows user group required to access the method in the Role field as shown in the following example.

```
[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]
public double Add(double a, double b)
{
    return a + b;
}
```

Additional Resources

- For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For authorization guidelines, see the Authorization section of "WCF 3.5 Security Guidelines" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

Configuration Management

- **How to Encrypt Sensitive Data in Your Configuration Files**
- **How to Run Your Service Under a Specific Identity**
- **How to Create a Service Account for Your WCF Service**
- **How to Stop Clients from Referencing Your Service**
- **How to Protect Against Message Replay Attacks**

How to Encrypt Sensitive Data in Your Configuration Files

Encrypt configuration sections that contain sensitive data such as SQL connection strings. Use DPAPI to encrypt the sensitive data in the configuration file on your WCF server machine. To encrypt sensitive data in your configuration files, use the aspnet_regiis.exe tool with the **-pe** (provider encryption) option.

For example, to encrypt the **connectionStrings** section**,** using the DPAPI provider with the machine key store (the default configuration), run the following command from a command prompt:

```
aspnet_regiis -pe "connectionStrings" -app "/MachineDPAPI" -prov
"DataProtectionConfigurationProvider"
```

The aspnet_regiis settings are:

- **-pe** – specifies the configuration section to encrypt.
- **-app** – specifies your Web application's virtual path. If your application is nested, you need to specify the nested path from the root directory; for example, "/test/aspnet/MachineDPAPI"
- **-prov** – specifies the provider name.

The Microsoft .NET Framework supports the following protected configuration providers:

- **RSAProtectedConfigurationProvider.** This is the default provider. It uses the RSA public key encryption to encrypt and decrypt data. Use this provider to encrypt configuration files for use on multiple WCF services in a Web farm.
- **DPAPIProtectedConfigurationProvider.** This provider uses the Windows Data Protection API (DPAPI) to encrypt and decrypt data. Use this provider to encrypt configuration files for use on a single Windows Server.

You do not need to take any special steps for decryption because the .NET run time takes care of this for you.

Additional Resources

- For more information on encrypting configuration sections, see "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI" at http://msdn2.microsoft.com/en-us/library/ms998280.aspx and "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA" at http://msdn2.microsoft.com/en-us/library/ms998283.aspx
- For more information on the aspnet_regiis tool, see "ASP.NET IIS Registration Tool (Aspnet_regiis.exe)" at http://msdn.microsoft.com/en-us/library/k6h9cz8h(VS.80).aspx

How to Run Your Service Under a Specific Identity

Running your WCF service with a specific identity helps to isolate the service, allows you to restrict service resources to your application's account, and allows you to use Windows auditing to track the application's activity separately from other applications or services.

If your service is hosted in IIS 6.0, use IIS Manager to create an application pool running as a specific identity. Use IIS Manager to assign your WCF service to that application pool. This would enable your WCF service to run under the security context of the specific identity.

If your service is hosted in a Windows service, configure the Windows service to run using the specific identity. This would enable the WCF service to run under the security context of the specific identity.

Additional Resources

- For information on IIS 6.0 management, see the "Server Administration Guide (IIS 6.0)" at http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/33e0a51a-5f8a-40f2-9923-cdd604e1a812.mspx
- For configuration management guidelines, see the Configuration Management section of "WCF 3.5 Security Guidelines" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Create a Service Account for Your WCF Service

Running your WCF service with a specific identity, such as a service account, helps to isolate the service. It allows you to restrict service resources to your application's account, and allows you to use Windows auditing to track the application's activity separately from other applications or services.

Perform the following steps to create a service account to run your WCF service:

1. Create a Windows account.

2. Run the following aspnet_regiis.exe command to assign the relevant ASP.NET permissions to the account:

   ```
   aspnet_regiis.exe -ga machineName\userName
   ```

   Note: This step is required when your application needs to run in ASP.NET compatibility mode; otherwise you can skip this step.

3. Use the Local Security Policy tool to grant the Windows account the **Deny logon locally** user right.
   This reduces the privileges of the account and prevents anyone from logging on to Windows locally with this account.

Additional Resources

- For more information on the aspnet_regiis tool, see "ASP.NET IIS Registration Tool (Aspnet_regiis.exe)" at http://msdn.microsoft.com/en-us/library/k6h9cz8h(VS.80).aspx

How to Stop Clients from Referencing Your Service

If you want to block clients from accessing the Web Services Description Language (WSDL) of your service, you should remove all metadata exchange (mex) endpoints and set the **httpGetEnabled** and **httpsGetEnabled** attributes to false. If the metadata is exposed, unwanted clients will be able to generate proxy files (e.g., using SvcUtil.exe) and inspect potentially sensitive methods and parameters offered by the service.

To stop your clients from referencing your service, stop your service from publishing its metadata. To do this, remove all the mex endpoints from your service configuration and configure **HttpGetEnabled** and **HttpsGetEnabled** to false in the **ServiceBehavior** section as shown below:

```
<serviceMetadata httpGetEnabled="False" httpsGetEnabled="False"/>
```

Additional Resources

- For more information on publishing metadata endpoints, see "Publishing Metadata Endpoints" at http://msdn.microsoft.com/en-us/library/ms788760.aspx

How to Protect Against Message Replay Attacks

A *replay attack* occurs when an attacker copies a stream of messages between two parties and replays the stream to one or more of the parties. To protect against message replay attacks, enable replay detection in the service.

Perform the following steps to enable replay detection:

1. Create a customBinding Element.
2. Create a <security> element.
3. Create a localClientSettings element or localServiceSettings element.

4. Set the following attribute values, as appropriate: **detectReplays**, **maxClockSkew**, **replayWindow**, and **replayCacheSize**.

The following example sets the attributes of both a <localServiceSettings> and a <localClientSettings> element:

```
<customBinding>
  <binding name="NewBinding0">
   <textMessageEncoding />
    <security>
     <localClientSettings
      replayCacheSize="800000"
      maxClockSkew="00:03:00"
      replayWindow="00:03:00" />
     <localServiceSettings
      replayCacheSize="800000"
      maxClockSkew="00:03:00"
      replayWindow="00:03:00" />
    <secureConversationBootstrap />
   </security>
  <httpTransport />
 </binding>
</customBinding>
```

Additional Resources

- For more information on replay detection, see "How to: Enable Message Replay Detection" at http://msdn2.microsoft.com/en-us/library/ms733063.aspx
- For a configuration management Q&A, see the Configuration Management section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

Deployment Considerations

- **How to Configure Certificates to Enable SSL In IIS**
- **How to Map Windows Accounts with Certificates**
- **How to Create a Service Principle Name (SPN)**
- **How to Configure WCF for NATs and Firewalls**
- **How to Create an X.509 Certificate**

How to Configure Certificates to Enable SSL in IIS

Use Secure Sockets Layer (SSL) in Internet Information Services (IIS) to protect the communication channel between your WCF-enabled Web application and the Web client. SSL protects sensitive data on the network from being stolen or modified.

Perform the following steps to configure certificates for SSL communication in IIS.

1. Click **Start** and then click **Run**.
2. In the **Run** dialog box, type **inetmgr** and then click **OK**.

3. In the **Internet Information Services (IIS) Manager** dialog box, expand the **(local computer)** node, and then expand the **Web Sites** node.
4. Right-click **Default Web Site** and then click **Properties**.
5. In the **Default Web Site Properties** dialog box, click the **Directory Security** tab, and then in the **Secure Communications** section, click **Server Certificate**.
6. On the Welcome screen of the Web Server Certificate Wizard, click **Next** to continue.
7. On the Server Certificate screen, select the **Assign an existing certificate** radio button option, and then click **Next**.
8. On the Available Certificates screen, select the certificate you created and installed in the previous step, and then click **Next**.
9. Verify the information on the certificate summary screen, and then click **Next.**
10. Click **Finish** to complete the certificate installation.
11. In the **Default Web Site Properties** dialog box, click **OK**.

Additional Resources

- For information on installing a server certificate, see "Install a Server Certificate (IIS 6.0)" at http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/a2f35fcd-d3b6-4f39-ba93-041a86f7e17f.mspx?mfr=true
- For a Q&A on deployment considerations, see the Deployment Considerations section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home
- For deployment guidelines, see the Deployment Considerations section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Security%20Guidelines&referringTitle=Home

How to Map Windows Accounts with Certificates

If you are using certificate authentication, you can map certificates to Windows accounts in order to enable authentication and authorization based on the Windows account.

You can map an X.509 certificate to a Windows account by setting the **mapClientCertificateToWindowsAccount** property to true. By default, when using the certificate client credential type on bindings, the certificate is not mapped to Windows accounts.

Perform the following steps to map certificates to Windows accounts:

1. Decide between the IIS certificate mapping versus Active Directory certificate mapping.
    a. IIS mapping is useful if you need only a limited number of mappings, or a different mapping on each WCF service.

   b. Use Active Directory mapping when the account mappings are identical on all IIS servers. Active Directory mapping is easier to maintain than IIS mapping because you only have to create the mapping in one location.
2. Configure IIS / Active Directory for mapping the certificates.
3. After you have enabled the client certificate mapping feature, set the **mapClientCertificateToWindowsAccount** property to true as follows:

```
<serviceBehaviors>
  <behavior name="MyServiceBehaviorForWebHttp">

     <serviceCredentials>
      <clientCertificate>
       <authentication mapClientCertificateToWindowsAccount="true" />
      </clientCertificate>
     </serviceCredentials>

  </behavior>
</serviceBehaviors>
```

Additional Resources

- For more information on certificates, see “Working with Certificates” at http://msdn.microsoft.com/en-us/library/ms731899.aspx
- For more information on mapping certificates to Windows accounts, see “Map certificates to user accounts” at http://technet2.microsoft.com/WindowsServer/f/?en/library/0539dcf5-82c5-48e6-be8a-57bca16c7e171033.mspx
- For more information on mapping certificates to Active Directory, see “Mapping Client Certificates with Directory Service Mapping” at http://technet2.microsoft.com/windowsserver/en/library/7cce4299-28f2-45fa-8730-4e0cbe3be8561033.mspx?mfr=true
- For more information on certificate mapping strategies, see “Mapping Strategies” at http://technet2.microsoft.com/windowsserver/en/library/aa61c564-1599-4414-a12d-2f64786f6ec31033.mspx?mfr=true

How to Create a Service Principle Name (SPN)

A service principal name (SPN) is the name by which a client uniquely identifies an instance of a service. The Kerberos authentication service can use an SPN to authenticate a service. When a client wants to connect to a service, it locates an instance of the service, composes an SPN for that instance, connects to the service, and then presents the SPN for the service to authenticate.

To create an SPN for a domain account, run the Setspn tool from a command prompt as shown below:

```
setspn -A HTTP/webservername domain\customAccountName
setspn -A HTTP/webservername.fullyqualifieddomainname
domain\customAccountName
```

The setspn tool creates an SPN for the custom domain account (domain\customAccountName) and associates the account with the HTTP service on the specified Web server. By running the command twice as shown above, you can associate the account with the NetBIOS server name and the fully qualified domain name (FQDN) of the server. This ensures that the SPN is established correctly even if your environment does not consistently use FQDNs.

Additional Resources

- For more information on SPN, see "Setspn.exe: Manipulate Service Principal Names for Accounts" at http://technet2.microsoft.com/windowsserver/en/library/b3a029a1-7ff0-4f6f-87d2-f2e70294a5761033.mspx?mfr=true

How to Configure WCF for NATs and Firewalls

Network address translators (NATs) and firewalls can impact the strategy by which your WCF clients and services communicate.

Perform the following steps to determine WCF configuration for a NAT or firewall:

1. Determine the addressability of the service and client machines. If the service or the client is behind a NAT and is not directly addressable, use a technology such as Microsoft Teredo to enable communication.
2. Determine if there are protocol or port constraints on the service or client machines. For example, port 80 might be open through a firewall while other ports might be blocked.

Once you understand the addressability, protocol, and port constraints on your service and its clients, you can determine service and endpoint configuration. Use the table in the MSDN® article "Working with NATS and Firewalls" at http://msdn.microsoft.com/en-us/library/ms731948.aspx to determine the best configuration for your particular scenario.

Additional Resources

- For more information on Microsoft Teredo, see "Teredo Overview" at http://technet.microsoft.com/en-us/library/bb457011.aspx
- For more information on configuring WCF for NATs and firewalls, see "Working with NATS and Firewalls" at http://msdn.microsoft.com/en-us/library/ms731948.aspx

How to Create an X.509 Certificate

You might need to create an X.509 certificate to aid in development and debugging of your WCF service. Temporary certificates are easier to generate and cost less money than a certificate issued by a trusted certificate authority (CA), so they are well suited for a development environment.

To create a temporary X.509 certificate in a development environment use the Makecert utility. In a production environment, use an X.509 certificate issued by a CA such as VeriSign.

Note: Do not use temporary development certificates in a production environment because this will open your communication channel to malicious spoofing, sniffing, and tampering.

Additional Resources

- For more information, see "How To – Create and Install Temporary Certificates in WCF for Message Security During Development" and "How To – Create and Install Temporary Certificates in WCF for Transport Security During Development" at http://www.codeplex.com/WCFSecurityGuide - Scroll to the How To's section.
- For more information on creating a certificate, see "Certificate Creation Tool" at http://msdn2.microsoft.com/en-us/library/bfsktky3(VS.80).aspx

Exception Management

- **How to Shield Exception Information with Fault Contracts**
- **How to Check the State of a Channel in WCF Proxy Client**
- **How to Avoid Faulting the Channels with Fault Contracts**
- **How to Create an Error Handler to Log Details of Faults for Auditing Purposes**
- **How to Handle Unhandled Exceptions in Downstream Services**
- **How to Throw an Exception with Complex Types or Data Contracts with a Fault Exception**
- **How to Handle Unknown Faults in a Service**
- **How to Implement a Data Contract to Propagate Exception Details for Debugging Purposes**
- **How to Implement Fault Contracts in Callback Functions**

How to Shield Exception Information with Fault Contracts

A *fault contract* details the set of exceptions that can be reported to the caller. You can specify the possible faults that can occur in your WCF service. This prevents exposing exception details beyond the defined set to your clients. Because a fault contract lists the types of errors that a WCF service can throw, it also allows your clients to distinguish between contracted faults and other possible errors.

Perform the following steps to shield exception information with fault contracts:

1. Define a fault contract by applying the **FaultContract** attribute directly on a contract operation, and specifying the error detailing type to the method as follows:

```
[ServiceContract]
interface ICalculator
{
   [OperationContract]
   [FaultContract(typeof(DivideByZeroException))]
   double Divide(double number1,double number2);
}
```

2. Implement the **Divide** operation that throws the fault and have it propagated to the client by throwing exactly the same detailing type listed in the fault contract.

```
class MyService : ICalculator
{
   public double Divide(double number1,double number2)
   {
      throw new FaultException<DivideByZeroException>(new
DivideByZeroException());
   }
}
```

Additional Resources

- For more information on fault contracts, see "Specifying and Handling Faults in Contracts and Services" at http://msdn.microsoft.com/en-us/library/ms733721.aspx

How to Check the State of a Channel in WCF Proxy Client

You can check the state of a channel during proxy invocation by checking its value (proxy.State). This allows you to avoid throwing an exception on the proxy channel as a result of the channel being closed due to an unhandled exception on the service. Following code examples shows you how to check the state of the channel.

```
try
{
     ServiceClient proxy = new ServiceClient();
     proxy.ClientCredentials.UserName.UserName = "user";
     proxy.ClientCredentials.UserName.Password = "password";
     proxy.GetData(2);
     if (proxy.State = CommunicationState.Opened)
     {
         proxy.GetData("data");
     }
     proxy.Close();
}
catch (FaultException ex)
{
    // handle the exception
}
```

How to Avoid Faulting the Channels with Fault Contracts

When calling a WCF service, if the service throws any exceptions, the communication channel goes into the faulted state and you cannot use the proxy for any further calls. You can avoid this by throwing a **FaultException** in your service operations. The service operation that throws a **FaultException** must be decorated with one or more **FaultContract** attributes.

Perform the following steps to throw a **FaultException**:

1. Define a fault contract by applying the **FaultContract** attribute directly on a contract operation and specifying the error detailing type to the method, as shown below:

```
[ServiceContract]
interface ICalculator
{
   [OperationContract]
```

```
   [FaultContract(typeof(DivideByZeroException))]
   double Divide(double number1,double number2);
}
```

2. Implement the **Divide** operation that throws the fault and have it propagated to the client by throwing exactly the same detailing type listed in the fault contract:

```
class MyService : ICalculator
{
   public double Divide(double number1,double number2)
   {
      throw new FaultException<DivideByZeroException>(new
DivideByZeroException());
   }
}
```

3. Handle the faults at the client side by catching the **FaultException** and any other communication exceptions that could occur when calling the service operations:

```
try
{
  proxy.Divide();
}
catch (FaultException<DivideByZeroException> ex)
{
  // only if a fault contract of type DivideByZeroException was
specified
}

catch (FaultException ex)
{
  // any other faults
}

catch (CommunicationException ex)
{
  // any communication errors?
}
```

Additional Resources

- For more information on fault contracts, see “Specifying and Handling Faults in Contracts and Services” at http://msdn.microsoft.com/en-us/library/ms733721.aspx

How to Create an Error Handler to Log Details of Faults for Auditing Purposes

You can create an error handler to log fault details by implementing the **IErrorHandler** interface methods in your service. This allows you to log and suppress the exceptions, or to log and throw them as a **FaultException**. Following code sample shows the methods of the **IErrorHandler** interface.

```
public interface IErrorHandler
```

```
{
        bool HandleError(Exception error, MessageFault fault);
        void ProvideFault(Exception error, ref MessageFault fault, ref string
faultAction);
}
```

To suppress the fault message, implement the **HandleError** method and return **false**. In this method, you can add your code for logging capabilities.

To raise a **FaultException** instead of suppressing the fault, implement the **ProvideFault** method to provide the **MessageFault** value. The following code shows a sample implementation of the **ProvideFault** method:

```
public void ProvideFault(Exception error, MessageVersion version, ref Message
fault)
{
  FaultException newEx = new FaultException();
  MessageFault msgFault = newEx.CreateMessageFault();
  fault = Message.CreateMessage(version, msgFault, newEx.Action);
}
```

Additional Resources

- For more information on fault contracts, see "Specifying and Handling Faults in Contracts and Services" at http://msdn.microsoft.com/en-us/library/ms733721.aspx

How to Handle Unhandled Exceptions In Downstream Services

Use a global exception handler to catch unhandled exceptions and prevent them from being propagated to the client.

You can handle the unhandled exceptions in a WCF service by subscribing to the **Faulted** event of a service host object. By subscribing to this event, you can determine the cause of a failure and then perform the necessary actions to abort or restart the service.

The following code snippet shows how to subscribe to the **Faulted** event:

```
// hosting a WCF service
ServiceHost customerServiceHost;
customerServiceHost = new ServiceHost(…);
…
// Subscribe to the Faulted event of the customerServiceHost object
customerServiceHost.Faulted += new EventHandler(faultHandler);
…
// FaultHandler method - invoked when customerServiceHost enters the Faulted
state
void faultHandler(object sender, EventArgs e)
{     // log the reasons for the fault…
}
```

Additional Resources

- For more information on fault contracts, see "Specifying and Handling Faults in Contracts and Services" at http://msdn.microsoft.com/en-us/library/ms733721.aspx

How to Throw an Exception with Complex Types or Data Contracts with a Fault Exception

The following steps show an example of how to throw an exception with a data contract that has a complex type:

1. Define the **DataContract** type to pass the details of Simple Object Access Protocol (SOAP) faults as exceptions from a service back to a client:

```
[DataContract]
public class DatabaseFault
{
[DataMember]
public string DbOperation;
[DataMember]
public string DbReason
[DataMember]
public string DbMessage;
}
```

2. Use the **FaultContract** attribute in the **ListCustomers** method to generate SOAP faults as follows:

```
[ServiceContract]
public interface ICustomerService
{
// Get the list of customers
[FaultContract(typeof(DatabaseFault))]
[OperationContract]
List<string> ListCustomers();
…
}
```

3. Create and populate the **DatabaseFault** object with the details of the exception in the service implementation class, and then throw a **FaultException** object with the **DatabaseFault** object details as follows:

```
catch(Exception e)
{    DatabaseFault df = new DatabaseFault();
df.DbOperation = "ExecuteReader";
df.DbReason = "Exception in querying the Northwind database.";
df.DbMessage = e.Message;
throw new FaultException<DatabaseFault>(df);
}
```

Additional Resources

- For more information on fault contracts, see "Specifying and Handling Faults in Contracts and Services" at http://msdn.microsoft.com/en-us/library/ms733721.aspx

How to Handle Unknown Faults in a Service

To handle unknown faults in a service, throw an instance of **FaultException** directly. Any **FaultException<T>** thrown by the service always reaches the client as a **FaultException<T>** or as **FaultException**. The **FaultException<T>** class is derived from the **FaultException** class.

```
throw new FaultException("Specify some reason");
```

Additional Resources

- For more information on fault contracts, see "Specifying and Handling Faults in Contracts and Services" at http://msdn.microsoft.com/en-us/library/ms733721.aspx

How to Implement a Data Contract to Propagate Exception Details for Debugging Purposes

Perform the following steps to implement a data contract to propagate exception details for debugging purposes:

1. Create a **DataContract**, with a member variable for storing the fault reason:

```
[DataContract]
   public class MyDCFaultException
   {
       private string _reason;

       [DataMember]
       public string Reason
       {
           get { return _reason; }
           set { _reason = value; }
       }
   }
```

2. Create a service contract, specifying a **FaultContract** with the above **DataContract** type for an operation:

```
[ServiceContract()]
   public interface IService
   {
       [OperationContract]
       [FaultContract(typeof(MyDCFaultException))]
       string DoSomeComplexWork();
   }
```

3. Implement the service operation. If there any errors occur, send the exception details by throwing a **FaultException** of type **MyDCFaultException** as follows:

```
public class Service : IService
{

  public string DoSomeComplexWork()
  {
   try
    {
      // some complex operations
    }
    catch (Exception exp)
    {
      MyDCFaultException theFault = new MyDCFaultException();
      theFault.Reason = "Some Error " + exp.Message.ToString();
      throw new FaultException<MyDCFaultException>(theFault);
    }
      return "No Error";

 }
```

4. Call the service operation from a client application. Get the original service exception reason by using the **Detail.Reason** property as follows:

```
try
 {
  localhost.IService proxy = new localhost.ServiceClient();
  result = proxy.DoSomeComplexWork();
 }

catch (FaultException<localhost.MyDCFaultException> ex)
 {
   result = "Exception: " + ex.Detail.Reason;
 }

 Console.WriteLine(result);
 Console.ReadLine();
```

You can also send managed exception information to the clients by using the **IncludeExceptionDetailInFaults** property in the **serviceDebug** element of your service behavior as shown below. By default, its value is false; you can change it to true for debugging or problem diagnosis.

```
<system.serviceModel>
    <services>
        <service name="MyService"
                        behaviorConfiguration="MyServiceBehavior">
        </service>
    </services>
    <behaviors>
        <serviceBehaviors>
            <behavior name="MyServiceBehavior">
                <serviceDebug includeExceptionDetailInFaults="true"/>
            </behavior>
        </serviceBehaviors>
    </behaviors>
</system.serviceModel>
```

Additional Resources

- For more information on fault contracts, see "Specifying and Handling Faults in Contracts and Services" at http://msdn.microsoft.com/en-us/library/ms733721.aspx

How to Implement Fault Contracts in Callback Functions

In duplex bindings, you need to implement the service with a callback contract specifying the interface so that the client can implement it. Use the **CallbackContract** attribute in the service contract to specify the type of callback contract that has the callback function.

The following code example shows a service that specifies a callback contract:

```
[ServiceContract(CallbackContract = typeof(IMyContractCallback))]
interface IMyContract
{
   [OperationContract]
   void DoSomething( );
}
interface IMyContractCallback
{
   [OperationContract]
   [FaultContract(typeof(InvalidOperationException))]
   void OnCallBack( );
}
```

Additional Resources

- For more information on fault contracts, see "Specifying and Handling Faults in Contracts and Services" at http://msdn.microsoft.com/en-us/library/ms733721.aspx

Hosting

- **How to Host WCF in IIS**
- **How to Host WCF in a Windows Service**
- **How to Self-host WCF**
- **How to Configure a Least-privileged Account to Host Your Service**

How to Host WCF in IIS

Use IIS to host your WCF service, unless you need to use a transport that IIS does not support. IIS provides a large number of features for efficient service management and scalability. By using IIS as your WCF service host, you can take full advantage of IIS features, such as process recycling, idle shutdown, process health monitoring, and message-based activation.

HTTP Bindings can be hosted in IIS 6.0 and IIS 7.0. You can host TCP and MSMQ bindings in IIS 7.0 or a Windows service. You can also host in IIS 6.0, but you must first activate the host W3wp process before using the service.

Perform the following high-level steps to host your WCF service in IIS:

1. Create a virtual directory in IIS.
2. Create a .svc file for the WCF service.
3. Deploy the WCF service implementation to the IIS virtual directory.
4. Configure the WF service.

Additional Resources

- For more information on hosting in IIS, see "Hosting in Internet Information Services" at http://msdn.microsoft.com/en-us/library/ms734710.aspx
- For more information, see "Deploying an Internet Information Services-Hosted WCF Service" at http://msdn.microsoft.com/en-us/library/aa751792.aspx
- For more information on hosting a WCF service in IIS, see "How to: Host a WCF Service in IIS" at http://msdn.microsoft.com/en-us/library/ms733766.aspx

How to Host WCF in a Windows Service

You should use a Windows service when you have to support transports such as TCP, MSMQ, or named pipes. Windows services have advantages over self-hosting in that they give the benefit of automatic startup, the service lifetime is controlled by the operating system, it is easier to run under a least-privileged account, and the Windows service host will restart your service if it fails. Windows services can be managed by using the Service Control Manager in the Microsoft Management Console (MMC).

Perform the following steps to host your WCF service in a Windows service:

1. Create a Windows Service Project using Visual Studio 2008.
2. Add service installers to the Windows Service Project.
3. Override the **OnStart** and **OnStop** methods to start and stop the service inside the Windows service, as shown in the following code example:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Linq;
using System.ServiceProcess;
using System.Text;
using System.ServiceModel;

namespace WindowsService1
{
    public partial class WCFServiceHost1 : ServiceBase
    {
        internal static ServiceHost myServiceHost = null;

        public WCFServiceHost1()
        {
            InitializeComponent();
        }
        protected override void OnStart(string[] args)
```

```
        {
            if (myServiceHost != null)
            {
                myServiceHost.Close();
            }
            myServiceHost = new ServiceHost(typeof(Service1));
            myServiceHost.Open();
        }
        protected override void OnStop()
        {
            if (myServiceHost != null)
            {
                myServiceHost.Close();
                myServiceHost = null;
            }
        }
    }
}
```

4. Install the Windows service by using the InstallUtil.exe command from the Visual Studio 2008 command prompt.

Additional Resources

- For more information, see "How To – Host WCF in a Windows Service Using TCP" contained in this Guide or at http://www.codeplex.com/WCFSecurityGuide/Wiki/Print.aspx?title=How%20To%20-%20Host%20WCF%20in%20a%20Windows%20Service%20Using%20TCP&version=1&action=Print
- For more information, see "How to: Host a WCF Service in a Managed Windows Service" at http://msdn.microsoft.com/en-us/library/ms733069.aspx

How to Self-host WCF

Self-hosting is best suited for development and debugging scenarios in which you want maximum flexibility and you want to get the service running as quickly as possible. When readying for deployment, you should choose between hosting in a Windows service or in IIS.

Use the following methods to self-host your WCF service in any .NET application:

1. Create a method to start the service, as shown in the following code example:

```
// Host the service within the application.
public static void Main()
{
    // Create a ServiceHost for the CalculatorService type.
    using (ServiceHost serviceHost =
           new ServiceHost(typeof(Service1)))
    {
        // Open the ServiceHost to create listeners
        // and start listening for messages.
        serviceHost.Open();
```

```
            Console.ReadLine();
        }
    }
```

2. In the self-hosted case, you must specify the base address. The following example shows how to configure the configuration file:

```
<service
    name="Service1"
    behaviorConfiguration="ServiceBehavior">
  <host>
    <baseAddresses>
      <add
baseAddress="http://localhost:8000/WCFSecuritySamples/service"/>
    </baseAddresses>
  </host>
  ...
</service>
```

Additional Resources

- For more information on self-hosting, see "Self-Host" at http://msdn.microsoft.com/en-us/library/ms750530.aspx
- For more information on hosting a WCF service in a managed application, see "How to: Host a WCF Service in a Managed Application" at http://msdn.microsoft.com/en-us/library/ms731758.aspx

How to Configure a Least-privileged Account to Host Your Service

Use a least-privileged account to host your service in order to reduce your application's overall attack surface and reduce the potential impact of security vulnerabilities in your service. Using a least-privileged account allows you to audit and authorize your services individually. Your service is also protected from changes made to the privileges and permissions within the default account.

Perform the following steps to create a least-privileged account to host your service:

1. Create a Windows account
2. Run the following aspnet_regiis.exe command to assign the relevant ASP.NET permissions to the account:

   ```
   aspnet_regiis.exe -ga machineName\userName
   ```

   Note: This step is needed if your application needs to run in ASP.NET compatibility mode; otherwise, you can skip the step.
3. Use the Local Security Policy tool to grant the Windows account the **Deny logon locally** user right.
   This reduces the privileges of the account and prevents anyone from logging on to Windows locally with this account.
4. Use the least-privileged account to run your WCF service:

- If your service is hosted in IIS 6.0, use IIS Manager to create an application pool running as an account identity. Use IIS Manager to assign your WCF service to that application pool.
- If your service is hosted in Windows service, configure the Windows service to run using the account identity. This would enable the WCF service will run under the security context of account identity.

Additional Resources

- For more information on the aspnet_regiis.exe tool, see "ASP.NET IIS Registration Tool (Aspnet_regiis.exe)" at http://msdn.microsoft.com/en-us/library/k6h9cz8h(VS.80).aspx

Impersonation/Delegation

- **How to Choose Between a Trusted Subsystem and Impersonation/Delegation**
- **How to Impersonate the Original Caller when Using Windows Authentication**
- **How to Impersonate Programmatically in WCF**
- **How to Impersonate Declaratively in WCF**
- **How to Delegate the Original Caller to Call Back-end Services when Using Windows Authentication**
- **How to Impersonate the Original Caller Without Windows Authentication**
- **How to Impersonate the Original Caller Using S4U Kerberos Extensions**
- **How to Delegate the Original Caller Using S4U Kerberos Extensions**
- **How to Impersonate and Delegate Using the LogonUser Windows API**
- **How to Flow the Original Caller from an ASP.NET Client to WCF**
- **How to Control Access to a Remote Resource Based on the Original Caller's Identity**

How to Choose Between a Trusted Subsystem and Impersonation/Delegation

With the trusted subsystem model, you use your WCF service's process identity to access downstream network resources such as databases. With impersonation/delegation, you use impersonation and use the original caller's identity to access the database.

A trusted subsystem offers better scalability because your application benefits from efficient connection pooling. You also minimize back-end ACL management. Only the trusted identity can access the database — your end users have no direct access. In the trusted subsystem model, the WCF service is granted broad access to back-end resources. As a result, a compromised WCF service could potentially make it easier for an attacker to gain broad access to back-end resources. Keeping the service account's credentials protected is essential.

With impersonation/delegation, you benefit from operating system auditing because you can track which users have attempted to access specific resources. You can also enforce granular access controls in the database, and individual user accounts can be restricted independently of one another in the database.

Additional Resources

- For more information on the trusted subsystem model, see "Trusted Subsystem" at http://msdn.microsoft.com/en-us/library/ms730288.aspx
- For more information on delegation and impersonation, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from Windows Forms" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Windows%20Forms&referringTitle=How%20Tos
- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from a Web Application" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Web%20Application&referringTitle=How%20Tos

How to Impersonate the Original Caller when Using Windows Authentication

When using Windows authentication, you have access to original callers Windows identities. You can impersonate the original caller whenever downstream code needs to authorize based on the original caller's identity. For instance, you may have authorization checks in business logic called by WCF, or you may want to access resources that have access control lists (ACLs) allowing specific user access.

You can impersonate the original caller either declaratively or programmatically, depending on the following circumstances:

- Impersonate the original caller declaratively when you want to access Microsoft Windows® resources that are protected with ACLs configured for your application's domain user accounts.
- Impersonate the original caller programmatically when you want to access resources predominantly by using the application's process identity, but specific sections of the operation need to use the original caller's identity.

Additional Resources

- For more information on delegation and impersonation, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from Windows Forms" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Windows%20Forms&referringTitle=How%20Tos

- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from a Web Application" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Web%20Application&referringTitle=How%20Tos

How to Impersonate Programmatically in WCF

Programmatic impersonation allows you to impersonate on specific lines of code rather than the entire operation. This fine-grained approach to impersonation can reduce security risks; however, be aware that it is easier to make a mistake during implementation that could leave your code impersonating at higher privilege in the event of an error. Use the **using** statement to revert impersonation automatically.

To impersonate the original caller programmatically, you need to have access to Windows identity of the original caller, calling into your WCF service. For this, you need to configure your WCF service to require Windows authentication.

Use the **Impersonate** method of the **ServiceSecurityContext.Current.WindowsIdentity** class as follows:

```
public string GetData(int value)
{
 using (ServiceSecurityContext.Current.WindowsIdentity.Impersonate())
 {
     // Execute under security context of the original caller
 }
}
```

Important: Revert the impersonation when you are done; in the above example, the **using** statement does this for you.

Additional Resources

- For more information on delegation and impersonation, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from Windows Forms" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Windows%20Forms&referringTitle=How%20Tos
- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from a Web Application" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Web%20Application&referringTitle=How%20Tos
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of "WCF 3.5 Security Questions and Answers" at

http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Impersonate Declaratively In WCF

Use the **OperationBehavior** attribute to impersonate declaratively. There are two options for declarative impersonation:

- **Impersonating on specific operations**
- **Impersonating on the entire service**

Impersonating on Specific Operations

Use this option when you want to impersonate the original caller for the entire duration of a specific operation. You can impersonate declaratively by applying the **OperationBehaviorAttribute** attribute on any operation that requires client impersonation, as shown in the following code example:

```
 [OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
   return “test”;
}
```

Impersonating on the Entire Service

Use this option when you want to impersonate the original caller for the entire duration of all the operations. To impersonate the entire service, set the **impersonateCallerForAllOperations** attribute to "**true**" in the WCF configuration file, as shown in the following example:

```
…
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceAuthorization impersonateCallerForAllOperations="true" />
    </behavior>
  </serviceBehaviors>
</behaviors>
…
```

When impersonating for all operations, the **Impersonation** property of the **OperationBehaviorAttribute** applied to each method must also be set to either **Allowed** or **Required**.

Additional Resources

- For more information on delegation and impersonation, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For more information on impersonation, see “How To – Impersonate the Original Caller in WCF Calling from Windows Forms” at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-

%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Windows%20Forms&referringTitle=How%20Tos

- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from a Web Application" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Web%20Application&referringTitle=How%20Tos
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Delegate the Original Caller to Call Back-end Services when Using Windows Authentication

Use delegation for flowing the impersonated original user's security context (Windows identity) to the remote back-end service. On the remote back-end service, the original user's Windows identity can be used to authenticate or impersonate the original caller, in order to restrict or authorize the original caller's access to local resources.

Perform the following steps to delegate the original caller to back-end resources:

1. Configure the WCF process Identity to be trusted for delegation. On Windows Server 2003 or later, use constrained delegation. This allows administrators to specify exactly which services can be accessed on a downstream server or a domain account.
2. Impersonate the original caller by using either programmatic impersonation or declarative impersonation, when accessing the downstream resources.

Additional Resources

- For more information on delegation and impersonation, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Impersonate the Original Caller Without Windows Authentication

When using non-Windows authentication such as certificate or username authentication, if you need to impersonate the original caller (if the caller has a Windows account) or a service account, you have the following two options:

1. **Using the service for user (S4U) Kerberos extensions.** To use this option, you must grant your process account the "Act as part of the operating system" user right.

2. **Using the LogonUser Windows API.** This needs to have access to the user credentials (username and password), which increases the security risk of maintaining the user credentials in the WCF service.

Note: S4U Kerberos extensions place your process within the trusted computing base (TCB) of the Web server, which makes your Web server process very highly privileged. Where possible, you should avoid this approach because an attacker who manages to inject code and compromise your Web application will have unrestricted capabilities on the local computer.

Additional Resources

- For more information on delegation and impersonation, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of "WCF 3.5 Security Questions and Answers" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Impersonate the Original Caller Using S4U Kerberos Extensions

Perform the following steps to impersonate the original caller using S4U Kerberos extensions:

1. Grant your WCF process account the **Act as part of the operating system** user right. If you are running using the network service account, the account has this right by default.
2. Get the user name for the original caller and create a user principal name (UPN) for the user n a format similar to the following:

   username@FullyQualifiedDomainName.com

3. Using the **WindowsIdentity** constructor, pass the UPN string as the parameter, get the WindowsIdentity token, and impersonate the original caller as follows:

```
String username = "username@FullyQualifiedDomainName.com";
WindowsIdentity winId = new WindowsIdentity(userName);
using (winId.Impersonate())
{
  // access the local resources on behalf of the original callers
}
```

4. Make sure to revert the impersonation; in the above example, the **using** statement does this for you automatically.

Additional Resources

- For more information on delegation and impersonation, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For more information on delegation and impersonation, see "How To – Use Protocol Transition for Impersonating and Delegating the Original Caller in WCF."

How to Delegate the Original Caller Using S4U Kerberos Extensions

Perform the following steps to delegate the original caller using S4U Kerberos extensions:

1. Grant your WCF process account the **Act as part of the operating system** user right. If you are running using the network service account, the account has this right by default.
2. Configure the WCF Process Identity with "Trust this computer for delegation to your specified services only," by selecting the **Use any authentication protocol** option.
3. Get the user name for the original caller and create a UPN for the user in a format similar to the following:

   username@FullyQualifiedDomainName.com

4. Using the **WindowsIdentity** constructor, pass the UPN string as the parameter, get the WindowsIdentity token, and impersonate the original caller as follows:

```
String username = "username@FullyQualifiedDomainName.com";
WindowsIdentity winId = new WindowsIdentity(userName);
using (winId.Impersonate())
{
 // access the remote resources on behalf of the original caller
}
```

5. Make sure to revert the impersonation; in the above example, the **using** statement does this for you automatically.

Additional Resources

- For more information on delegation and impersonation, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For more information on delegation and impersonation, see "How To – Use Protocol Transition for Impersonating and Delegating the Original Caller in WCF."

How to Impersonate and Delegate Using the LogonUser Windows API

You can use the Microsoft Win32® LogonUser() API (via **P/Invoke**) to create impersonation tokens, but only when your WCF service is not trusted for delegation, because this option forces you to store usernames and passwords on your WCF service.

The following code example shows how the LogonUser API is used for impersonation:

```
using System.Runtime.InteropServices;
…
// Declare the logon types as constants
const long LOGON32_LOGON_NETWORK = 3;

// Declare the logon providers as constants
const long LOGON32_PROVIDER_DEFAULT = 0;

[DllImport("advapi32.dll",EntryPoint = "LogonUser")]
private static extern bool LogonUser(
```

```
            string lpszUsername,
            string lpszDomain,
            string lpszPassword,
            int dwLogonType,
            int dwLogonProvider,
            ref IntPtr phToken);
[DllImport("kernel32.dll", CharSet=CharSet.Auto)]
public extern static bool CloseHandle(IntPtr handle);

private void ImpersonateAndUse(string Username,
                                        string Password,
                                        string Domain)
{
  IntPtr token = new IntPtr(0);
  token = IntPtr.Zero;
  // Call LogonUser to obtain a handle to an access token.
  bool returnValue = LogonUser(Username, Domain,Password,
                                  (int)LOGON32_LOGON_NETWORK,
                                  (int)LOGON32_PROVIDER_DEFAULT,
                                  ref token);
  if (false == returnValue)
  {
     int ret = Marshal.GetLastWin32Error();
     string strErr = String.Format("LogonUser failed with error code : {0}",
ret);
     throw new ApplicationException(strErr, null);
  }
  WindowsIdentity newId = new WindowsIdentity(token);
  WindowsImpersonationContext impersonatedUser = newId.Impersonate();
  try
  {
     // do the operations using original user security context
  }
  finally
  {
     // stop impersonating
     impersonatedUser.Undo();
     CloseHandle(token);
  }
}
```

Additional Resources

- For more information on the LogonUser API, see “How to validate user credentials on Microsoft operating systems” at http://support.microsoft.com/kb/q180548/
- For more information on delegation and impersonation, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For an impersonation and delegation Q&A, see the Impersonation/Delegation section of “WCF 3.5 Security Questions and Answers“ at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=WCF%20Questions%20and%20Answers%20%28Q%26A%29&referringTitle=Home

How to Flow the Original Caller from an ASP.NET Client to WCF

Perform the following steps to impersonate the original caller from an ASP.NET client to a WCF service:

1. Configure your WCF service to use Windows authentication.
2. Configure the ASP.NET application's process identity for constrained delegation to the WCF service.
3. Impersonate the Original Caller in ASP.NET when calling the WCF service, as follows:

```
using System.Security.Principal;
…
protected void Button1_Click(object sender, EventArgs e)
{
    // Obtain the authenticated user's Identity and impersonate the
original caller
    using
(((WindowsIdentity)HttpContext.Current.User.Identity).Impersonate())
    {
        WCFTestService.ServiceClient myService = new
WCFTestService.ServiceClient();
        Response.Write(myService.GetData(123) + "<br/>");
        myService.Close();
    }
}
…
```

Additional Resources

- For more information on delegation and impersonation, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from a Web Application" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Web%20Application&referringTitle=How%20Tos

How to Control Access to a Remote Resource Based on the Original Caller's Identity

Use delegation to flow the impersonated original user's security context (Windows identity) to the remote back-end service. On the remote back-end service, the original user's Windows identity can be used to authenticate or impersonate the original caller, in order to restrict or authorize the original caller's access to local resources.

When using delegation, on Windows Server 2003 or later, use constrained delegation. This allows administrators to specify exactly which services on a downstream server or a domain account can be accessed when using an impersonated user's security context.

Additional Resources

- For more information on delegation and impersonation, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from Windows Forms" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Windows%20Forms&referringTitle=How%20Tos
- For more information on impersonation, see "How To – Impersonate the Original Caller in WCF Calling from a Web Application" at http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Web%20Application&referringTitle=How%20Tos

Message Validation

- **How to Protect Your Service from Malicious Messages**
- **How to Protect Your Service from Malicious Input**
- **How to Protect Your Service from Denial of Service Attacks**
- **How to Validate Parameters with Parameter Inspectors**
- **How to Validate Parameters with Message Inspectors Using Schemas**
- **How to Validate Data Contracts with Message Inspectors Using Schemas**
- **How to Validate Message Contracts with Message Inspectors Using Schemas**
- **How to Use Regular Expressions to Validate Format, Range, and Length in Schemas**
- **How to Validate Inbound Messages on a Service**
- **How to Validate Outbound Messages on a Service**
- **How to Validate Outbound Messages on the Client**
- **How to Validate Inbound Messages on the Client**
- **How to Validate Input Parameters**
- **How to Validate Output Parameters**

How to Protect Your Service from Malicious Messages

Use schema validation to validate and protect your service from maliciously formed messages. Validation of messages with schemas can protect parameters and/or fields in operation, data, and message contracts. Use schema validation to validate for format, range, type, and length. Using schema validation allows separation of business code from validation logic. It also allows validating message and data contract which include several fields. Complex types in parameters can also be validated with schemas. Without scehemas, such validation can often require writing complex validation code.

Additional Resources

- For more information, see "Message Inspectors" at http://msdn.microsoft.com/en-us/library/aa717047.aspx

How to Protect Your Service from Malicious Input

Use schemas to validate your service against malicious input. You can protect parameters in operation contracts, and fields in message and data contracts. The parameters in operation contracts can be simple or complex types. This protection level will require that you implement message inspectors to be used by your service and/or by the clients that consume your service. You can also protect your service by validating the parameters in the operation contracts. This protection level will require that you implement parameter inspectors to be used by your service and/or the clients that consume your service. You can do client-side and service-side validation for both schema and parameter validation.

Additional Resources

- For more information, see "Message Inspectors" at http://msdn.microsoft.com/en-us/library/aa717047.aspx
- For more information, see "How To Validate an XML Document by Using DTD, XDR, or XSD in Visual C# .NET" at http://support.microsoft.com/kb/307379

How to Protect Your Service from Denial Of Service Attacks

Protect against denial of service (DoS) attacks by limiting message sizes, and by using quotas to restrict memory consumption by WCF.

Restrict the message size that is processed by WCF by using the **maxReceivedMessageSize** configuration present in the bindings, as shown below:

```
<binding name="wsHttpEndpointBindingconfig" maxReceivedMessageSize="65535">
     <security>
      <message negotiateServiceCredential="false" />
     </security>
</binding>
```

Restrict the buffer size used by WCF by using the **maxBufferPoolSize** configuration present in the bindings, as shown below:

```
<binding name="wsHttpEndpointBindingconfig" maxBufferPoolSize="524287"
         maxReceivedMessageSize="65535">
     <security>
      <message negotiateServiceCredential="false" />
     </security>
</binding>
```

In streaming scenarios, use the reader quotas to limit the size of arrays with **maxArrayLength**, the length of the string in XML elements with **maxStringContentLength**, the maximum depth of the XML node with **maxDepth**, the maximum bytes to be read with **maxBytesPerRead**, and the maximum number of characters in a table with **maxNameTableCharCount**.

```
      <basicHttpBinding>
        <binding name="BasicBindingConfiguration">
          <readerQuotas maxDepth="2" maxStringContentLength="200"
maxArrayLength="2000"
```

```
      maxBytesPerRead="1000" maxNameTableCharCount="1000" />
    <security mode="Transport">
      <transport clientCredentialType="None" />
    </security>
  </binding>
</basicHttpBinding>
```

How to Validate Parameters with Parameter Inspectors

Perform the following steps to validate parameters with parameter inspectors:

1. Create a class that implements the validation logic. This class has to derive from **IParameterInspector**. The class has the following characteristics:
    - It implements the **AfterCall()** and **BeforeCall()** methods, both of which will have the validation logic.
    - When used as part of the service, **BeforeCall()** will be invoked before the parameters are dispatched to the service operation. **AfterCall()** will be invoked after the service has processed the call and is returning a the response to the client. Use **BeforeCall()** to validate your input parameters, and use **AfterCall()** to validate your output parameters.
    - When used as part of the client, **BeforeCall()** will be invoked before calling the service, and **AfterCall()**will be invoked before the service's response is dispatched to the client code. Use **AfterCall()** to validate the response from the service, and use **BeforeCall()** to validate input parameters before calling the service.
2. Create a class that implements a custom endpoint behavior. This class derives from **IEndpointBehavior**, which the service and/or client endpoint will use as a configuration extensibility point for the endpoint. This class has the following characteristics:
    - It implements **ApplyClientBehavior()** to add the **ParamaterInspector** to the client operation and enable client-side validation.
    - It implements **ApplyDispatchBehavior()** to add the **ParameterInspector** to the dispatch operation and enable service-side validation.
    - It verifies that it is enabled in the configuration before adding the **ParameterInspector** to the client or dispatch run time.
3. Create a class that implements a custom configuration element. This class derives from **BehaviorExtensionElement**. This class allows you to expose the endpoint configuration in WCF as a behavior element extension, which can be used by the service as an endpoint behavior configuration. This class has the following characteristics:
    - It implements **CreateBehavior()** to create an instance of the **ValidationBehavior** class.
    - It implements **BehaviorType()** to return the **ValidationBehavior** type. This allows the custom behavior to be exposed in the service or client configuration sections.
4. Add the custom behavior to the configuration file . Add the custom behavior element to the behavior element extension items so that it can be used by the endpoint behavior.

You add it in the configuration file by using the configuration tool to browse to the assembly and then selecting your custom behavior type.

5. Create an endpoint behavior and map it to use the custom behavior. The custom behavior is the extensibility point containing the parameter validation. The behavior is instantiated by the assembly implementing the parameter inspector logic.
6. Configure the service endpoint to use the endpoint behavior. Configure the endpoint to use the endpoint behavior that is using the parameter inspector.

Additional Resources

- For more information, see "How to: Inspect or Modify Parameters" at http://msdn.microsoft.com/en-us/library/ms733747.aspx?wt.svl=overview
- For more information, see "Message Inspectors" at http://msdn.microsoft.com/en-us/library/aa717047.aspx
- For more information, see "How to Perform Input Validation" contained in the "How To" section of this guide.

How to Validate Messages with Message Inspectors Using Schemas

Perform the followings steps to validate messages with message inspectors using schemas:

1. Create a class that implements the validation logic. This class has to derive from **IClientMessageInspector,IDispatchMessageInspector**, depending on whether you want to do client-side and/or server-side validation. This class implements the **AftterReceiveRequest**(), **BeforeSendReply**(), **BeforeSendRequest**(), and **AfterReceiveReply**() methods. This class has the following characteristics:
   - **On the dispatcher: AfterReceiveRequest** will be implemented when inbound messages are received by the dispatcher, before the operation is invoked and deserialization of messages has occurred. If the message is encrypted, decryption will take place first. **BeforeSendReply** will be implemented when outbound messages are to be sent back to the client, after the operation is invoked and serialization has occurred. If the message is encrypted, encryption will not take place.
   - **On the client: BeforeSendRequest** will be implemented when outbound messages are sent by the client, after serialization has occurred. If a message is encrypted, encryption will not take place. **AfterReceiveReply** will be implemented when inbound messages are received by the client, before deserialization of message has occurred. If the message is encrypted, decryption will take place first.
2. Create a class that implements a custom endpoint behavior. This class derives from **IEndpointBehavior**, which the service and/or client endpoint will use as a configuration extensibility point for the endpoint. This class has the following characteristics:
   - It implements **ApplyClientBehavior()** to add the **ParamaterInspector** to the client operation and enable client-side validation.
   - It implements **ApplyDispatchBehavior()** to add the **ParameterInspector** to the dispatch operation and enable service-side validation.

    - It verifies that it is enabled in the configuration before adding the **ParameterInspector** to the client or dispatch run time.
3. Create a class that implements a custom configuration element. This class derives from **BehaviorExtensionElement**, which allows you to expose the endpoint configuration in WCF as a behavior element extension that can be used by the service as an endpoint behavior configuration. This class has the following characteristics:
    - It implements **CreateBehavior()** to create an instance of the **ValidationBehavior** class.
    - It implements **BehaviorType()** to return the **ValidationBehavior** type. This allows the custom behavior to be exposed in the service or client configuration sections.
4. Add the custom behavior to the configuration file. Add the custom behavior element to the behavior element extension items, so that it can be used by the endpoint behavior. You add it in the configuration file by using the configuration tool to browse to the assembly and then selecting your custom behavior type.
5. Create an endpoint behavior and map it to use the custom behavior. The custom behavior is the extensibility point containing the parameter validation. The behavior is instantiated by the assembly implementing the parameter inspector logic.
6. Configure the service endpoint to use the endpoint behavior. Configure the endpoint to use the endpoint behavior that is using the message inspector.

Additional Resources

- For more information, see “Message Inspectors” at http://msdn.microsoft.com/en-us/library/aa717047.aspx
- For more information, see “How to Perform Input Validation” contained in the “How To” section of this guide.

How to Validate Data Contracts with Message Inspectors Using Schemas

Perform the following high-level steps to validate data contracts passed to operations in WCF:

1. Create a message inspector to perform schema validation.
2. Create a schema file to validate the fields of the data contract with facets in the schema file.

See the following example, in which the **CustomerData** complex type is validated by using **CustomerN** and **CustIDLimitor**. This example limits integers to a value no greater than 5, and the string can be no longer than five characters.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
targetNamespace="http://Microsoft.PatternPractices.WCFGuide"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://Microsoft.PatternPractices.WCFGuide">
  <xs:element name="GetData">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element minOccurs="1" name="CustomerInfo"
                    nillable="false" type="tns:CustomerData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="CustomerData">
    <xs:sequence>
      <xs:element name="CustomerID" type="tns:CustIDLimitor">
      </xs:element>
      <xs:element name="text" type="tns:CustomerN">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="CustomerN">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
      <xs:maxLength value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CustIDLimitor">
    <xs:restriction base="xs:int">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="GetDataResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="GetDataResult"
                    nillable="false" type="tns:CustomerData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Additional Resources

- For more information, see “Message Inspectors” at http://msdn.microsoft.com/en-us/library/aa717047.aspx

How to Validate Message Contracts with Message Inspectors Using Schemas

Perform the following steps to validate message contracts passed to operations in WCF:

1. Create a message inspector to perform schema validation.
2. Create a schema file to validate the fields of the message contract with facets in the schema file.

See the following example, in which the message contract **CustomerData** is validated by using **CustomerN** and **CustIDLimitor**. This example limits integers to a value no greater than 5, and the string can be no longer than five characters.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<xs:schema elementFormDefault="qualified"
targetNamespace="http://Microsoft.PatternPractices.WCFGuide"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://Microsoft.PatternPractices.WCFGuide">
  <xs:element name="GetData">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="CustomerInfo"
                    nillable="false" type="tns:MessageData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="CustomerData">
    <xs:sequence>
      <xs:element name="CustomerID" type="tns:CustIDLimitor">
      </xs:element>
      <xs:element name="text" type="tns:CustomerN">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="CustomerN">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
      <xs:maxLength value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CustIDLimitor">
    <xs:restriction base="xs:int">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="GetDataResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="GetDataResult"
                    nillable="false" type="tns:CustomerData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Additional Resources

- For more information, see “Message Inspectors” at http://msdn.microsoft.com/en-us/library/aa717047.aspx

How to Use Regular Expressions to Validate Format, Range, and Length in Schemas

Use regular expressions in schemas to validate format, range, or length. This allows you to use complex validation logic without needing to implement the code. It also allows decoupling of the validation logic from the business logic. The example schema below exemplifies the validation of integers with values between 1 and 5, the string of length 5, and a Social Security Number (SSN) and ZIP code with good formats:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
targetNamespace="http://Microsoft.PatternPractices.WCFGuide"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://Microsoft.PatternPractices.WCFGuide">
  <xs:element name="GetData">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="CustomerInfo"
                    nillable="false" type="tns:CustomerData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="CustomerData">
    <xs:sequence>
      <xs:element name="CustomerID" type="tns:CustIDLimitor">
      </xs:element>
      <xs:element name="text" type="tns:CustomerN">
      </xs:element>
      <xs:element name="socialSecurity" type="tns:SSN">
      </xs:element>
      <xs:element name="custZipCode" type="tns:CustomerN">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="CustomerN">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
      <xs:maxLength value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CustIDLimitor">
    <xs:restriction base="xs:int">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="SSN">
    <xs:restriction base="xs:token">
      <xs:pattern value="[0-9]{3}-[0-9]{2}-[0-9]{4}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="us-zipcode">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{5}(-[0-9]{4})?"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="GetDataResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="GetDataResult"
                    nillable="false" type="tns:CustomerData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

How to Validate Inbound Messages on a Service

Implement the **AfterReceiveRequest** method of the message inspector's **IDispatchMessageInspector** interface in order to validate inbound messages on a service. This allows you to validate the message after the request has arrived but before service operation invocation and deserialization.

```
object IDispatchMessageInspector.AfterReceiveRequest(ref
System.ServiceModel.Channels.Message request,
System.ServiceModel.IClientChannel channel,
System.ServiceModel.InstanceContext instanceContext)
{
    try
    {
        validateMessage(ref request);
    }
    catch (FaultException e)
    {
        throw new FaultException<string>(e.Message);
    }
    return null;
}
```

Additional Resources

- For more information, see "Message Inspectors" at http://msdn.microsoft.com/en-us/library/aa717047.aspx

How to Validate Outbound Messages on a Service

Implement the **BeforeSendReply** method of the message inspector's **IDispatchMessageInspector** interface in order to validate outbound messages on a service. This allows you to validate the message before sending the response to the client and before service operation invocation and serialization.

```
void IDispatchMessageInspector.BeforeSendReply(ref
System.ServiceModel.Channels.Message reply, object correlationState)
{
    try
    {
        validateMessage(ref reply);
    }
    catch (FaultException fault)
    {
        // if a validation error occurred, the message is replaced
        // with the validation fault.
        reply = Message.CreateMessage(reply.Version, new
FaultException("validation error in reply message").CreateMessageFault() ,
reply.Headers.Action);
    }
}
```

Additional Resources

- For more information, see "Message Inspectors" at http://msdn.microsoft.com/en-us/library/aa717047.aspx

How to Validate Outbound Messages on the Client

Implement the **BeforeSendRequest** method of the message inspector's **IClientMessageInspector** interface in order to validate outbound messages on the client. This allows you to validate the message after serialization but before sending the request to the service.

```
object IClientMessageInspector.BeforeSendRequest(ref
System.ServiceModel.Channels.Message request,
System.ServiceModel.IClientChannel channel)
{
    validateMessage(ref request);
    return null;
}
```

Additional Resources

- For more information, see "Message Inspectors" at http://msdn.microsoft.com/en-us/library/aa717047.aspx

How to Validate Inbound Messages on the Client

Implement the **AfterReceiveReply** method of the message inspector's **IClientMessageInspector** interface in order to validate inbound messages on the client. This allows you to validate the message after the client response has arrived but before deserialization and before returning the data to the client application.

```
void IClientMessageInspector.AfterReceiveReply(ref
System.ServiceModel.Channels.Message reply, object correlationState)
 {
     validateMessage(ref reply);
 }
```

Additional Resources

- For more information, see "Message Inspectors" at http://msdn.microsoft.com/en-us/library/aa717047.aspx

How to Validate Input Parameters

Implement the **BeforeCall**() method on the parameter inspector to validate input parameters on a client or service. Inside **BeforeCall**(), implement the validation logic to validate the parameters.

```
public class ValidationParameterInspector : IParameterInspector
{
  public object BeforeCall(string operationName, object[] inputs)
  { …    }
}
```

Additional Resources

- For more information, see "How to: Inspect or Modify Parameters" at http://msdn.microsoft.com/en-us/library/ms733747.aspx?wt.svl=overview

How to Validate Output Parameters

Implement the **AfterCall**() method on the parameter inspector to validate output parameters on a client or service. Inside **AfterCall**(), implement the validation logic to validate the parameter.

```
public class ValidationParameterInspector : IParameterInspector
{
   public void AfterCall(string operationName, object[] outputs,
                object returnValue, object correlationState)
   { …    }
}
```

Additional Resources

- For more information, see "How to: Inspect or Modify Parameters" at http://msdn.microsoft.com/en-us/library/ms733747.aspx?wt.svl=overview

Message Security

- **How to Use Message Security**
- **How to Control the Level of Message Encryption**
- **How to Use Out-of-band Credentials with Message Security**

How to Use Message Security

Use the **<Security mode>** attribute to configure message security on your binding.

Perform the following steps to configure **wsHttpBinding** to use message security:

1. Open your app.config or web.config file and set the security mode to **Message** as follows:

```
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security mode="Message">
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

2. Save the configuration file.

Message security is available on all of the bindings except for **netNamedPipeBinding**.

Additional Resources

- For more information, see "Message Security in WCF" at http://msdn.microsoft.com/en-us/library/ms733137.aspx

How to Control the Level of Message Encryption

If you are using message security, use the **[ServiceContract(ProtectionLevel)]** attribute to specify message security protection levels on the interface or operation level.

The protection level options available are:

- **None.** Use None to turn off signing and encryption on the operation or interface.
- **Sign.** Use Sign to sign the interface or operation but not encrypt it.
- **EncryptAndSign.** Use EncryptAndSign to both encrypt and sign the interface or operation.

If you are using transport security, you cannot partially encrypt your messages.

The following code example shows how set the protection level to **Sign** on an interface:

```
[ServiceContract(ProtectionLevel=ProtectionLevel.Sign]
public interface IService
{
      string GetData(int value);
}
```

The following code example shows how to set the protection level to **Sign** on an:

```
[OperationContract(ProtectionLevel=ProtectionLevel.Sign]
string GetData(int value);
```

Additional Resources

- For more information on protection level sand partial encryption, see "Understanding Protection Level" at http://msdn.microsoft.com/en-us/library/aa347692.aspx

How to Use Out-of-band Credentials with Message Security

Set the **negotiateCredentials** attribute to **false** to use out-of-band credentials. This will require you to provide certificates to the client so that they can encrypt and sign messages.

Perform the following steps to configure the **negotiateCredentials** attribute:

1. Open your app.config or web.config file and set the security mode to Message.

```
<wsHttpBinding>
  <binding name="MessageAndUserName">
    <security mode="Message">
      <message clientCredentialType="UserName"
negotiateCredentials=”false” algorithmSuite="Default" />
    </security>
  </binding>
```

```
</wsHttpBinding>
```

2. Save the configuration file.

Proxy Considerations

- **How to Avoid Proxy Spoofing**
- **How to Publish Service Metadata for Your Clients**
- **How to Create a Proxy for an IIS-hosted Service with Certificate Authentication and Transport Security**

How to avoid proxy spoofing

Consider the following to avoid proxy spoofing at the time of adding a WCF service reference:

- Publish metadata securely, over Secure HTTP (HTTPS). Use **mexHttpsBinding** and configure a server certificate for the service. The following configuration shows how to publish metadata securely:

  ```
  <serviceMetadata httpGetEnabled="False" httpsGetEnabled="True"/>
  ```

- If you are required to use a mex endpoint instead of exposing your service reference by using **httpGet**, use a secure binding. Use any standard binding (that has security features) for the mex service endpoint; the only requirement is to use the **IMetadataExchange** contract. This will require you to use a custom serviceutil.exe.config file to generate the proxy.

Consider the following to avoid proxy spoofing at run time:

- Make sure that your WCF service uses mutual authentication. Mutual authentication is enforced when using either message or transport security.
- If you are using **basicHttpBinding**, this binding does not use any security by default. Make sure that it is configured to use either transport or message security.
- Do not rely on the NTLM protocol for authentication because it does not provide mutual authentication.

How to Publish Service Metadata for Your Clients

To publish service metadata for your clients:

1. If your service uses HTTP binding, you will need to enable metadata via **HttpGet** or **HttpsGet**. **HttpGet** is required if you are not using transport security. **HttpsGet** is required if you are using transport security. The trade-off with this configuration is that clients will be able to browse your service metadata.

   ```
   <serviceMetadata httpsGetEnabled="true" />
   <serviceMetadata httpGetEnabled="true"  />
   ```

2. If your service uses HTTP binding, you can use a mex endpoint without enabling **HttpGet** or **HttpsGet**. In this case, browsers will not be able to browse metadata, but the clients will be able to create proxies using the mex endpoint. The trade-off with this configuration is that mex endpoints are not possible if IIS does not have anonymous authentication enabled.

```
<system.serviceModel>
    <services>
      <service behaviorConfiguration="" name="Service">
        <endpoint address="" binding="wsHttpBinding"
bindingConfiguration=""
          name="WsBinding" contract="IService" />
        <endpoint address="mex" binding="mexHttpBinding"
bindingConfiguration=""
          name="mexendpoint" contract="IMetadataExchange" />
      </service>
    </services>
   ….
  </system.serviceModel>
```

3. If your service uses HTTP binding, you can use a custom endpoint that implements **IMetadataExchange** without enabling **HttpGet** or **HttpsGet**. In this case, browsers will not be able to browse metadata ,but the clients will be able to create proxies using the mex endpoint. Additionally, you will be able to use the mex endpoint with any authentication scheme.

```
  <services>
    <service behaviorConfiguration="returnFaults" name="MyService">
      <endpoint binding="wsHttpBinding" bindingConfiguration=""
        name="wsHttpEndpoint" contract="IService" />
      <endpoint address="mex" binding="wsHttpMexBinding"
        bindingConfiguration=""
        name="mexEndpoint" contract="IMetadataExchange" />
    </service>
  </services>
…
```

4. If your service does not use HTTP binding, you will need to configure service metadata and create a mex endpoint as follows:

```
<system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="BehaviorConfiguration">
          <serviceDebug includeExceptionDetailInFaults="true" />
          <serviceMetadata />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    ….
    <services>
      <service behaviorConfiguration="BehaviorConfiguration"
               name="WCFServicecHost.MyService">
        <endpoint
```

```
            address="Mex"
            binding="mexTcpBinding"
            bindingConfiguration=""
            name="MexEndpoint"
            contract="IMetadataExchange" />
         <endpoint
            address=""
            binding="netTcpBinding"
            bindingConfiguration="BindingConfiguration"
            name="TcpBinding"
            contract="WCFServicecHost.IMyService" />
      </service>
    </services>
 </system.serviceModel>
```

Additional Resources

- For more information on metadata endpoints, see “How to: Secure Metadata Endpoints “ at http://msdn.microsoft.com/en-us/library/ms733114.aspx

How to Create a Proxy for an IIS-hosted Service with Certificate Authentication and Transport Security

Perform the following steps to create a proxy to a service hosted in IIS that requires certificate authentication and transport security:

1. Create a new **wsHttpBinding** endpoint on the service that implements **IMexdataExchange** and uses a binding configuration with the certificate authentication type.

```
<services>
   <service behaviorConfiguration="returnFaults" name="MyService">
      <endpoint binding="wsHttpBinding" bindingConfiguration=""
        name="wsHttpEndpoint" contract="IService" />
      <endpoint address="mex" binding="wsMexHttpBinding"
        bindingConfiguration=""
        name="mexEndpoint" contract="IMetadataExchange" />
    </service>
</services>…
```

2. Create a svcutil.exe.config file on the client with configuration pointing to the certificate used to authenticate the service. The endpoint should have the contract with the **IMetadataExchange** type and will point to a binding configuration with certificate authentication.

```
<configuration>
  <system.serviceModel>
    <client>
      <endpoint behaviorConfiguration="ClientCertificateBehavior"
      binding="wsHttpBinding"
        bindingConfiguration="Binding1" contract="IMetadataExchange"
        name="https" />
    </client>
    <bindings>
```

```
        <wsHttpBinding>
          <binding name="Binding1">
            <security mode="Transport">
              <transport clientCredentialType="Certificate" />
            </security>
          </binding>
        </wsHttpBinding>
      </bindings>
      <behaviors>
        <endpointBehaviors>
          <behavior name="ClientCertificateBehavior">
            <clientCredentials>
              <clientCertificate findValue="CN=clienttempcert"
              storeLocation="CurrentUser"
                storeName="My"
              x509FindType="FindBySubjectDistinguishedName" />
            </clientCredentials>
          </behavior>
        </endpointBehaviors>
      </behaviors>
    </system.serviceModel>
</configuration>
```

3. Copy svcutil from C:\Program Files\Microsoft Visual Studio 8\Common7\IDE to the same location where svcutil.exe.config was created on the client, and then run the command **svcutil serviceurl**

Additional Resources

- For more information on metadata, see “Publishing Metadata“ at http://msdn.microsoft.com/en-us/library/aa751951.aspx

Sensitive Data

- **How to Encrypt Sensitive Data in Configuration Files**
- **How to Protect Sensitive Data in Memory**
- **How to Protect Sensitive Data on the Network**

How to Encrypt Sensitive Data in Configuration Files

To encrypt sensitive data in configuration files, use the aspnet_regiis.exe tool with the **-pe** (provider encryption) option.

Use the following command to encrypt the **connectionStrings** section using the Data Protection API (DPAPI) provider with the machine key store (the default configuration). Run the following command from a command prompt:

```
aspnet_regiis -pe "connectionStrings" -app "/MachineDPAPI" -prov
"DataProtectionConfigurationProvider"
```

In this command:

- **-pe** specifies the configuration section to encrypt.

- **-app** specifies your Web application's virtual path. If your application is nested, you need to specify the nested path from the root directory; for example, "/test/aspnet/MachineDPAPI".
- **-prov** specifies the provider name.

The Microsoft .NET Framework supports the **R** following protected configuration providers:

- **RSAProtectedConfigurationProvider.** This is the default provider. It uses RSA public key encryption to encrypt and decrypt data. Use this provider to encrypt configuration files for use on multiple WCF services in a Web farm.
- **DPAPIProtectedConfigurationProvider.** This provider uses DPAPI to encrypt and decrypt data. Use this provider to encrypt configuration files for use on a single Windows Server.

You do not need to take any special steps for decryption; the .NET run time takes care of this for you.

Additional Resources

- For more information on encrypting configuration sections, see "How To: Encrypt Configuration Sections Using DPAPI" at http://msdn2.microsoft.com/en-us/library/ms998280.aspx and "How To: Encrypt Configuration Sections Using RSA" at http://msdn2.microsoft.com/en-us/library/ms998283.aspx

How to Protect Sensitive Data in Memory

To minimize exposure of secrets in memory, consider the following measures:

- Avoid creating multiple copies of the secret. Having multiple copies of the secret data increases your attack surface. Pass references to secret data instead of making copies of the data. Also, be aware that if you store secrets in immutable objects such as **System.String**, a new copy is created after each object manipulation.
- Keep the secret encrypted for as long as possible. Decrypt the data at the last possible moment before you need to use the secret.
- Clean the clear text version of the secret as soon as you are done using it.

You can use the **SecureString** method to implement the above measures. The value of a **SecureString** object is automatically encrypted, can be modified until your application marks it as read-only, and can be deleted from computer memory by either your application or the .NET Framework garbage collector.

The following C# code creates an instance of the **SecureString** class and stores a data value in it.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace TestSecureString
{
```

```
        class Program
        {
            static void Main(string[] args)
            {
            System.Security.SecureString secstr = new
            System.Security.SecureString();
            secstr.AppendChar('W');
            secstr.AppendChar('C');
            secstr.AppendChar('F');
            secstr.MakeReadOnly();
            Console.WriteLine(secstr);
            }
        }
    }
```

An exception is thrown if you attempt to alter the data because the code locks the string value with the **MakeReadOnly** method after the final character has been added. Therefore this string value cannot be altered.

Additional Resources

- For more information on the **SecureString** class, see "SecureString Class" at http://msdn.microsoft.com/en-us/library/system.security.securestring.aspx?ref=herseybedava.info
- For more information on the **SecureString** class, see "SecureString Application Sample" at http://msdn.microsoft.com/en-us/library/07b9wyhy.aspx

How to Protect Sensitive Data on the Network

Use message or transport security to encrypt your messages and keep sensitive information from being sniffed off the network. Message security encrypts each individual message to protect sensitive data. Transport security secures the end-to-end network connection to protect the network traffic.

Additional Resources

- For more information on transport security, see "Transport Security" at http://msdn.microsoft.com/en-us/library/ms733043.aspx
- For more information on message security, see "Message Security in WCF" at http://msdn.microsoft.com/en-us/library/ms733137.aspx

Transport Security

- **How to Use Transport Security**
- **How to Use Secure Conversations in WCF**

How to Use Transport Security

Use the **<Security mode>** attribute to configure transport security on your binding. The following example shows **wsHttpBinding** configured to use transport security:

```
<bindings>
  <wsHttpBinding>
```

```
    <binding name="wsHttpEndpointBinding">
      <security mode="Transport">
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

Transport security is available on all of the bindings except for **wsDualHttpBinding**.

Additional Resources

- For more information on transport security, see "Transport Security" at http://msdn.microsoft.com/en-us/library/ms733043.aspx

How to Use Secure Conversations in WCF

Secure conversations are turned on by default for all bindings that support Web Services Security (WS-Security). This includes **wsHttpBinding**, **netTcpBinding**, and **netMsmqBinding**. If you are using a custom binding, turn on secure conversations with the **authenticationMode** attribute as follows:

```
<customBinding>
  <binding name="ServiceBinding">
    <security authenticationMode="SecureConversation"
         requireSecurityContextCancellation ="false">
       <secureConversationBootstrap authenticationMode="MutualCertificate">
       </secureConversationBootstrap>
    </security>
    <httpTransport/>
  </binding>
 </customBinding>
```

Additional Resources

- For more information on secure conversations, see "How to: Create a Stateful Security Context Token for a Secure Session" at http://msdn.microsoft.com/en-us/library/ms731814.aspx

X.509 Certificates

- **How to Create a temporary X.509 Certificate for Transport Security**
- **How to Create a temporary X.509 Certificate for Message Security**
- **How to Create a temporary X.509 Certificate for Certificate Authentication**

How to Create a Temporary X.509 Certificate for Transport Security

Perform the following steps to create a temporary X.509 certificate for transport security:

1. Create a certificate to act as your root Certificate Authority (CA):

```
makecert -n "CN=RootCATest" -r -sv RootCATest.pvk RootCATest.cer
```

2. Install your root CA on both the server and client machines. Use Microsoft Management Console (MMC) to install RootCATes.cer on the client and server machines in the Trusted Root Certification Authorities store.

3. Create and install your temporary service certificate:

```
makecert -sk keyName -iv RootCATest.pvk -n "CN=MachineName.domain.com"
-ic RootCATest.cer -sr localmachine -ss my -sky exchange -pe
```

4. Use **Inetmgr** to configure the Web site and virtual directory to use the certificate and to require Secire Sockets Layer (SSL) to secure communication.

Additional Resources

- For more information on creating certificates, see "Certificate Creation Tool (Makecert.exe)" at http://msdn.microsoft.com/en-us/library/bfsktky3(VS.80).aspx

How to Create a Temporary X.509 Certificate for Message Security

Perform the following steps to create a temporary X.509 certificate for message security:

1. Create a certificate to act as your root Certificate Authority (CA):

```
makecert -n "CN=RootCATest" -r -sv RootCATest.pvk RootCATest.cer
```

2. Create a Certificate Revocation List (CRL) file from the root certificate:

```
makecert -crl -n "CN=RootCATest" -r -sv RootCATest.pvk RootCATest.crl
```

3. Install your root CA on both the server and client machines. Use Microsoft Management Console (MMC) to install RootCATes.cer on the client and server machines in the Trusted Root Certification Authorities store.

4. Install the CRL file on both the server and client machines. Use MMC to install RootCATes.crl on the client and server machines in the Trusted Root Certification Authorities store.

5. Create and install your temporary service certificate:

```
makecert -sk MyKeyName -iv RootCATest.pvk -n "CN=tempCert" -ic
RootCATest.cer -sr localmachine -ss my -sky exchange -pe
```

6. Give the WCF process identity access to the temporary certificate's private key:

```
FindPrivateKey.exe My LocalMachine -n "CN=tempCert"
cacls.exe "C:\Documents and Settings\All Users\Application
Data\Microsoft\Crypto\RSA\Machinekeys\4d657b73466481beba7b0e1b5781db81_
c225a308-d2ad-4e58-91a8-6e87f354b030" /E /G "NT AUTHORITY\NETWORK
```

```
SERVICE":R
```

The value "C:\Documents and Settings\All Users\Application Data\Microsoft\Crypto\RSA\Machinekeys**4d657b73466481beba7b0e1b5781db81_c225a308-d2ad-4e58-91a8-6e87f354b030**" should be the one returned by findprivatekey.

Additional Resources

- For more information on creating certificates, see "Certificate Creation Tool (Makecert.exe)" at http://msdn.microsoft.com/en-us/library/bfsktky3(VS.80).aspx

How to Create a Temporary X.509 Certificate for Certificate Authentication

Perform the following steps to create a temporary X.509 certificate for certificate authentication:

1. Create a certificate to act as your Root Certificate Authority (CA):

```
makecert -n "CN=RootCATest" -r -sv RootCATest.pvk RootCATest.cer
```

2. Create a Certificate Revocation List (CRL) file from the root certificate:

```
makecert -crl -n "CN=RootCATest" -r -sv RootCATest.pvk RootCATest.crl
```

3. Install your root CA on both the server and client machines. Use Microsoft Management Console (MMC) to install the RootCATes.cer on the client and server machines in the Trusted Root Certification Authorities store.

4. Install the CRL file on both the server and client machines. Use MMC to install RootCATes.crl on the client and server machines in the Trusted Root Certification Authorities store.

5. Create and install your temporary service certificate:

```
makecert -sk MyKeyName -iv RootCATest.pvk -n "CN=tempCert" -ic
RootCATest.cer -sr currentuser -ss my -sky signature –pe
```

Additional Resources

- For more information on creating certificates, see "Certificate Creation Tool (Makecert.exe)" at http://msdn.microsoft.com/en-us/library/bfsktky3(VS.80).aspx

WCF Security Question and Answers (Q&A)

Index

Design Considerations

- How do I decide on an authentication strategy?
- How do I decide on an authorization strategy?
- When should I use message security versus transport security?
- How do I use my existing Active Directory infrastructure?
- What bindings should I use over the Internet?
- What bindings should I use over the intranet?
- When should I use resource-based authorization versus roles-based authorization?
- When should I impersonate the original caller?
- When should I flow the original caller's identity to back-end resources?
- How do I migrate to WCF from an ASMX Web service?
- How do I migrate to WCF from a COM application?
- How do I migrate to WCF from a DCOM application?
- How do I migrate to WCF from a WSE application?

Auditing and Logging

- What WCF service security events should be logged?
- How do I enable logging and auditing in WCF?
- How do I stop my service if there has been an auditing failure?
- How do I log important business events in WCF?
- How do I implement log throttling in WCF?
- How do I use the health monitoring feature with WCF?
- How do I protect my log files?
- How do I pass user identity information in a message for auditing purpose?

Authentication

- How do I decide on an authentication strategy in WCF?
- When should I use the SQL Server membership provider?
- How do I authenticate against Active Directory?
- How do I authenticate against a SQL store?
- How do I authenticate against a custom store?
- How do I protect passwords in my user store?
- How do I use certificate authentication with X.509 certificates?
- What is the most common authentication scenario for intranet applications?
- What is the most common authentication scenario for Internet applications?
- How do I support authentication for multiple client types?
- What is federated security?

- How do I send credentials in the message when I am using transport security?
- How do I avoid cleartext passwords?

Authorization

- How do I decide on an authorization strategy in WCF?
- What's the difference between resource-based, roles-based, and claims-based authorization?
- How do I use Windows groups for role authorization in WCF?
- How do I use the SQL Server role provider for ASP.NET role authorization in WCF?
- How do I use the Windows Token role provider for ASP.NET role authorization in WCF?
- How do I use the Authorization Store role provider for ASP.NET role authorization in WCF?
- What is the difference between declarative and imperative roles authorization?
- How do I restrict access to WCF operations to specific Windows users?
- How do I associate roles with a certificate?
- What is a service principal name (SPN)?
- How do I create a service principal name (SPN)?

Bindings

- What is a binding?
- What bindings are available?
- Which bindings are best suited for the Internet?
- Which bindings are best suited for an intranet?
- How do I choose an appropriate binding?

Configuration Management

- How do I encrypt sensitive data in the WCF configuration file?
- How do I run a WCF service with a particular identity?
- How do I create a service account for running my WCF service?
- When should I use a configuration file versus the WCF object model?
- What is a metadata exchange (MEX) binding?
- How do I keep clients from referencing my service?

Deployment Considerations

- What are the additional considerations for using WCF in a Web farm?
- How do I configure Active Directory groups and accounts for role-based authorization checks?
- How do I create an X.509 certificate?
- When should I use a service principal name (SPN)?
- How do I configure a least-privileged account for my service?

Exception Management

- How do I implement a global exception handler?
- What is a fault contract?
- How do I define a fault contract?
- How do I avoid sending exception details to the client?

Hosting

- How do I configure a least-privileged account to host my service?
- When should I host my service in Internet Information Services (IIS)?
- When should I host my service in a Windows service?
- When should I self-host my service?

Impersonation/Delegation

- What are my impersonation options?
- What is the difference between impersonation and delegation?
- How do I impersonate the original caller for an operation call?
- How do I temporarily impersonate the original caller in an operation call?
- How do I impersonate a specific (fixed) identity?
- What is constrained delegation?
- What is protocol transition?
- How do I flow the original caller from the ASP.NET client to a WCF service?
- What is the difference between declarative and programmatic impersonation?
- What is the trusted subsystem model?
- When should I flow the original caller to back-end code?
- How do I control access to a remote resource based on the original caller's identity?

Input/Data Validation

- How do I implement input and data validation in WCF?
- What is schema validation?
- What is parameter validation?
- Should I validate before or after message serialization?
- How do I protect my service from denial of service (DoS) attacks?
- How do I protect my service from malicious input attacks?
- How do I protect my service from malformed messages?

Message Protection

- When should I use message security?
- When should I use transport security?
- How do I protect my message when there are intermediaries routing my message?
- How do I protect my message when there are multiple protocols used during message transit?

Proxy Considerations

- When should I use a channel factory?
- When do I need to expose a metadata exchange (MEX) endpoint for my service?
- How do I avoid proxy spoofing?

Sensitive Data

- How do I protect sensitive data in configuration files?
- How do I protect sensitive data in memory?
- How do I protect my metadata?
- How do I protect sensitive data from being read on the wire?
- How do I protect sensitive data from being tampered with on the wire?

X.509 Certificates

- How do I create X.509 certificates?
- Do I need to create a certificate signed by the root CA certificate?
- How do I use X.509 certificate revocation?

Design Considerations

- **How do I decide on an authentication strategy?**
- **How do I decide on an authorization strategy?**
- **When should I use message security versus transport security?**
- **How do I use my existing Active Directory infrastructure?**
- **What bindings should I use over the Internet?**
- **What bindings should I use over the intranet?**
- **When should I use resource-based authorization versus roles-based authorization?**
- **When should I impersonate the original caller?**
- **When should I flow the original caller's identity?**
- **How do I migrate to WCF from an ASMX Web service?**
- **How do I migrate to WCF from a COM application?**
- **How do I migrate to WCF from a DCOM application?**
- **How do I migrate to WCF from a WSE application?**

How do I decide on an authentication strategy?

Decide your authentication strategy based on your user credential store location and the location of your clients on the Internet or intranet.

Internet

- **Username authentication with SQL Membership Provider**. If your users are not in active directory, consider SQL Membership Provider. This will give you a store that can be easily deployed and created. Configure message or mixed mode security to protect your users' credentials.

- **Basic authentication with Windows.** If your users are already in active directory, or local machine accounts, consider using basic authentication. Use transport security to secure the communication channel and protect your credentials.
- **Username authentication with Custom Store.** If your users are in a custom store, consider using user name authentication with a custom validator in order to validate user credentials against your custom store. Unlike the other scenarios, you will have to write custom code to validate your user's credentials. Use message or mixed mode security to protect your users' credentials.
- **Certificate authentication with certificates.** If your clients are partners or mobile clients connecting over VPN in a peer-to-peer authentication scenario, consider using certificate authentication. If your users have Windows accounts in your domain you can map the certificates to Windows accounts and enable authorization checks based on Windows roles. Certificate authentication requires that you manage certificates, however, it allows seamless authentication for clients who are outside your firewall. Use transport security to secure the communication channel and protect your credentials.

Intranet

- **Username authentication with SQL Membership Provider.** If your users are not in active directory, consider SQL Membership Provider. This will give you a store that can be easily deployed and created. Use transport security to secure the communication channel and protect your credentials.
- **Windows authentication with windows**. If your users are already in active directory or local machine accounts, consider using windows authentication to leverage this infrastructure. Windows authentication will give you also the benefits of using Windows roles for authorization checks. Use transport security to secure the communication channel and protect your credentials. Consider that local machine accounts configure a authentication with NTLM protocol, which is prone to brute force attacks. For more secure peer to peer authentication, consider using certificate authentication.
- **Username authentication with Custom Store.** If your users are in a custom store, consider using user name authentication with a custom validator in order to validate user credentials against your custom store. Unlike the other scenarios, you will have to write custom code to validate your user's credentials. Use message or mixed mode security to protect your users' credentials.
- **Certificate authentication with certificates.** If your clients are partners or mobile clients connecting over VPN in a peer-to-peer authentication scenario, consider using certificate authentication. If your users have Windows accounts in your domain you can map the certificates to Windows accounts and enable authorization checks based on Windows roles. Certificate authentication requires that you manage certificates, however, it allows seamless authentication for clients who are outside your firewall. Use transport security to secure the communication channel and protect your credentials.

Additional Resources

For more information on choosing a security mode, see "When should I use message security vs. transport security?" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=When%20should%20I%20use%20message%20security%20vs.%20transport%20security%3f&referringTitle=Questions%20and%20Answers

How do I decide on an authorization strategy?

Know your authorization options and choose the most appropriate for your scenario. First decide if you want to use resource-based or role-based authorization. Resource-based authorization uses ACLs on the resource to authorize the original caller. Role-based allows you to authorize access to service operations or resources based upon the group a user is in.

- If you choose to use role-based authorization then you can store your roles in Windows groups or in ASPNET roles.
- If you are using Active Directory then consider using Windows groups based on ease of maintenance and the fact you maintain both roles and credentials in the Active Directory store. If you are not using Active Directory, consider using ASPNET roles and the ASP.NET Role Provider.

Your authorization strategy may also be influenced by your choice of authentication type:

Resource-based authorization

- If you are using certificates authentication you will need to map certificates to Windows groups.
- If you are using username authentication you will need to perform protocol transition.
- Windows authentication will work with resource-based authorization by default.
- Basic authentication will work with resource-based authorization by default.
- **Note:** You need to impersonate for resource-based authorization.

Role-based authorization

- If you are using certificates authentication you will need to map certificates to Windows groups.
- If you are using username authentication with Windows groups, you will need to perform protocol transition.
- Username authentication will work with ASPNET roles by default.
- Windows authentication will work with Windows groups by default.
- Basic authentication will work with Windows groups by default.

Additional Resources

- For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx

- For more information on protocol transition, see http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=What%20is%20protocol%20transition%3f&referringTitle=Questions%20and%20Answers

When should I use message security vs. transport security?

Message security encrypts each individual message to protect sensitive data. *Transport security* secures the end-to-end network connection to protect the network traffic.

Use the following criteria to decide whether to use transport security:

- **Point-to-point** – Transport security supports point-to-point communication and does not support intermediary scenarios or protocol transition.
- **Streaming** – Transport security can support streaming data scenarios.
- **Binding limitations** – Transport security does not work with **wsDualHttpBinding**.
- **Authentication limitations** – Transport security does not work with negotiation, username, or Kerberos direct authentication.
- **Performance** – Transport security may provide better performance than message security.

Use the following criteria to decide whether to use message security:

- **Intermediaries** – Message security supports scenarios with intermediaries or protocol transition.
- **Encryption flexibility** – Message security allows you to encrypt part of a message while leaving other parts in cleartext format.
- **Binding limitations** – Message security does not work with **netNamedPipeBinding**.
- **Secure conversations** – Secure conversations only works with message security.
- **Authentication limitations** – Message security does not work with Basic or Digest authentication

Additional Resources

- For more information on message protection, see “Message Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For more information on choosing a transport, see “Choosing a Transport” at http://msdn2.microsoft.com/en-us/library/ms733769.aspx

How do I use my existing Active Directory infrastructure?

If your users are in Active Directory, consider using Windows, Username or Basic authentication. All of these authentication schemes can be mapped to users in Active Directory.

- **Windows authentication**. This authentication scheme will default to users in Active Directory. It has the benefits of providing support for message security without requiring to install certificates. It also provides support for transport security with netTcpBinding withour requiring to install certificates. It cannot cross firewall boundaries

- **Basic authentication.** Basic authentication maps to users in Active Directory. Transport security will be required to protect user credentials. It has the benefits of crossing firewall boundaries.
- **Username authentication.** Client username/password information is automatically mapped to Windows user accounts. Message security will be required to protect credentials. It has the benefits of crossing firewall boundaries.

What bindings should I use over the Internet?

The following bindings work well over the Internet, depending on your scenario:

- If your service is interacting with WCF clients, use **wsHttpBinding** because this binding provides the best **WS-*** interoperability features, including depending on your scenario, **WS-Addressing**, and **WS-AtomicTransaction**. The combination of features offered by **wsHttpBinding** provides the most reliable connection offered by WCF over the Internet.
- If your service is interacting with ASP.NET Web Services (ASMX) clients, you must use **basicHttpBinding** because it is the only WCF binding that supports ASMX clients.
- Clients and services that require full-duplex communication should use **wsDualHttpBinding** because it is the only binding that supports full-duplex.
- If your service interacts with Web Services Enhancements (WSE) clients, you must use **customBinding**. The service must use a custom binding to be compatible with the August 2004 version of **WS-Addressing**.

What bindings should I use over the intranet?

Although you can use any binding over an intranet, **netTcpBinding** will provide the best throughput performance. On an intranet, you generally do not need to be as concerned about the connection going down as with an Internet connection, so some of the **WS-*** advantages that are supplied with **wsHttpBinding** may not be as necessary.

When should I use resource-based authorization vs. roles-based authorization?

Use resource-based authorization when you want to allow access to ACL-secured resources based on the original caller. Use roles-based authorization when you want to manage multiple users in groups to authorize on or within operations based on business logic.

When should I impersonate the original caller?

Impersonation allows your service to run in a least-privileged security context and only elevate privileges when it is necessary. Use impersonation when the service needs to use the original caller's credentials to access resources. Without impersonation, the service will authorize resource access based on the least-privileged account under which it is running.

The most common impersonation scenarios in WCF are:

- Resource-based authorization on any resource using an ACL.

- Role-based authorization in which the user's security context will be authorized downstream in another component.
- Database authorization based on original caller.

Additional Resources

For more information, see "How To: Impersonate the Original Caller in WCF Calling from Windows Forms" at
http://www.codeplex.com/WCFSecurityGuide/Wiki/View.aspx?title=How%20To%20-%20Impersonate%20the%20Original%20Caller%20in%20WCF%20calling%20from%20Windows%20Forms&referringTitle=How%20Tos

When should I flow the original caller's identity?

You flow the original caller's identity when some action needs to be done on the client's behalf. There are two methods that this can be accomplished: Impersonation, which allows the service to act as the client while doing some business operation; that can be access to a resource or backend system. Delegation is another way of flowing the caller's identity. In this case the service flows impersonation capabilities to a backend service. The backend service is passed the caller identity and it acts on behalf of the original caller. So delegation includes another hop to another service with machine boundary, which will impersonate the original caller. Impersonation and delegation are features with Kerberos based authentication. They require a windows identity.

Additional Resources

For more information, see "Delegation and Impersonation with WCF" at
http://msdn2.microsoft.com/en-us/library/ms730088.aspx

How do I migrate to WCF from an ASMX Web service?

In order to migrate to WCF from an ASMX web service, you will have to update the service contract definition, elements, attributes, and configuration definitions. You can keep the exposed methods, properties, and business logic contained within the ASMX service.

Note the following considerations:

- WCF requires .NET Framework version 3.0 or above.
- You can leave your ASMX clients as-coded after you upgrade the service by using **basicHttpBinding** for the service. The **basicHttpBinding** binding is compatible with ASMX clients.
- The client proxy is nearly identical between ASMX and WCF, but should be regenerated using SvcUtil.exe to be sure that it is up to date.
- Because WCF configuration is different from ASMX configuration, it needs to be changed (e.g., for the app.config or web.config file).
- WCF implements a [**ServiceContract]** attribute to define the service interface, and an [**OperationContract]** attribute for each method or property exposed. Web services

implement a [**WebService()]** attribute to define the service and a [**WebMethod()]** attribute for each method or property exposed.

For example:

ASMX Web Service

```
[WebService()]
public class ThisService : WebService
{
  [WebMethod()]
  public String Hello(String inputName)
  {
    return "Hello, " + inputName;
  }
```

WCF Service:

```
[ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples")]
public interface IThisService
{
    [OperationContract]
    string Hello(String inputName);
}

public class ThisService : IThisService
{
  public String Hello(String inputName)
  {
    return "Hello, " + inputName;
  }
```

How do I migrate to WCF from a COM application?

WCF offers the advantage of a more distributable Service Oriented Architecture (SOA). Your application can call the service objects on the same computer, or across different computers on an intranet or across the Internet. The service will be available to other applications as well.

To migrate a COM object to WCF:

1. Install .NET Framework 3.0 or above.
2. Create a WCF service project. Move your code from the legacy COM application to the managed-code WCF service project.
3. Add **WCF [ServiceContract]** and **[OperationContract]** attributes to define the service contract and operation contract elements. See “How do I migrate to WCF from an ASMX Web service?” above for more information on how to specify the **[ServiceContract]** and **[OperationContract]** attributes.
4. Fill in the WCF service configuration file. If hosting in Internet Information Services (IIS), configure in the web.config file. If self-hosting, configure in the app.config file. If your application is calling the WCF service on the same computer, the Named Pipes binding will offer the best performance.

How do I migrate to WCF from a DCOM application?

WCF will eliminate the difficulties of DCOM reliability and configuration. WCF offers the advantages of a more distributable Service Oriented Architecture (SOA). Your application can call the service objects on the same computer, or across different computers on an Intranet or across the Internet. The service will be available to other applications as well.

To migrate a DCOM object to WCF:

1. Install .NET Framework 3.0 or above.
2. Create a WCF service project. Move your code from the legacy DCOM application to the managed-code WCF service project.
3. Add WCF **[ServiceContract]** and **[OperationContract]** attributes to define the service contract and operation contract elements. See "How do I migrate to WCF from an ASMX Web service?" above for more information on how to specify the **[ServiceContract]** and **[OperationContract]** attributes.
4. Fill in the WCF service configuration file. If hosting in IIS, configure in the web.config file. If self-hosting, configure in the app.config file. If your application is calling the WCF service on the same computer, the Named Pipes binding will offer the best performance.

How do I migrate to WCF from a WSE application?

You can keep the exposed methods, properties, and business logic contained within the WSE service. Update the service contract definition, elements, attributes, and configuration definitions to update the service from WSE to WCF. Note the following considerations:

- WCF requires .NET Framework 3.0 or above.
- The client proxy is nearly identical between WSE and WCF, but should be regenerated using SvcUtil.exe to be sure that it is up to date.
- WCF requires a different configuration than WSE (e.g., for the app.config or web.config file).
- WCF implements a [**ServiceContract]** attribute to define the service interface, and an [**OperationContract]** attribute for each method or property exposed. Web services implement a [**WebService()]** attribute to define the service and a [**WebMethod()]** attribute for each method or property exposed.

For example:

ASMX Web Service

```
[WebService()]
public class ThisService : WebService
{
  [WebMethod()]
  public String Hello(String inputName)
  {
    return "Hello, " + inputName;
  }
```

WCF Service:

```
[ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples")]
public interface IThisService
```

```
{
    [OperationContract]
    string Hello(String inputName);
}

public class ThisService : IThisService
{
  public String Hello(String inputName)
  {
    return "Hello, " + inputName;
  }
```

WCF services are wire-level compatible with WSE clients when configured to use the August 2004 version of **WS-Addressing**. WCF services that interact with WSE clients require a custom binding configuration to facilitate the **WS-Addressing** compatibility.

Additional Resources

For more information on migrating from WSE, see "How to: Configure WCF Services to Interoperate with WSE 3.0 Clients" at http://msdn2.microsoft.com/en-us/library/ms730049.aspx

Auditing and Logging

- **What WCF service security events should be logged?**
- **How do I enable logging and auditing in WCF?**
- **How do I stop my service if there has been an auditing failure?**
- **How do I log important business events in WCF?**
- **How do I implement log throttling in WCF?**
- **How do I use the health monitoring feature with WCF?**
- **How do I protect my log files?**
- **How do I pass user identity information in a message for auditing purpose?**

What WCF service security events should be logged?

In order to improve auditing and increase your chances of discovering an attack on your system, the following security events should be logged:

- Use WCF auditing to log authentication successes and failures.
- Use WCF auditing to log authorization successes and failures.
- Use WCF message logging to log malformed Simple Object Access Protocol (SOAP) messages.
- Use ASP.NET health monitoring in conjunction with your input and data validation routines to log malformed parameters and schema in incoming WCF messages.

How do I enable logging and auditing in WCF?

There are three key technologies you can use to audit and log operations in your WCF service:

1. Use WCF auditing to audit security events such as authentication and authorization failures.

2. Use WCF message logging to log malformed SOAP messages or to trace incoming messages.
3. Use ASP.NET health monitoring to provide custom logging; for instance, to log the occurrence of malformed input parameters or other significant occurrences.

Enable WCF Auditing in your config file with a **serviceSecurityAudit** service behavior as follows:

```
…
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
        <serviceSecurityAudit auditLogLocation="Application"
serviceAuthorizationAuditLevel="SuccessOrFailure"
         messageAuthenticationAuditLevel="SuccessOrFailure" />
    </behavior>
  </serviceBehaviors>
</behaviors>
…
```

Enable message logging in your config file by creating a **ServiceModelMessageLoggingListener** and **System.ServiceModel.MessageLogging** source and then add **MessageLogging** under the diagnostics node as follows:

```
…
<configuration>
<system.diagnostics>
  <sources>
   <source name="System.ServiceModel.MessageLogging"
switchValue="Warning, ActivityTracing">
    <listeners>
     <add type="System.Diagnostics.DefaultTraceListener"
name="Default">
      <filter type="" />
     </add>
     <add name="ServiceModelMessageLoggingListener">
      <filter type="" />
     </add>
    </listeners>
   </source>
  </sources>
  <sharedListeners>
   <add
initializeData="c:\inetpub\wwwroot\WCFService\web_messages.svclog"
    type="System.Diagnostics.XmlWriterTraceListener, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    name="ServiceModelMessageLoggingListener"
traceOutputOptions="Timestamp">
    <filter type="" />
   </add>
  </sharedListeners>
</system.diagnostics>
</configuration>
…
…
<system.serviceModel>
  <diagnostics>
```

```
      <messageLogging logEntireMessage="false" logMalformedMessages="true"
       logMessagesAtServiceLevel="false"
    logMessagesAtTransportLevel="true" />
      </diagnostics>
    …
```

Additional Resources

- For more information on auditing, see "Auditing Security Events" at http://msdn2.microsoft.com/en-us/library/ms731669.aspx
- For more information see "How to: Audit Windows Communication Foundation Security Events" at http://msdn2.microsoft.com/en-us/library/ms734737.aspx

How do I stop my service if there has been an auditing failure?

If non-repudiation is more important to you than ensuring that your service is always running and providing service, you can set the **SuppressAuditFailure** element to false to throw an exception when there has been an auditing failure. By default, this property is set to true, which means your service can continue running even after auditing has failed and no additional events are being logged.

The following configuration code shows how to stop the service if there is an audit failure:

```
<configuration>
  <system.serviceModel>
    <behaviors>
      <behavior>
        <serviceSecurityAudit
                  auditLogLocation="Application"
                  suppressAuditFailure="false"
                  serviceAuthorizationAuditLevel="Failure"
                  messageAuthenticationAuditLevel="SuccessOrFailure" />
      </behavior>
    </behaviors>
  </system.serviceModel>
</configuration>
```

You can also shut down the Windows system immediately when there is a failure to audit events by enabling the following Windows Local Security Setting property:

```
Audit: Shut down system immediately if unable to log security audits
```

To set the property, open the Control Panel and then under **Administrative Tools**, open the **Local Security Settings** dialog box . Click **Local Policies** and then click **Security Options** to find the above property.

Additional Resources

- For more information on auditing, see "Auditing Security Events" at http://msdn2.microsoft.com/en-us/library/ms731669.aspx

- For additional information on auditing, see "How to: Audit Windows Communication Foundation Security Events" at http://msdn2.microsoft.com/en-us/library/ms734737.aspx

How do I log important business events in WCF?

You can use ASP.NET Health Monitoring to log sensitive operations. For instance, you could track usage of methods that relate to financial transactions or access to sensitive data.

To instrument your application with Health Monitoring:

1. Create a class library and a class that inherits from **WebSuccessAuditEvent** that displays some business information, such as bank account transactions.

```
using System.Web.Management;

 public class MyAccountTransactions : WebSuccessAuditEvent
    {

        public MyAccountTransactions(string msg, object eventSource,
int eventCode)
            : base(msg, eventSource, eventCode)
        {
            // Do some processing
        }

        public MyAccountTransactions(string msg, object eventSource,
int eventCode, int eventDetailCode)
            : base(msg, eventSource, eventCode, eventDetailCode)
        {
            // Do some processing
        }
   public override void FormatCustomEventDetails(WebEventFormatter
formatter)
        {
            base.FormatCustomEventDetails(formatter);

            // Display some transaction information
            formatter.AppendLine("Some Credit\Debit information");
      }
```

2. Configure your WCF service for health monitoring.

```
      …
      <system.web>
            <healthMonitoring>
                  <eventMappings>
                        <add name="SomeTransactionInfo"
      type="MyEventLibrary.MyAccountTransactions, MyEventLibrary"/>
                  </eventMappings>
                  <rules>
                        <add name="Custom event"
      eventName="SomeTransactionInfo" provider="EventLogProvider"
      minInterval="00:00:01"/>
                  </rules>
            </healthMonitoring>
      </system.web>
      …
```

3. Instrument the WCF service by raising the custom event in a service contract.

```
public string InvokeBusinessEvent()
    {
        MyAccountTransactions obj = new
MyAccountTransactions("Invoking some business operation", this,
WebEventCodes.WebExtendedBase + 1);
        obj.Raise();
        return "Displaying some transaction details";
    }
```

4. Verify the service events in the Event Log after calling the service method from a test client.

Additional Resources

- For more information on health monitoring, see "How To: Use Health Monitoring in ASP.NET 2.0" at http://msdn2.microsoft.com/en-us/library/ms998306.aspx
- For additional information on health monitoring, see "ASP.NET Health Monitoring Overview" at http://msdn.microsoft.com/en-us/library/bb398933.aspx
- For more information on auditing, see "Auditing Security Events" at http://msdn2.microsoft.com/en-us/library/ms731669.aspx
- For additional information on auditing, see "How to: Audit Windows Communication Foundation Security Events" at http://msdn2.microsoft.com/en-us/library/ms734737.aspx

How do I implement log throttling in WCF?

If you are using WCF message logging, you can specify the maximum messages to log as well as the maximum size message to log. These attributes can be found in the <messagelogging> element:

- **maxMessagesToLog –** Allows you to limit the size of the log file, by reducing the total number of messages in the log. This setting can be used to reduce the chances of a denial of service (DoS) attack on your log but can be used by an attacker to fill up the log and conceal their intrusion.
- **maxSizeOfMessagesToLog –** Allows you to limit the size of the log file, by restricting very large messages from being logged. This setting can be used to reduce the chances of a DoS attack on your log but could potentially be used by an attacker to conceal their intrusion by ensuring that certain messages are not logged.

When the message limit is reached, a trace at the Information level is produced, and all message logging activities stop.

Additional Resources

For more information on log throttling, see "Configuring Message Logging" at http://msdn2.microsoft.com/en-us/library/ms730064.aspx

How do I use health monitoring feature with WCF?

To use the health monitoring feature in WCF, configure your WCF service by performing the following steps:

1. Create a custom health monitoring event.
2. Configure your WCF service for health monitoring.
3. Instrument an application to raise a custom event.

The following example shows how to implement the above steps for health monitoring in a WCF service:

1. Create a custom user management Web event, by creating a class library and then creating a class that inherits from **WebAuditEvent**.

```
using System.Web.Management;

 public class MyEvent : WebAuditEvent
 {
        public MyEvent(string msg, object eventSource, int eventCode)
            : base(msg, eventSource, eventCode)
        {
        }

        public MyEvent(string msg, object eventSource, int eventCode,
int eventDetailCode)
            : base(msg, eventSource, eventCode, eventDetailCode)
        {
        }
        public override void FormatCustomEventDetails(WebEventFormatter
formatter)
        {
            base.FormatCustomEventDetails(formatter);

            // Display some custom event message
            formatter.AppendLine("Some Critical Event Fired");
        }
}
```

2. Configure your WCF service for health monitoring.

```
…
<system.web>
      <healthMonitoring>
            <eventMappings>
                  <add name="Some Custom Event"
type="MyEventLibrary.MyEvent, MyEventLibrary"/>
            </eventMappings>
            <rules>
                  <add name="Custom event" eventName="Some Custom
Event" provider="EventLogProvider" minInterval="00:00:01"/>
            </rules>
      </healthMonitoring>
</system.web>
…
```

3. Instrument the WCF service by raising the custom event in a service contract.

```
[OperationContract]
string InvokeCriticalEvent();

public string InvokeCriticalEvent()
    {
        MyEvent obj = new MyEvent("Invoking Some Custom Event", this,
WebEventCodes.WebExtendedBase + 1);
        obj.Raise();
        return "Critical event invoked";
    }
```

4. Verify the service events in the Event Log after calling the service method from a test client.

Additional Resources

- For more information on health monitoring, see "How To: Use Health Monitoring in ASP.NET 2.0" at http://msdn2.microsoft.com/en-us/library/ms998306.aspx
- For additional information on health monitoring, see "ASP.NET Health Monitoring Overview" at http://msdn.microsoft.com/en-us/library/bb398933.aspx
- For more information on auditing, see "Auditing Security Events" at http://msdn2.microsoft.com/en-us/library/ms731669.aspx
- For additional information on auditing, see "How to: Audit Windows Communication Foundation Security Events" at http://msdn2.microsoft.com/en-us/library/ms734737.aspx

How do I protect my log files?

Protect your log files using Windows access control lists (ACLs) with restricted access. If you log events to Microsoft SQL Server® or to some custom event sink, use appropriate access controls to limit access to the event data. For example, grant write access to the account or accounts used by your application, grant full control to administrators, and grant read-only access to operators.

How to I pass user identity information in a message for auditing purpose?

You might want to pass user identity information in a message in cases where you are not impersonating the caller. This could be useful when auditing business-critical operations such as financial transactions.

To pass user identity information in a custom event for auditing:

1. Create a custom user management Web event, by creating a class library and then creating a class that inherits from **WebAuditEvent**.

```
using System.Web.Management;

 public class MyEvent : WebAuditEvent
    {

        public MyEvent(string msg, object eventSource, int eventCode)
            : base(msg, eventSource, eventCode)
```

```
        {
              // Obtain the HTTP Context and store authentication
      details

              userID = HttpContext.Current.User.Identity.Name;
              authType =
      HttpContext.Current.User.Identity.AuthenticationType;
              isAuthenticated =
      HttpContext.Current.User.Identity.IsAuthenticated;
        }

        public MyEvent(string msg, object eventSource, int eventCode,
int eventDetailCode)
            : base(msg, eventSource, eventCode, eventDetailCode)
        {
              // Obtain the HTTP Context and store authentication
      details

              userID = HttpContext.Current.User.Identity.Name;
              authType =
      HttpContext.Current.User.Identity.AuthenticationType;
              isAuthenticated =
      HttpContext.Current.User.Identity.IsAuthenticated;
        }
   public override void FormatCustomEventDetails(WebEventFormatter
formatter)
        {
            base.FormatCustomEventDetails(formatter);

            // Display user identity information in the event message
            formatter.AppendLine("User ID: " + userID);
            formatter.AppendLine("Authentication Type: " + authType);
            formatter.AppendLine("User Authenticated: " +
isAuthenticated.ToString());
      }
```

2. Configure your WCF service for health monitoring.

```
…
<system.web>
      <healthMonitoring>
            <eventMappings>
                  <add name="Some Custom Event"
type="MyEventLibrary.MyEvent, MyEventLibrary"/>
            </eventMappings>
            <rules>
                  <add name="Custom event" eventName="Some Custom
Event" provider="EventLogProvider" minInterval="00:00:01"/>
            </rules>
      </healthMonitoring>
</system.web>
…
```

3. Instrument the WCF Service by raising the custom event in a service contract.

```
[OperationContract]
string InvokeEvent();
```

```
public string InvokeEvent()
    {
        MyEvent obj = new MyEvent("Invoking Custom Event – User Info",
this, WebEventCodes.WebExtendedBase + 1);
        obj.Raise();
        return "Event showing User information";
    }
```

4. Verify the service events in the Event Log after calling the service method from a test client.

```
Event code: 100001
Event message: Invoking Custom Event – User Info
…
Application information:
    Application domain: /LM/w3svc/1/ROOT/HealthMonitoring-7-
127656015969887178
…
Custom event details:
User ID: DomainName\UserName
    Authentication Type: Negotiate
    User Authenticated: True
```

Additional Resources

- For more information on auditing, see "Auditing Security Events" at http://msdn2.microsoft.com/en-us/library/ms731669.aspx
- For additional information on auditing, see "How to: Audit Windows Communication Foundation Security Events" at http://msdn2.microsoft.com/en-us/library/ms734737.aspx

Authentication

- **How do I decide on an authentication strategy in WCF?**
- **When should I use the SQL Server membership provider?**
- **How do I authenticate against Active Directory?**
- **How do I authenticate against a SQL store?**
- **How do I authenticate against a custom store?**
- **How do I protect passwords in my user store?**
- **How do I use certificate authentication with X.509 certificates?**
- **What is the most common authentication scenario for intranet applications?**
- **What is the most common authentication scenario for Internet applications?**
- **How do I support authentication for multiple client types?**
- **What is federated security?**
- **How do I send credentials in the message when I am using transport security?**
- **How do I avoid cleartext passwords?**

How do I decide on an authentication strategy in WCF?

Decide your authentication strategy based on your user credential store location and the location of your clients on the Internet or intranet.

Internet

- **Username authentication with SQL Membership Provider**. If your users are not in active directory, consider SQL Membership Provider. This will give you a store that can be easily deployed and created. Configure message or mixed mode security to protect your users' credentials.
- **Basic authentication with Windows.** If your users are already in active directory, or local machine accounts, consider using basic authentication. Use transport security to secure the communication channel and protect your credentials.
- **Username authentication with Custom Store.** If your users are in a custom store, consider using user name authentication with a custom validator in order to validate user credentials against your custom store. Unlike the other scenarios, you will have to write custom code to validate your user's credentials. Use message or mixed mode security to protect your users' credentials.
- **Certificate authentication with certificates.** If your clients are partners or mobile clients connecting over VPN in a peer-to-peer authentication scenario, consider using certificate authentication. If your users have Windows accounts in your domain you can map the certificates to Windows accounts and enable authorization checks based on Windows roles. Certificate authentication requires that you manage certificates, however, it allows seamless authentication for clients who are outside your firewall. Use transport security to secure the communication channel and protect your credentials.

Intranet

- **Username authentication with SQL Membership Provider.** If your users are not in active directory, consider SQL Membership Provider. This will give you a store that can be easily deployed and created. Use transport security to secure the communication channel and protect your credentials.
- **Windows authentication with windows**. If your users are already in active directory or local machine accounts, consider using windows authentication to leverage this infrastructure. Windows authentication will give you also the benefits of using Windows roles for authorization checks. Use transport security to secure the communication channel and protect your credentials. Consider that local machine accounts configure a authentication with NTLM protocol, which is prone to brute force attacks. For more secure peer to peer authentication, consider using certificate authentication.
- **Username authentication with Custom Store.** If your users are in a custom store, consider using user name authentication with a custom validator in order to validate user credentials against your custom store. Unlike the other scenarios, you will have to write custom code to validate your user's credentials. Use message or mixed mode security to protect your users' credentials.
- **Certificate authentication with certificates.** If your clients are partners or mobile clients connecting over VPN in a peer-to-peer authentication scenario, consider using

certificate authentication. If your users have Windows accounts in your domain you can map the certificates to Windows accounts and enable authorization checks based on Windows roles. Certificate authentication requires that you manage certificates, however, it allows seamless authentication for clients who are outside your firewall. Use transport security to secure the communication channel and protect your credentials.

Additional Resources

For more information on authentication, see "Authentication" at http://msdn2.microsoft.com/en-us/library/ms733082.aspx

When should I use the SQL Server Membership provider?

You should use SQL Server Membership provider when users are not in active directory and you need a user store that can be easily created and deployed. As additional benefits, you can use authentication schemes in WCF that can cross firewall boundaries. You will need to use transport security to secure the communication channel to protect user credentials.

Additional Resources

For more information on authentication, see "Authentication" at http://msdn2.microsoft.com/en-us/library/ms733082.aspx

How do I authenticate against Active Directory?

If your users are in Active Directory, consider using Windows, Username or Basic authentication. All those authentication schemes can be mapped to users in Active Directory.

- **Windows authentication –** This authentication scheme will default to users in Active Directory. It has the benefits of providing support for message security without requiring to install certificates. It also provides support for transport security with netTcpBinding withour requiring to install certificates. It cannot cross firewall boundaries
- **Basic authentication –** Basic authentication maps to users in Active Directory. Transport security will be required to protect user credentials. It has the benefits of crossing firewall boundaries.
- **Username authentication** – Client username/password information is automatically mapped to Windows user accounts. Message security will be required to protect credentials. It has the benefits of crossing firewall boundaries. The following example configures the binding for username authentication and message security:

Additional Resources

For more information on authentication, see "Authentication" at http://msdn2.microsoft.com/en-us/library/ms733082.aspx

How do I authenticate against a SQL store?

To use username authentication with a SQL Server database, you can configure your application to use the ASP.NET membership feature.

To configure the membership provider, perform the following steps:

1. Configure your SQL Server database for membership. From a Microsoft Visual Studio® 2008 command prompt, run the following command:
   ```
   aspnet_regsql -S .\SQLExpress -E -A m -d <<YourDatabaseName>>
   ```

   In this command:
   - **-S** specifies the server, which is (.\SQLExpress) in this example.
   - **-E** specifies to use Windows authentication to connect to SQL Server.
   - **-A m** specifies to add only the membership feature. For simple authentication against a SQL Server user store, only the membership feature is required.
   - **-d** specifies the SQL Server database name. If this option is not used, a default aspnetdb database will be created.

   For a complete list of the commands, run **Aspnet_regsql /?**
2. Modify your Web.config file in your WCF service application by adding the following sections:
   ```
   <connectionStrings>
     <add name="MyLocalSQLServer"
          connectionString="Initial Catalog=<<YourDatabaseName>>;
         data source=.\sqlexpress;Integrated Security=SSPI;" />
   </connectionStrings>

   …
   <system.web>
     ...
     <membership defaultProvider="MySqlMembershipProvider" >
       <providers>
         <clear/>
         <add name="MySqlMembershipProvider"
              connectionStringName="MyLocalSQLServer"
              applicationName="MyAppName"
              type="System.Web.Security.SqlMembershipProvider" />
       </providers>
     </membership>
   </system.web>
   …
   ```

3. Configure the service to use username authentication.
   ```
   …
   <bindings>
     <wsHttpBinding>
       <binding name="wsHttpEndpointBinding">
         <security>
           <message clientCredentialType="UserName" />
         </security>
       </binding>
     </wsHttpBinding>
   </bindings>
   ```

4. Configure the service to use the membership provider.
   ```
   <behaviors>
   ```

```
  <serviceBehaviors>
    <behavior name="ServiceBehavior">

      <serviceCredentials>
        <userNameAuthentication
userNamePasswordValidationMode="MembershipProvider"
          membershipProviderName="MySqlMembershipProvider" />
      </serviceCredentials>

    </behavior>
  </serviceBehaviors>
</behaviors>
…
```

Additional Resources

For more information, see "How To – Use Username Authentication with the SQL Server Membership Provider and Message Security in WCF from Windows Forms" at http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=How%20To%20-%20Use%20Username%20Authentication%20with%20the%20SQL%20Membership%20Provider%20and%20Message%20Security%20in%20WCF%20from%20Windows%20Forms&referringTitle=How%20Tos

How do I authenticate against a custom store?

To use a custom user/identity store with username authentication, configure your application to use the username authentication with a custom username and password validator. The custom validator will be configured in a service behavior and implemented in a class library. The username and password validator will be used by your service to authenticate your users based on your custom user store.

The following configuration snippet shows how to configure a custom validator for your WCF service:

```
<serviceCredentials>
      <userNameAuthentication userNamePasswordValidationMode="Custom"
      customUserNamePasswordValidatorType="MyUserNamePasswordValidator,
      Host"/>
      <serviceCertificate findValue="CN=FabrikamEnterprises"/>
</serviceCredentials>
```

The following code snippet shows how to implement a custom username and password validator:

```
using System;
using System.Collections.Generic;
using System.IdentityModel.Selectors;
using System.IdentityModel.Tokens;
using System.Text;

namespace Validator
{
    public class MyUserNamePasswordValidator :
UserNamePasswordValidator
    {
```

```
            public override void Validate(string userName, string password)
            {
                Console.Write("\nValidating username, {0}, and password,
    {1} ... ", userName, password);
                if ((string.Compare(userName, "don", true) != 0) ||
    (string.Compare(password, "hall", false) != 0))
                {
                    throw new SecurityTokenException("Unknown user.");
                }
                Console.Write("Done: Credentials accepted. \n");
            }
        }
    }
```

How do I protect passwords in my user store?

Protect passwords in your user store by storing one-way password hashes with a salt. Generate the hash from a combination of the password and a random salt value. Use an algorithm such as SHA256. If your credential store is compromised, the salt value helps to slow an attacker who is attempting to perform a dictionary attack.

How do I use certificate authentication with X.509 certificates?

Configure your service to use **wsHttpBinding** with message security and **clientCredentialType** set to Certificate, as follows:

```
    <wsHttpBinding>
      <binding name="WSHttpBinding_ICalculator">
        <security mode="Message">
          <message clientCredentialType="Certificate" />
        </security>
      </binding>
    </wsHttpBinding>
```

You can map an X509 certificate to Windows account by setting the **mapClientCertificateToWindowsAccount** property to true. By default, when using the certificate client credential type on bindings, the certificate is not mapped to Windows accounts. You can override this behavior by using the **mapClientCertificateToWindowsAccount** property as follows:

```
<serviceBehaviors>
  <behavior name="MyServiceBehaviorForWebHttp">

     <serviceCredentials>
      <clientCertificate>
       <authentication mapClientCertificateToWindowsAccount="true" />
      </clientCertificate>
     </serviceCredentials>

  </behavior>
</serviceBehaviors>
```

Additional Resources

- For more information on using WCF with certificates, see “Working with Certificates” at http://msdn.microsoft.com/en-us/library/ms731899.aspx

- For more information on mapping certificates to Windows accounts see, "Map certificates to user accounts" at http://technet2.microsoft.com/WindowsServer/f/?en/library/0539dcf5-82c5-48e6-be8a-57bca16c7e171033.mspx
- For more information on mapping certificates to Active Directory, see "Mapping Client Certificates with Directory Service Mapping" at http://technet2.microsoft.com/windowsserver/en/library/7cce4299-28f2-45fa-8730-4e0cbe3be8561033.mspx?mfr=true
- For more information on certificate-mapping strategies see, "Mapping Strategies" at http://technet2.microsoft.com/windowsserver/en/library/aa61c564-1599-4414-a12d-2f64786f6ec31033.mspx?mfr=true

What is the most common authentication scenario for intranet applications?

One or more clients connect to a WCF service using Windows authentication with users in Active Directory. The service uses transport security over the netTcpBinding and is hosted in a Windows service. This combination gives the benefits of leveraging existing Windows users and groups for authentication and authorization while choosing the security mode and binding that are likely to yield the best performance.

What is the most common authentication scenario for Internet applications?

One or more clients connect to a WCF service using username authentication with users accessed via the SqlMembershipProvider. The service uses message security over the wsHttpBinding and is hosted in IIS. This combination works well over the Internet since it can cross firewalls and doesn't assume your users are in Active Directory.

How do I support authentication for multiple client types?

You can supply multiple endpoints from within one service to support multiple client types. For example, you can have one endpoint with **basicHttpBinding** for legacy Web service client connections and one with **wsHttpBinding** for WCF clients located on the Internet.

What is federated security?

Federated security provides a layer of abstraction between your service and the authentication/authorization mechanism you choose. This enables collaboration across multiple systems, networks, and organizations in different trust realms. WCF provides support for building and deploying distributed systems that employ federated security.

Using federated security provides you the flexibility of providing one set of credentials to a user and converting it to another set of credentials; for instance, converting the certificate given by the client to a Security Assertions Markup Language (SAML) token. Federated security also gives you the flexibility to alter your internal security mechanisms; for example, the client can provide a username/password pair to replace the certificate.

For more information, see "Federation and Issued Tokens" at http://msdn.microsoft.com/en-us/library/ms731161.aspx

How do I send credentials in the message when I am using transport security?

In intermediary scenarios, you might want to send credentials in messages that are protected by transport security, allowing your user to be authenticated by a downstream system.

To send credentials in the message, set the **security mode** to **TransportWithMessageCredentials** and **clientCredentialType** to the type of credentials you want to include.

```
<wsHttpBinding>
   <binding name="WindowsClientOverwsHttp">
     <security mode="TransportWithMessageCredentials">
       <transport clientCredentialType="Windows" />
     </security>
   </binding>
</wsHttpBinding>
```

How do I avoid cleartext passwords?

Perform the following steps to avoid sending cleartext passwords over the network:

- If possible, remove the need for a password at all by specifying **ClientCredentialType**="Windows", **ClientCredentialType**="Certificate", or a custom token that does not require a password.
- If the user must enter a password, protect the password by specifying either <Security mode="Transport"> to secure the channel or <Security mode="Message"> to secure the messages. Do not specify <Security mode="None"> in the configuration as this will provide no communication security.

Authorization

- **How do I decide on an authorization strategy in WCF?**
- **What's the difference between resource-based, roles-based, and claims-based authorization?**
- **How do I use Windows groups for role authorization in WCF?**
- **How do I use the SQL Server role provider for ASP.NET role authorization in WCF?**
- **How do I use the Windows Token role provider for ASP.NET role authorization in WCF?**
- **How do I use the Authorization Store role provider for ASP.NET role authorization in WCF?**
- **What's the difference between declarative and imperative roles authorization?**
- **How do I restrict access to WCF operations to specific Windows users?**
- **How do I associate roles with a certificate?**
- **What is a service principal name (SPN)?**
- **How do I create a service principal name (SPN)?**

How do I decide on an authorization strategy in WCF?

Know your authorization options and choose the most appropriate for your scenario. First decide if you want to use resource-based or role-based authorization. Resource-based authorization uses ACLs on the resource to authorize the original caller. Role-based allows you to authorize access to service operations or resources based upon the group a user is in.

- If you choose to use role-based authorization then you can store your roles in Windows groups or in ASPNET roles.
- If you are using Active Directory then consider using Windows groups based on ease of maintenance and the fact you maintain both roles and credentials in the Active Directory store. If you are not using Active Directory, consider using ASPNET roles and the ASP.NET Role Provider.

Your authorization strategy may also be influenced by your choice of authentication type:

Resource-based authorization

- If you are using certificates authentication you will need to map certificates to Windows groups.
- If you are using username authentication you will need to perform protocol transition.
- Windows authentication will work with resource-based authorization by default.
- Basic authentication will work with resource-based authorization by default.
- **Note:** You need to impersonate for resource-based authorization.

Role-based authorization

- If you are using certificates authentication you will need to map certificates to Windows groups.
- If you are using username authentication with Windows groups, you will need to perform protocol transition.
- Username authentication will work with ASPNET roles by default.
- Windows authentication will work with Windows groups by default.
- Basic authentication will work with Windows groups by default.

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For more information on protocol transition, see http://www.codeplex.com/WCFSecurity/Wiki/View.aspx?title=What%20is%20protocol%20transition%3f&referringTitle=Questions%20and%20Answers

What’s the difference between resource-based, roles-based, and claims-based authorization?

Roles-based authorization is used to group users into groups (roles) and then set permissions on the role rather than on individual users. This eases management by allowing you to administer a smaller set of roles rather than a larger set of users.

Resource-based authorization sets permissions on the resource itself. For instance, you would set an ACL on a Windows resource and then use the identity of the original caller to determine access rights to the resource. If you use resource-based authorization in WCF, you will need to impersonate the original caller through the application layer (e.g., ASP.NET application), through the WCF service layer, and to the business logic code that is accessing the file resource.

Claims-based authorization provides additional layers of abstraction on your authorization strategy in order to make it easier to separate your authorization rules from the mechanism you use for authorization and authentication. For instance, you could authenticate a user with a certificate or with username/password credentials and then pass that claim-set to the service to determine access to resources. You create authorization policies that are used to generate a claim-set based on the authentication evidence presented by the user (e.g., username and password, certificate, Kerberos). The claim-set is then used by your service to determine what resources the original caller has access to.

Additional Resources

For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx

How do I use Windows groups for role authorization in WCF?

Map Windows groups to WCF service methods using the WCF **PrincipalPermission** attribute. Incoming client username credentials will be mapped to the associated Windows group. Service method access will be granted to a user only if he or she is a member of the group associated with the service method being called.

The following example demonstrates how the WCF service "Add" will only run for users belonging to the "CalculatorClients" Windows group.

```
// Only members of the CalculatorClients group can call this method.
[PrincipalPermission(SecurityAction.Demand, Role =
"CalculatorClients")]
public double Add(double a, double b)
{
    return a + b;
}
```

Additional Resources

For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx

How do I use the SQL Server role provider for ASP.NET role authorization in WCF?

The SQL Server role provider is configured in the WCF service web.config file. User and role information are stored in the Aspnetdb database. Incoming client connections supply a username and password for each method call. The SQL Server role provider matches the client

username/password combination to information in the Aspnetdb database and determines if the associated role matches the **PrincipalPermission** attribute role required in the method definition.

The following example configures the service to enable the SQL Server role provider:

```
<!-- Configure the Sql Role Provider -->
<roleManager enabled ="true"
             defaultProvider ="SqlRoleProvider" >
  <providers>
    <add name ="SqlRoleProvider"
         type="System.Web.Security.SqlRoleProvider"
         connectionStringName="SqlConn"
         applicationName="MembershipAndRoleProviderSample"/>
  </providers>
</roleManager>
<!-- Configure role based authorization to use the Role Provider -->
<serviceAuthorization principalPermissionMode ="UseAspNetRoles"
                      roleProviderName ="SqlRoleProvider" />
```

Service methods include a **PrincipalPermission** directive that specifies the required authorization access role required.

```
[PrincipalPermission(SecurityAction.Demand, Role = "Registered Users")]
public double Multiply(double n1, double n2)
  {
     double result = n1 * n2;
     return result;
  }
```

The following code shows how to do the authorization check-in code:

```
if (Roles.IsUserInRole(@"accounting"))
{
//authorized
}
Else
{
//authorization failed

}
```

The following client connection supplies a username and password to call the method:

```
// Set credentials to Alice
client.ClientCredentials.UserName.UserName = "Alice";
client.ClientCredentials.UserName.Password = "ecilA-123";

// Call the Add service operation.
double value1 = 100.00D;
double value2 = 15.99D;
double result = client.Multiply(value1, value2);
```

Additional Resources

For more information on the ASP.NET role provider, see “How to: Use the ASP.NET Role Provider with a Service” at http://msdn.microsoft.com/en-us/library/aa702542.aspx

How do I use the Windows Token role provider for ASP.NET role authorization in WCF?

If you use ASP.NET roles, consider using the ASP.NET role provider with the **AspNetWindowsTokenRoleProvider** name. This allows you to separate the design of the authorization from the implementation inside your service. If you decide to change the role provider, it will not affect the code needed to perform the authorization. When using imperative checks, consider using the role syntax instead of performing authorization checks with **WindowsPrincipal.isInrole**.

The following configuration example shows how to configure **AspNetWindowsTokenRoleProvider**:

```
<system.web>
…
<roleManager enabled="true"
             defaultProvider="AspNetWindowsTokenRoleProvider" />
…
</system.web>

    <system.serviceModel>
        <behaviors>
            <serviceBehaviors>
                <behavior name="BehaviorConfiguration">
                    <serviceAuthorization
principalPermissionMode="UseAspNetRoles"

roleProviderName="AspNetWindowsTokenRoleProvider" />
                    <serviceMetadata />
                </behavior>
            </serviceBehaviors>
        </behaviors>
```

The following code shows how to do the authorization check-in code:

```
if (Roles.IsUserInRole(@"accounting"))
{
//authorized
}
Else
{
//authorization failed

}
```

Additional Resources

For more information on the ASP.NET role provider, see “How to: Use the ASP.NET Role Provider with a Service” at http://msdn.microsoft.com/en-us/library/aa702542.aspx

How do I use the Authorization Store role provider for ASPNET role authorization in WCF?

You can integrate Authorization Manager into your WCF service to provide authorization for your users. Configure the Authorization Manager ASP.NET role provider for the application that

is hosting the WCF service. By configuring the ASP.NET role manager to use the **AuthorizationStoreRoleProvider**, you can use the role management API against an AzMan policy store.

Like other ASP.NET role providers, the Authorization Manager ASP.NET role provider is configured using the <providers> element. The following configuration example shows how to integrating Authorization Manager into a WCF service:

```
      <system.web>
          <roleManager enabled="true" defaultProvider="AzManRoleProvider">
            <providers>
              <add name="AzManRoleProvider"
                   type="System.Web.Security.AuthorizationStoreRoleProvider,
      System.Web, Version=2.0.0.0, Culture=neutral,
      publicKeyToken=b03f5f7f11d50a3a"
                   connectionStringName="AzManPolicyStoreConnectionString"
                   applicationName="MyWCFService"/>
            </providers>
          </roleManager>
      </system.web>

      <system.serviceModel>
        <behaviors>
            <serviceBehaviors>
                <behavior name="BehaviorConfiguration">
                    <serviceAuthorization
principalPermissionMode="UseAspNetRoles"
                        roleProviderName="AzManRoleProvider" />
                    <serviceMetadata />
                </behavior>
            </serviceBehaviors>
       </behaviors>
```

Additional Resources

For more information on the ASP.NET role provider, “How to: Use the ASP.NET Role Provider with a Service” at http://msdn.microsoft.com/en-us/library/aa702542.aspx

What is the difference between declarative and imperative roles authorization?

Imperative authorization supports fine-grained authorization choices based on business logic. Imperative roles-based authorization is written into your code and processed at run time. Imperative security is useful when the resource to be accessed or action to be performed is not known until run time or when finer-grained access control beyond the level of a code method is required.

Declarative authorization can be added to application code at design time by specifying required access for a particular method or class declared as an attribute on the operation. Declarative roles-based authorization is best for authorizing access to WCF at the operation level. Declarative authorization can be added to application code at design time by specifying

required access for a particular method or class declared as an attribute on the operation. Because attribute metadata is discoverable using reflection, it is easier to track the security principals that are allowed to access each method. Declarative authorization checks will work if you are using the ASP.NET role provider or Windows groups.

The following code example shows how to use the **PrinciplePermission** attribute to perform declarative authorization:

```
[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]
public double Add(double a, double b)
{
    return a + b;
}
```

The following is an example of an Imperative check using the ASP.NET role provider:

```
if (Roles.IsUserInRole(@"accounting"))
{
//authorized
}
Else
{
//authorization failed

}
```

Additional Resources

- For more information on authorization, see “Authorization” at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For information on the **Roles.IsUserInRole** method, see “Roles.IsUserInRole Method (String)” at http://msdn.microsoft.com/en-us/library/4z6b5d42.aspx

How do I restrict access to WCF operations to specific Windows users?

Perform the following steps to restrict access to specific Windows users:

1. Open the Computer Management Windows applet.
2. Create a Windows group that contains the specific Windows users to which you want to give access. For example, a group can be called “CalculatorClients”.
3. Configure your service to require ClientCredentialType = “Windows”. This will require clients to connect using Windows authentication.
4. Configure your service methods with the **PrincipalPermission** attribute to require connecting users be members of the CalculatorClients group.

```
// Only members of the CalculatorClients group can call this method.
[PrincipalPermission(SecurityAction.Demand, Role =
"CalculatorClients")]
public double Add(double a, double b)
{
    return a + b;
}
```

Additional Resources

For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx

How do I associate roles with a certificate?

Perform the following steps to associate roles with a certificate:

1. Configure your service to require ClientCredentialType = "Certificate". This will require clients to connect using certificate authentication.
2. Configure clients to supply a certificate. The incoming client requests will contain a certificate name and a unique thumbprint ID that can be matched against the service method **PrincipalPermission** certificate name and thumbprint.
3. Configure your service methods with the **PrincipalPermission** attribute that specifies the required certificate name and thumbprint. Only incoming requests that supply credentials that match the certificate will be allowed to access the method.

```
// Only a client authenticated with a valid certificate that has the
// specified subject name and thumbprint can call this method.
[PrincipalPermission(SecurityAction.Demand,
    Name = "CN=MyCertificate;
123456712345677E8E230FDE624F841B1CE9D41E")]
public double Multiply(double a, double b)
{
    return a * b;
}
```

Additional Resources

For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx

What is a service principal name (SPN)?

A service principal name (SPN) is the name by which a client uniquely identifies an instance of a service. The Kerberos authentication service can use an SPN to authenticate a service. When a client wants to connect to a service, it locates an instance of the service, composes an SPN for that instance, connects to the service, and presents the SPN for the service to authenticate.

Additional Resources

For more information on SPN, see "Setspn Overview" at http://technet2.microsoft.com/windowsserver/en/library/b3a029a1-7ff0-4f6f-87d2-f2e70294a5761033.mspx?mfr=true

How do I create a service principal name (SPN)?

You can use the SETSPN.EXE tool to associate an SPN with your service.
The following example creates an SPN that can be used to access the service. The service is named "CalcService," the computer is named "ComputerA," the domain is named "DomainA," and the service account is the ASPNET account.

```
C:\ Setspn –A CalcService/ComputerA  DomainA\ASPNET
```

Additional Resources

For more information on SPNs, see "Setspn Overview" at
http://technet2.microsoft.com/windowsserver/en/library/b3a029a1-7ff0-4f6f-87d2-f2e70294a5761033.mspx?mfr=true

Bindings

- **What is a binding?**
- **What bindings are available?**
- **Which bindings are best suited for the Internet?**
- **Which bindings are best suited for an intranet?**
- **How do I choose an appropriate binding?**

What is a binding?

A WCF service endpoint comprises an address, a binding, and a contract. Bindings define how clients can connect and communicate with your service. A binding includes definitions for the WS-* protocols used, the message encoding, and the transport protocol. For instance, the **wsHttpBinding** uses HTTP, XML 1.0 encoding, message security, reliable sessions, and transactions by default. Bindings are exposed by a service endpoint that includes the binding plus a Uniform Resource Identifier (URI) to which the client will send messages.

The following is an example of a **wsHttpBinding** that has been configured to use transport security:

```
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security mode="Transport">
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

The following configuration snippet shows an endpoint that exposes this binding:

```
<endpoint address="" binding="wsHttpBinding"
  bindingConfiguration="wsHttpEndpointBinding"
  name="wsHttpEndpoint" contract="IService">
```

Additional Resources

For more information on bindings, see "Windows Communication Foundation Bindings" at
http://msdn2.microsoft.com/en-us/library/ms733027.aspx

What bindings are available?

The following table summarizes common bindings:

Binding	Description

customBinding	Allows you to create a custom binding with full control over the message stack.
basicHttpBinding	It represents a bindings that configures and expose endpoints that are able to communicate with ASMX-based Web services and clients and other services that conform to the WS-I Basic Profile 1.1. By defaults it has security disabled.
wsHttpBinding	Defines a secure, reliable, interoperable binding suitable for non-duplex service contracts. The binding implements the following specifications: WS-Reliable Messaging for reliability, and WS-Security for message security and authentication. The transport is HTTP, and message encoding is Text/XML encoding. By default it provides message security with windows authentication.
ws2007HttpBinding	Defines a secure, reliable, interoperable binding suitable for non-duplex service contracts. The binding implements the following specifications: WS-Reliable Messaging for reliability, and WS-Security for message security and authentication. The transport is HTTP, and message encoding is Text/XML encoding. The **ws2007HttpBinding** provides binding similar to **wsHttpBinding** but uses the standard for OASIS (Organization for the Advancement of Structured Information Standards). By default it provides message security with windows authentication.
netTcpBinding	Specifies a secure, reliable, optimized binding suitable for cross-machine communication. By default, it generates a runtime communication stack with transport security and windows authentication as default security settings. It uses TCP protocol for message delivery, and binary message encoding.
netNamedPipeBinding	Defines a binding that is secure, reliable, optimized for on-machine cross process communication. By default, it generates a runtime communication stack with WS-ReliableMessaging for reliability, transport security for transfer security, named pipes for message delivery, and binary message encoding. It is not secured by default.
netMsmqBinding	Defines a queued binding suitable for cross-machine communication.
wsFederationHttpBinding	Defines a binding that supports federated security. It helps implementing Federation which is the ability to flow and share identities across multiple enterprises or trust domains for authentication and authorization. WCF implements federation over message and mixed mode security but not over transport security. Services configured with this binding

	must use the HTTP protocol as transport
ws2007FederationHttpBinding	**ws2007FederationHttpBinding.** Defines a binding that derives from **wsFederationHttpBinding** and supports federated security. It helps implementing Federation which is the ability to flow and share identities across multiple enterprises or trust domains for authentication and authorization. WCF implements federation over message and mixed mode security but not over transport security. Services configured with this binding must use the HTTP protocol as transport. The **ws2007FederationHttpBinding** provides binding similar to **ws2007FederationHttpBinding** but uses the standard for OASIS (Organization for the Advancement of Structured Information Standards)
wsDualHttpBinding	Defines a secure, reliable and interoperable binding that is suitable for duplex service contracts or communication through SOAP intermediaries.
customBinding	Allows you to create a custom binding with full control over the message stack.

Additional Resources

For more information on bindings, see "Windows Communication Foundation Bindings" at http://msdn2.microsoft.com/en-us/library/ms733027.aspx

Which bindings are best suited for the Internet?

- If you are exposing your WCF service over the Internet to clients that expect a legacy ASMX Web service, use **basicHttpBinding**. Keep in mind that this binding does not have any security enabled by default, so all messages will be sent in plain text format.
- If you are exposing your WCF service over the Internet to Windows Forms clients, use **wsHttpBinding**.
- If you are exposing your WCF service over an intranet to an ASP.NET application, which in turn is exposed to the clients over the Internet, use **netTcpBinding**.

Additional Resources

For more information on bindings, see "Windows Communication Foundation Bindings" at http://msdn2.microsoft.com/en-us/library/ms733027.aspx

Which bindings are best suited for the Intranet?

- If you are exposing your WCF service over your intranet to clients that expect a legacy ASMX Web service, use **basicHttpBinding**. Keep in mind that this binding does not have any security enabled by default, so all messages will be sent in plain text format.
- If you are exposing your WCF service over your intranet to Windows Forms or ASP.NET clients, use **netTcpBinding**.

Additional Resources

For more information on bindings, see "Windows Communication Foundation Bindings" at http://msdn2.microsoft.com/en-us/library/ms733027.aspx

How do I choose an appropriate binding?

If your service needs to support legacy clients that expect an ASMX Web service, consider using **basicHttpBinding**. Because **basicHttpBinding** does not implement any security by default, if you require message or transport security, you should configure it explicitly on this binding. Use **basicHttpBinding** to expose endpoints that are able to communicate with ASMX-based Web services and clients and other services that conform to the WS-I Basic Profile 1.1. When configuring transport security, **basicHttpBinding** defaults to no credentials just like a classic ASMX web service. **BasicHttpBinding** allows you to host your service in IIS 5.0 or IIS 6.0.

If your service will be called by WCF clients over the Internet, consider using **wsHttpBinding**. **wsHttpBinding** is a good choice for Internet scenarios in which you do not have to support legacy clients that expect an ASMX Web service. If you do need to support legacy clients, consider using **basicHttpBinding** instead. **WsHttpBinding** allows you to host your service in IIs 5.0 or IIS 6.0.

If you need to support clients within your intranet, consider using **netTcpBinding**. **netTcpBinding** is a good choice for the intranet scenario if transport performance is important to you and it is acceptable to host the service in a Windows service instead of in IIS. The **netTcpBinding** uses the TCP protocol and provides full support for SOAP security, transactions, and reliability. Use this binding when you want to provide a secure and reliable binding environment for .NET-to-.NET cross-machine communication. **NetTcpBinding** does not allow you to host your service in IIS 5.0 or IIS 6.0; instead, host in a Windows service or in IIS 7.0.

If you need to support WCF clients on the same machine as your service, consider using **netNamedPipeBinding**. This binding provides a secure and reliable binding environment for cross-process, same-machine communication. Use this binding when you want to make use of the NamedPipe protocol and provide full support for SOAP security, transactions, and reliability. **NetNamedPipeBinding** does not allow you to host your service in IIS 5.0 or IIS 6.0; instead, host in a Windows service or in IIS 7.0.

If you need to support disconnected queuing, use **netMsmqBinding**. Queuing is provided by using Microsoft Message Queuing (MSMQ) as a transport, which enables support for disconnected operations, failure isolation, and load leveling. You can use **netMsmqBinding** when the client and the service do not have to be online at the same time. You can also manage any number of incoming messages by using load leveling. MSMQ supports failure isolation, where messages can fail without affecting the processing of other messages. **NetMsmqBinding** does not allow you to host your service in IIS 5.0 or IIS 6.0; instead host in a Windows service or in IIS 7.0.

If you need to support a duplex service, use **wsDualHttpBinding**. A *duplex service* is a service that uses duplex message patterns, thus providing the ability for a service to communicate back to

the client via a callback. You can also use this binding to support communication via SOAP intermediaries. **WsDualHttpBinding** does not allow you to host your service in IIS 5.0 or IIS 6.0; instead, host in a Windows service or in IIS 7.0.

Additional Resources

For more information on bindings, see "Windows Communication Foundation Bindings" at http://msdn2.microsoft.com/en-us/library/ms733027.aspx

Configuration Management

- **How do I encrypt sensitive data in the WCF configuration file?**
- **How do I run a WCF service with a particular identity?**
- **How do I create a service account for running my WCF service?**
- **When should I use a configuration file versus the WCF object model?**
- **What is a metadata exchange (MEX) binding?**
- **How do I keep clients from referencing my service?**

How do I encrypt sensitive data in the WCF configuration file?

Use the aspnet_regiis.exe tool with the **-pe** (provider encryption) option to encrypt sections of the configuration files.

For example, to encrypt the **connectionStrings** section**,** using the Windows Data Protection API (DPAPI) provider with the machine key store (the default configuration), run the following command from a command prompt:

```
aspnet_regiis -pe "connectionStrings" -app "/MachineDPAPI"
-prov "DataProtectionConfigurationProvider"
```

The configuration options for aspnet_regiis are:

- **-pe** specifies the configuration section to encrypt.
- **-app** specifies your Web application's virtual path. If your application is nested, you need to specify the nested path from the root directory; for example, "/test/aspnet/MachineDPAPI"
- **-prov** specifies the provider name.

The .NET Framework supports the **RSAProtectedConfigurationProvider** and **DPAPIProtectedConfigurationProvider** protected configuration providers:

- **RSAProtectedConfigurationProvider –** This is the default provider and uses RSA public key encryption to encrypt and decrypt data. Use this provider to encrypt configuration files for use on multiple WCF services in a Web farm.
- **DPAPIProtectedConfigurationProvider –** This provider uses DPAPI to encrypt and decrypt data. Use this provider to encrypt configuration files for use on a single Microsoft Windows Server®.

You do not need to take any special steps for decryption beause the .NET run time takes care of this for you.

Additional Resources

- For more information on the aspnet_regiis tool, see "ASP.NET IIS Registration Tool (Aspnet_regiis.exe)" at http://msdn.microsoft.com/en-us/library/k6h9cz8h(VS.80).aspx
- For more information on encrypting configuration sections, see "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI" at http://msdn2.microsoft.com/en-us/library/ms998280.aspx and "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA" at http://msdn2.microsoft.com/en-us/library/ms998283.aspx

How do I run a WCF Service with a particular identity?

If your service is hosted in IIS 6.0, use IIS Manager to create an application pool running as a specific identity and then use IIS Manager to assign your WCF service to that application pool.

If your service is hosted in a Windows service, configure the Windows service to run using a particular identity. The WCF service will run under the security context of the Windows service.

Running a WCF service with a specific identity helps to isolate your service, allows you to restrict service resources to your application's account, and allows you to use Windows auditing to track the activity of the application separately from other applications or services.

How do I create a service account for running my WCF Service?

Perform the following steps to create a service account to run your WCF service:

1. Create a Windows account
2. Run the following aspnet_regiis.exe command to assign the relevant ASP.NET permissions to the account:
   ```
   aspnet_regiis.exe -ga machineName\userName
   ```

   Note: This step is required when your application needs to run in ASP.NET compatibility mode; otherwise, you can skip this step.

3. Use the Local Security Policy tool to grant the Windows account the **Deny logon locally** user right. This reduces the privileges of the account and prevents anyone from logging onto Windows locally with this account.

Additional Resources

For more information on the aspnet_regiis tool, see "ASP.NET IIS Registration Tool (Aspnet_regiis.exe)" at http://msdn.microsoft.com/en-us/library/k6h9cz8h(VS.80).aspx

When should I use a configuration file versus the WCF object model?

In general, you should configure your WCF service and clients using the web.config or app.config files. Using configuration files allows you to change transport, security, and other settings without having to rewrite and recompile your code. Object model code will override configuration settings, so you can use a combination of both if necessary.

What is a metadata exchange (MEX) binding?

A metadata exchange endpoint publishes the metadata for the service. The service metadata is consumed by clients to create a proxy and then call the service. The endpoint supports a standard for exchanging the metadata; WCF provides the implementation in the form of **IMetadataExchange**.

As with any other endpoint, the metadata endpoint consists of address, contract, and binding. The metadata bindings are the means for the clients to interact with the service and get the metadata for generating the proxies.

Several out-of-box bindings such as **MexHttpBinding**, **MexHttpsBinding**, and **MexTcpbinding** are available tosupport specific protocols.

Additional Resources

For more information on publishing metadata endpoints,see “Publishing Metadata Endpoints” at http://msdn.microsoft.com/en-us/library/ms788760.aspx

How do I keep clients from referencing my service?

To stop your service from publishing metadata, remove all the Mex endpoints from your service configuration and configure **HttpGetEnabled** and **HttpsGetEnabled** to False in the ServiceBehavior section, as shown below:

```
<serviceMetadata httpGetEnabled="False" httpsGetEnabled="False"/>
```

Additional Resources

For more information on publishing metadata endpoints, see “Publishing Metadata Endpoints” at http://msdn.microsoft.com/en-us/library/ms788760.aspx

Deployment Considerations

- **What are the additional considerations for using WCF in a Web farm?**
- **How do I configure Active Directory groups and accounts for role-based authorization checks?**
- **How do I create an X.509 certificate?**
- **When should I use a service principal name (SPN)?**
- **How do I configure a least-privileged account for my service?**

What are the additional considerations for using WCF in a Web farm?

When hosting your WCF service in a Web farm, use RSA instead of DPAPI to encrypt your configuration files. RSA is a better choice because it is easier to export RSA key containers and transport them between servers.

If your WCF services are hosted in an IIS Web farm in which multiple servers are addressed using the same endpoint URL, you will need to configure the default identity in IIS to use an explicit hostname.

Additional Resources

For more information on WCF hosting best practices, see "Internet Information Services Hosting Best Practices" at http://msdn2.microsoft.com/en-us/library/aa751802.aspx

How do I configure Active Directory groups and accounts for role-based authorization checks?

You do not need to do anything special to configure Active Directory groups and accounts for WCF role-based authorization checks. You can use them directly for either declarative or programmatic authorization.

The following is an example of a declarative authorization check using an Active Directory group in WCF:

```
[PrincipalPermission(SecurityAction.Demand, Role = "accounting")]
public double Add(double a, double b)
{
    return a + b;
}
```

The following is an example of a programmatic authorization check using an Active Directory group in WCF:

```
WindowsPrincipal myPrincipal = new
WindowsPrincipal(ServiceSecurityContext.Current.WindowsIdentity);
if(myPrincipal.IsInRole(@"domain\Accounting"))
{
//authorized
}
else
{
//not authorized
}
```

Additional Resources

- For more information on authorization, see "Authorization" at http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- For **Roles.IsUserInRole** method information, see "Roles.IsUserInRole Method (String)" at http://msdn.microsoft.com/en-us/library/4z6b5d42.aspx

How do I create an X.509 certificate?

In a production environment, use an X.509 certificate issued by a Certificate Authority (CA) such as VeriSign. In a development environment, use the MakeCert utility to create a temporary X.509 certificate.

For more information, see "How To - Create and Install Temporary Certificates in WCF for Message Security During Development" and "How To - Create and Install Temporary Certificates in WCF for Transport Security During Development."

Note: Do not use temporary development certificates in a production environment as this will open your communication channel to malicious spoofing, sniffing, and tampering.

Additional Resources

- For more information on working with certificates in WCF, see "Working with Certificates" at http://msdn.microsoft.com/en-us/library/ms731899.aspx
- For more information on mapping certificates to Windows accounts, see "Map certificates to user accounts" at http://technet2.microsoft.com/WindowsServer/f/?en/library/0539dcf5-82c5-48e6-be8a-57bca16c7e171033.mspx
- For more information on mapping certificates to Active Directory, see "Mapping Client Certificates with Directory Service Mapping" at http://technet2.microsoft.com/windowsserver/en/library/7cce4299-28f2-45fa-8730-4e0cbe3be8561033.mspx?mfr=true
- For more information on certificate mapping strategies, see "Mapping Strategies" at http://technet2.microsoft.com/windowsserver/en/library/aa61c564-1599-4414-a12d-2f64786f6ec31033.mspx?mfr=true

When should I use a service principal name (SPN)?

You will need to use an SPN under the following conditions:

- If you are using a custom domain account in the identity pool for your WCF application, create an SPN for Kerberos to authenticate the client.
- If you are using a custom service account and need to use trusted for delegation, create an SPN.
- If you are hosting your service in a Windows service, using a custom domain identity, and ASP.NET needs to use constrained trusted for delegation when calling the service, create an SPN.

Additional Resources

For more information on SPN, see "Setspn Overview" at http://technet2.microsoft.com/windowsserver/en/library/b3a029a1-7ff0-4f6f-87d2-f2e70294a5761033.mspx?mfr=true

How do I configure a least-privileged account for my service?

Perform the following steps to create a least-privileged account for your service:

1. Create a Windows account.
2. Run the following aspnet_regiis.exe command to assign the relevant ASP.NET permissions to the account:
```
aspnet_regiis.exe -ga machineName\userName
```
3. If your application needs to run in ASP.NET compatibility mode, use the Local Security Policy tool to grant the Windows account the **Deny logon locally** user right. This reduces

the privileges of the account and prevents anyone logging onto Windows locally with this account. Otherwise, skip this step.

4. Use the least-privileged account to run your WCF service:
    - If your service is hosted in IIS 6.0, use IIS Manager to create an application pool running as an account identity. Use IIS Manager to assign your WCF service to that application pool.
    - If your service is hosted in Windows service, configure the Windows service to run using the account identity. The WCF service will run under the security context of the Windows service.

Additional Resources

- For more information on the aspnet_regiis tool, see "ASP.NET IIS Registration Tool (Aspnet_regiis.exe)" at http://msdn.microsoft.com/en-us/library/k6h9cz8h(VS.80).aspx

Exception Management

- **How do I implement a global exception handler?**
- **What is a fault contract?**
- **How do I define a fault contract?**
- **How do I avoid sending exception details to the client?**

How do I implement a global exception handler?

To implement a global exception handler, subscribe to the Faulted event of a service host object. When you receive the faulted event, you can determine the cause of a failure and perform the necessary actions to abort or restart the service.

The following code snippet shows how to subscribe to the Faulted event:

```
// hosting a WCF service
ServiceHost customerServiceHost;
customerServiceHost = new ServiceHost(…);
…
// Subscribe to the Faulted event of the customerServiceHost object
customerServiceHost.Faulted += new EventHandler(faultHandler);
…
// FaultHandler method - invoked when customerServiceHost enters the
Faulted state
void faultHandler(object sender, EventArgs e)
{     // log the reasons for the fault  ..
}
```

Additional Resources

For more information about exceptions, see "Exceptions Reference" at http://msdn2.microsoft.com/en-us/library/ms733763.aspx

What is a fault contract?

A fault contract is a message contract that details the set of exceptions that may be reported to the caller. You can specify the possible faults that can occur in you WCF service. This prevents exposing exception details beyond the defined set to your clients. Because a fault contract lists the types of errors that a WCF service can throw, it also allows your clients to distinguish between contracted faults and other possible errors.

A fault contract can be set on an operation in your service like this:

```
[ServiceContract]
interface ICalculator
{
   [OperationContract]
   [FaultContract(typeof(DivideByZeroException))]
   double Divide(double number1,double number2);
}
```

Additional Resources

For more information on exceptions, see “Exceptions Reference” at http://msdn2.microsoft.com/en-us/library/ms733763.aspx

How do I define a fault contract?

To define a fault contract, apply the **FaultContract** attribute directly on a contract operation, specifying the error detailing type as shown below:

```
[ServiceContract]
interface ICalculator
{

   [OperationContract]
   [FaultContract(typeof(DivideByZeroException))]
   double Divide(double number1,double number2);

}
```

In this example, the **FaultContract** attribute is limited to the **Divide** method. This means only that method can throw that fault and have it propagated to the client. Also, the service must throw exactly the same detailing type listed in the fault contract to propagate the exception, as shown below:

```
class MyService : ICalculator
{
   public double Divide(double number1,double number2)
   {

      throw new FaultException<DivideByZeroException>(new
DivideByZeroException());
   }
}
```

Additional Resources

For more information on exceptions, see "Exceptions Reference" at http://msdn2.microsoft.com/en-us/library/ms733763.aspx

How do I avoid sending exception details to the client?

To avoid sending potentially sensitive exception details to the client, use the **FaultContract** attribute in a service contract to specify the possible faults that can occur in your WCF service.

The following code snippet shows how to use the **FaultContract** attribute to return error information:

1. Define the type to pass the details of SOAP faults as exceptions from a service back to a client.

```
[DataContract]
public class DatabaseFault
{
    [DataMember]
    public string DbOperation;
    [DataMember]
    public string DbReason
    [DataMember]
    public string DbMessage;
}
```

2. Use the **FaultContract** attribute in the **ListCustomers** method to generate SOAP faults.

```
[ServiceContract]
public interface ICustomerService
{
    // Get the list of customers
    [FaultContract(typeof(DatabaseFault))]
    [OperationContract]
    List<string> ListCustomers();
    …
}
```

3. Create and populate the **DatabaseFault** object with the details of the exception in the Service implementation class and then throw a **FaultException** object with the **DatabaseFault** object details.

```
catch(Exception e)
{    DatabaseFault df = new DatabaseFault();
    df.DbOperation = "ExecuteReader";
    df.DbReason = "Exception in querying the Northwind database.";
    df.DbMessage = e.Message;
    throw new FaultException<DatabaseFault>(df);
}
```

Additional Resources

- For more information on exceptions, see "Exceptions Reference" at http://msdn2.microsoft.com/en-us/library/ms733763.aspx

Hosting

- **How do I configure a least-privileged account to host my service?**
- **When should I host my service in IIS?**
- **When should I host my service in a Windows service?**
- **When should I self-host my service?**

How do I configure a least-privileged account to host my service?

Perform the following steps to create a least-privileged account for your service:

1. Create a Windows account.
2. Run the following aspnet_regiis.exe command to assign the relevant ASP.NET permissions to the account:
   ```
   aspnet_regiis.exe -ga machineName\userName
   ```
3. If your application needs to run in ASP.NET compatibility mode, use the Local Security Policy tool to grant the Windows account the **Deny logon locally** user right. This reduces the privileges of the account and prevents anyone from logging onto Windows locally with this account. Otherwise, skip this step.
4. Use the least-privileged account to run your WCF service:
   - If your service is hosted in IIS 6.0, use IIS Manager to create an application pool running as an account identity. Use IIS Manager to assign your WCF service to that application pool.
   - If your service is hosted in a Windows service, configure the Windows service to run using the account identity. The WCF service will run under the security context of the Windows service.

Additional Resources

- For more information about hosting, see “Hosting” at http://msdn2.microsoft.com/en-us/library/ms729846.aspx
- more information on the aspnet_regiis tool, see “ASP.NET IIS Registration Tool (Aspnet_regiis.exe)“ at http://msdn.microsoft.com/en-us/library/k6h9cz8h(VS.80).aspx
- For more information on running IIS under a least-privileged service account, see “How To: Create a Service Account for an ASP.NET 2.0 Application” at http://msdn2.microsoft.com/en-us/library/ms998297.aspx

When should I host my service in IIS?

Use IIS to host your WCF service, unless you need to use a transport that IIS does not support. IIS provides a large number of features for efficient service management and scalability. By using IIS as your WCF service host, you can take full advantage of IIS features such as process recycling, idle shutdown, process health monitoring, and message-based activation.

IIS 5.0 and 6.0 only support bindings over HTTP, so if you need to use TCP, MSMQ, or named pipes, you should host in a Windows service instead.

IIS 7.0 supports all the commonly used transport protocols such as HTTP, TCP, MSMQ, and named pipes.

Additional Resources

- For more information, see "Hosting" at http://msdn2.microsoft.com/en-us/library/ms729846.aspx

When should I host my service in a Windows service?

You should use a Windows service when you have to support transports such as TCP, MSMQ, and named pipes and you don't yet have IIS 7.0. Windows services offer advantages over self-hosting in that they give the benefit of automatic startup; service lifetime is controlled by the operating system; it is easier to run under a least-privileged account; and the Windows service host will restart your service if it fails. Windows services can be managed in the Service Control Manager in the Microsoft Management Console (MMC).

The drawback of using a Windows service compared to IIS is that Windows services have limited features to support availability, manageability, and deployment, so you will have to write custom code to support these features. For instance, you will need to add an installer to your service so that it can be installed on the system.

Additional Resources

For more information, see "Hosting" at http://msdn2.microsoft.com/en-us/library/ms729846.aspx

When should I self-host my service?

Self-hosting should only be used for development and demonstration purposes because it is not suitable for a production scenario. Similar to using a Windows service, self-hosting can be used with any transport and therefore provides binding flexibility beyond the capabilities of IIS 5.0 or IIS 6.0. Self-hosting is also the easiest hosting mechanism to get up and running, furthering its usefulness in a development environment. The drawback of self-hosting compared to a Windows service is that service lifetime is controlled by the self-hosted application rather than by the operating system, which makes it harder to run under a least-privileged account, and there is no automatic recovery in the case of failure.

Additional Resources

For more information, see "Hosting" at http://msdn2.microsoft.com/en-us/library/ms729846.aspx

Impersonation/Delegation

- **What are my impersonation options?**
- **What is the difference between impersonation and delegation?**
- **How do I impersonate the original caller for an operation call?**
- **How do I temporarily impersonate the original caller in an operation call?**

- **How do I impersonate a specific (fixed) identity?**
- **What is constrained delegation?**
- **What is protocol transition?**
- **How do I flow the original caller from the ASP.NET client to a WCF service?**
- **What is the difference between declarative and programmatic impersonation?**
- **What is the trusted subsystem model?**
- **When should I flow the original caller to back-end code?**
- **How do I control access to a remote resource based on the original caller's identity?**

What are my impersonation options?

There are three options for impersonation:

1. **Impersonating the original caller declaratively on specific operations.** Use this option when you want to impersonate the original caller for the entire duration of a specific operation.
2. **Impersonating the original caller declaratively on the entire service.** Use this option when you want to impersonate the original caller for the entire duration of all operations in the service.
3. **Impersonating the original caller programmatically within an operation.** Use this option when you want to impersonate the original caller for a short duration in a service operation.

Additional Resources

For more information, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

What is the difference between impersonation and delegation?

Impersonation flows the original caller's identity to back-end resources on the same computer. Delegation flows the original caller's identity to back-end resources on computers other than the computer running the service.

For example, if a service is running within IIS without impersonation, the service will access resources using the ASPNET account in IIS 5.0, or the NetworkService account in IIS 6.0. With impersonation, if the client is connecting using the original caller's account, the service will access resources such as a SQL Server database on the same machine using the original caller's account instead of the system ASPNET account. Delegation is similar except that the SQL Server database could be on a different machine that is remote to the service.

Additional Resources

For more information, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

How do I impersonate the original caller for an operation call?

Because impersonation is a costly operation and is usually used for higher-privileged original callers, you should use impersonation only on operations that need it to reduce the potential attack surface.

You can impersonate declaratively by applying the **OperationBehaviorAttribute** attribute on any operation that requires client impersonation, as shown in the following code example:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
   return “test”;
}
```

Additional Resources

For more information, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

How do I temporarily impersonate the original caller in an operation call?

Because impersonation is a costly operation and is usually used for higher-privileged original callers, you should use impersonation only when it is needed to reduce the potential attack surface. Programmatic impersonation allows you to impersonate on specific lines of code rather than the entire operation.

You can use programmatic impersonation to temporarily impersonate the original caller in an operation call, as shown in the following example:

```
public string GetData(int value)
{
 using (ServiceSecurityContext.Current.WindowsIdentity.Impersonate())
 {
     // return the impersonated user (original users identity)
     return string.Format("Hi, {0}, you have entered: {1}",
          WindowsIdentity.GetCurrent().Name, value);
 }
}
```

In the above example, the **using** statement is employed to ensure that the impersonation is reverted after execution of the using block. It is important to revert impersonation because failure to do so can form the basis for denial of service (DoS) and elevation of privilege attacks.

Additional Resources

For more information, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

How do I impersonate a specific (fixed) identity?

Use the **WindowsIdentity** class to obtain a Windows token and logon session for a given domain account by supplying a user principal name (UPN). With this approach, you do not need the account's password.

```
using System.Security.Principal;
…
WindowsIdentity wi = new
WindowsIdentity(userName@fullyqualifieddomainName);
WindowsImpersonationContext ctx = null;

try
{
  ctx = wi.Impersonate();
  // Thread is now impersonating you can call the backend operations
here...

catch
{
  // Prevent exceptions propagating.
}
finally
{
  // Ensure impersonation is reverted
  ctx.Undo();
}
```

Note: The **WindowsIdentity** constructor relies on a Windows Server 2003 extension to the Kerberos protocol called Service for User to Self (S4U2Self). You can use this approach if your application runs on a Windows Server 2003 server in a Windows Server 2003 domain. The advantage of this approach is that you do not have to store credentials as you do for LogonUser.

Additional Resources

- For more information, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

What is constrained delegation?

Impersonation is a WCF service configuration in which the service will access resources on the same computer using a client’s user identity. Delegation is similar to impersonation except that the WCF service can access resources that are on the same machine or on other machines using the client’s user identity. Delegation flows the original caller’s identity to back-end resources on the computers other than the computer running the service.

The Microsoft Windows Server 2003 operating system provides a more secure form of delegation called constrained delegation. With constrained delegation, you can configure the Microsoft Active Directory service to restrict the services and servers that your WCF service

application can access with the impersonated identity. Constrained delegation in Windows Server 2003 requires Kerberos authentication.

Additional Resources

- For more information, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

What is protocol transition?

Protocol transition is a Windows Server 2003 feature that allows you to switch from an alternate, non-Windows authentication mode (such as forms-based or certificate authentication) to Kerberos authentication. This is useful when your application cannot use Kerberos authentication to authenticate its callers, and when your application needs to use constrained delegation to access downstream network resources.

Additional Resources

- For more information, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

How do I flow the original caller from the ASP.NET client to a WCF service?

The following steps show how to impersonate the original caller from the ASP.NET client to a WCF service:

1. Configure your WCF service to use Windows authentication.

```
…
<services>
  <service name="Service" behaviorConfiguration="ServiceBehavior">
    <endpoint address="" binding="wsHttpBinding" contract="IService">
      <identity>
        <dns value="localhost"/>
      </identity>
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
         contract="IMetadataExchange"/>
  </service>
</services>
…
```

2. Configure the SPN identity for the WCF service endpoint.

```
 <endpoint address="" binding="wsHttpBinding" contract="IService">
   <identity>
     <servicePrincipalName value="HOST/YourMachineName" />
     <dns value="" />
   </identity>
 </endpoint>
```

3. Implement impersonation in the WCF service.

```
using System.Security.Principal;
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
      public string GetData(int value)
      {
        return string.Format("Hi, {0}, you have entered: {1}",
```

```
                                    WindowsIdentity.GetCurrent().Name, value);
        }
```

4. Create a Web application test client and add the WCF service reference.
5. Impersonate the original caller when calling the WCF service.

```
Using System.Security.Principal;
…
protected void Button1_Click(object sender, EventArgs e)
{
    // Obtain the authenticated user's Identity and impersonate the
original caller
    using
(((WindowsIdentity)HttpContext.Current.User.Identity).Impersonate())
    {
        WCFTestService.ServiceClient myService = new
WCFTestService.ServiceClient();
        Response.Write(myService.GetData(123) + "<br/>");
        myService.Close();
    }
}
…
```

6. Configure the Web application for constrained delegation.
    a. If your ASP.NET application runs using the Network Service machine account, you must enable constrained delegation for your Web server computer.
    b. If your ASP.NET application runs under a custom domain account, you must enable protocol transition and constrained delegation for the custom domain account.
7. Test the client and WCF service.

Additional Resources

- For more information on constrained delegation, see “How To: Use Protocol Transition and Constrained Delegation in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998355.aspx

What is the difference between declarative and programmatic impersonation?

Impersonation is used to restrict or authorize the original caller’s access to a WCF service’s local resources, such as files. Use declarative impersonation to define impersonation at the operation or service level.

Impersonate declaratively by applying the **OperationBehaviorAttribute** attribute on any operation that requires client impersonation, as shown in the following code example:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
   return “test”;
}
```

Use programmatic impersonation to define finer-grained impersonation based on business logic. Programmatic impersonation is specified in code and applied at run time.

Programmatic impersonation can be performed as shown in the following example:

```
public string GetData(int value)
{
 using (ServiceSecurityContext.Current.WindowsIdentity.Impersonate())
 {
     // return the impersonated user (original users identity)
     return string.Format("Hi, {0}, you have entered: {1}",
          WindowsIdentity.GetCurrent().Name, value);
 }
}
```

Additional Resources

- For more information, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

What is the trusted subsystem model?

A trusted subsystem describes an architecture in which an upstream tier is trusted to authenticate and authorize the original caller for downstream components. For instance, a database server trusts the web application to authenticate users and then all calls from the web application to the database server are made with the web application’s identity instead of the original caller’s identity. In this model, the web application identity is trusted to make calls on behalf of the original caller.

The advantages of the trusted subsystem model include support for efficient connection pooling, no direct data access because only the service account is granted access to the back-end resources, and minimal back-end access control list (ACL) management.

Additional Resources

- For more information see “Trusted Subsystem” at http://msdn.microsoft.com/en-us/library/ms730288.aspx

When should I flow the original caller to back-end code?

Flow the original caller to back-end code when you need to authorize access to resources based on the original caller’s identity, or when the back-end code needs to perform roles-based authorization.

Additional Resources

- For more information, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

How do I control access to a remote resource based on the original caller's identity?

Use delegation to flow the impersonated original user's security context (Windows identity) to the remote back-end service. On the remote back-end service, the original user's windows identity can be used to authenticate or impersonate the original caller in order to restrict or authorize the original caller's access to local resources.

When using delegation on Windows Server 2003 or later, use constrained delegation. This allows administrators to specify exactly which services on a downstream server or a domain account can be accessed when using an impersonated user's security context.

Additional Resources

- For more information, see "Delegation and Impersonation with WCF" at http://msdn2.microsoft.com/en-us/library/ms730088.aspx

Input/Data Validation

- **How do I implement input and data validation in WCF?**
- **What is schema validation?**
- **What is parameter validation?**
- **Should I validate before or after message serialization?**
- **How do I protect my service from denial of service (DoS) attacks?**
- **How do I protect my service from malicious input attacks?**
- **How do I protect my service from malformed messages?**

How do I implement input and data validation in WCF?

If you need to validate operations that do not accept message or data contracts, use parameter inspectors. Parameter inspectors provide a convenient mechanism to process service method invocations when they are in a parameterized form. Parameter inspectors allow pre- and post-processing of messages through the use of custom validators. Unlike using a schema for validation, a custom validator requires you to write your own custom validation code.

If your service has operations that accept message or data contracts, use schemas to validate your messages. Message validation provides a way to validate messages when operations consume message contracts or data contracts that are not possible to validate using parameter validation. Message validation allows the creation of validation logic inside the schemas, thereby providing more flexibility and reducing development time. Schema validation occurs before serializing or encrypting the message. In order to reduce development time, you can start by using the service.xsd schema file that is generated when you add a service reference to your client.

What is schema validation?

Schema validation allows you to validate incoming messages against a set of configurable Extensible Markup Language (XML) Schema documents.
Schema validation is implemented in WCF using message inspectors. Client message inspectors implement the **IClientMessageInspector** interface, while service message inspectors implement the **IDispatchMessageInspector** interface.

The following steps show you how to perform message validation using schemas:

1. Use the schema.xsd schema file that is created by svcutil.exe when you add a service reference or create a schema that represents the operations of your service and the types consumed by those operations.
2. Create a .NET class that implements a custom client message inspector and custom dispatcher message inspector to validate the messages sent and received by the service.
3. Implement a custom endpoint behavior to enable message validation on both the client and the service.
4. Implement a custom configuration element on the class that allows you to expose the extended custom endpoint behavior in the configuration file of the service or the client.

Additional Resources

- For more information, see “Message Inspectors” at http://msdn.microsoft.com/en-us/library/aa717047.aspx

What is parameter validation?

Parameter validation allows you to inspect and validate incoming or outgoing message parameters.

You can inspect or modify the incoming or outgoing messages for a single operation on a WCF client object or WCF service by implementing the **System.ServiceModel.Dispatcher.IParameterInspector** interface and inserting it into the client or service run time.

```
public class Validation
{
       public class ValidationParameterInspector : IParameterInspector
        {
            public void AfterCall(string operationName, object[] outputs,
object returnValue, object correlationState)
            { …   }


            public object BeforeCall(string operationName, object[] inputs)
            { …    }
      …
}
```

Additional Resources

For more information, see "How to: Inspect or Modify Parameters" at http://msdn2.microsoft.com/en-us/library/ms733747.aspx

Should I validate before or after message serialization?

When performing schema validation, you will validate before deserialization because you are validating on the message itself. When performing parameter validation, you will validate after deserialization because you are validating data parameters within the message.

Additional Resources

- For more information, see "How to: Inspect or Modify Parameters" at http://msdn2.microsoft.com/en-us/library/ms733747.aspx

How do I protect my service from denial of service (DoS) attacks?

Configure your service with the **ServiceThrottlingBehavior** class to limit concurrent client calls, instances, and sessions.

The following example configures the service behavior for **serviceThrottling**. The three service throttling behavior attributes are described below.

```
<behaviors>
  <serviceBehaviors>
    <behavior name="Throttled">
      <serviceThrottling
        maxConcurrentCalls="1"
        maxConcurrentSessions="1"
        maxConcurrentInstances="1"
      />
      <serviceMetadata
        httpGetEnabled="true"
        httpGetUrl=""
      />
    </behavior>
  </serviceBehaviors>
</behaviors>
```

- **MaxConcurrentCalls** – Gets or sets a value that specifies the maximum number of messages actively processing across a **ServiceHost** object.
- **MaxConcurrentInstances** – Gets or sets a value that specifies the maximum number of **InstanceContext** objects in the service that can execute at one time.
- **MaxConcurrentSessions** – Gets or sets a value that specifies the maximum number of sessions a **ServiceHost** object can accept at one time.

How do I protect my service from malicious input attacks?

Check for known good data and constrain input by validating it for type, length, format, and range. Do not trust any input. An attacker passing malicious input can attempt SQL injection, cross-site scripting, and other injection attacks that aim to exploit your application's

vulnerabilities. If your service has operations that accept message or data contracts, use schemas to validate your messages. If you need to validate operations that do not accept message or data contracts, use parameter inspectors.

How do I protect my service from malformed messages?

WCF catches malformed SOAP messages and you can log this event with WCF auditing. If the envelope is valid but the message fields are malformed in the sense of length, range, format, or type, you can use schema validation to perform message validation.

To log malformed messages using WCF auditing, set the **logMalformedMessages** attribute to true in the <messagelogging> element. Malformed message logging allows you to see any messages that were rejected by the WCF stack due to validation errors. This can help you determine if someone is attacking your service with poorly formed SOAP messages.

The following steps briefly show you how to perform message validation using schemas:

1. Use the schema.xsd schema file that is created by svcutil.exe when you add a service reference or create a schema that represents the operations of your service and the types consumed by those operations.
2. Create a .NET class that implements a custom client message inspector and custom dispatcher message inspector to validate the messages sent and received by the service.
3. Implement a custom endpoint behavior to enable message validation on both the client and the service.
4. Implement a custom configuration element on the class that allows you to expose the extended custom endpoint behavior in the configuration file of the service or the client.

Message Protection

- **When should I use message security?**
- **When should I use transport security?**
- **How do I protect my message when there are intermediaries routing my message?**
- **How do I protect my message when there are multiple protocols used during message transit?**

When should I use message security?

Message security encrypts each individual message to protect sensitive data. Transport security secures the end-to-end network connection to protect the network traffic.

Use the following criteria to decide whether to use message security:

- **Intermediaries** – Message security supports scenarios with intermediaries or protocol transition.
- **Encryption flexibility** – Message security allows you to encrypt part of a message while leaving other parts in cleartext format.
- **Binding limitations** – Message security does not work with **netNamedPipeBinding**.

- **Secure conversations** – Secure conversations only works with message security.
- **Authentication limitations** – Message security does not work with Basic or Digest authentication.

Additional Resources

- For more information on message protection, see "Message Security in WCF" at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For more information on choosing a transport, see "Choosing a Transport" at http://msdn2.microsoft.com/en-us/library/ms733769.aspx

When should I use transport security?

Message security encrypts each individual message to protect sensitive data. *Transport security* secures the end-to-end network connection to protect the network traffic.

Use the following security criteria to decide whether to use transport security:

- **Point-to-point** – Transport security supports point-to-point communication and does not support intermediary scenarios or protocol transition.
- **Streaming** – Transport security can support streaming data scenarios.
- **Binding limitations** – Transport security does not work with **wsDualHttpBinding**.
- **Authentication limitations** – Transport security does not work with negotiation, username, or Kerberos direct authentication.
- **Performance** – Transport security may provide better performance than message security.

Additional Resources

- For more information on choosing a transport, see "Choosing a Transport" at http://msdn2.microsoft.com/en-us/library/ms733769.aspx
- For more information on message security, see "Message Security in WCF" at http://msdn.microsoft.com/en-us/library/ms733137.aspx

How do I protect my message when there are intermediaries routing my message?

Use message security to protect your message when there are intermediaries routing your message. Message security protects message contents even if the message must travel between many points before reaching the intended recipient.

Additional Resources

- For more information on choosing a transport, see "Choosing a Transport" at http://msdn2.microsoft.com/en-us/library/ms733769.aspx

How do I protect my message when there are multiple protocols used during message transit?

Use message security to protect your message when there are multiple protocols used during message transit. Because each individual message is encrypted, message protection is transport-agnostic and can be used in protocol transition scenarios.

Additional Resources

- For more information on choosing a transport, see “Choosing a Transport” at http://msdn2.microsoft.com/en-us/library/ms733769.aspx

Proxy Considerations

- **When should I use a channel factory?**
- **When do I need to expose a metadata exchange endpoint for my service?**
- **How do I avoid proxy spoofing?**

When should I use a channel factory?

Use a channel factory when you control both ends of the wire and would rather code directly against the same common language runtime (CLR) interface instead of manually keeping the Web Services Description Language (WSDL) interface in sync. Instead of using WSDL as the shared contract, you use a shared "interface assembly."

When do I need to expose a metadata exchange endpoint for my service?

Expose the metadata exchange (MEX) endpoint to share the service metadata so that client programs can use the metadata to generate a proxy file to include in their code, in order to call service objects.

Additional Resources

- For more information on publishing metadata endpoints, see “Publishing Metadata” at http://msdn2.microsoft.com/en-us/library/aa751951.aspx
- For more information on metadata security considerations, see “Security Considerations with Metadata” at http://msdn.microsoft.com/en-us/library/ms734741.aspx

How do I avoid proxy spoofing?

Publish your service metadata over HTTPS to protect clients from being spoofed when adding a service reference. If you expose your service metadata over HTTP, clients cannot be certain that they have added a reference to the right service – the service may have been spoofed through DNS poisoning or a man-in-the-middle attack. To publish your service metadata over HTTPS, use **mexHttpsBinding** and configure a server certificate for the service.

If you are running your service in a scenario in which mutual authentication has been turned off, be aware that your service might be spoofed by a malicious attacker. Without mutual

authentication, calls to your service might be diverted to a malicious service through DNS poisoning or a man-in-the-middle attack.

The follow scenarios will result in mutual authentication being turned off:

- If you turn off message and transport security on your binding
- If you use **basicHttpBinding**, which has message and transport security turned off by default
- If you use NTLM authentication

Additional Resources

For more information on publishing metadata endpoints, see “Publishing Metadata Endpoints” at http://msdn.microsoft.com/en-us/library/ms788760.aspx

Sensitive Data

- **How do I protect sensitive data in configuration files?**
- **How do I protect sensitive data in memory?**
- **How do I protect my metadata?**
- **How do I protect sensitive data from being read on the wire?**
- **How do I protect sensitive data from being tampered with on the wire?**

How do I protect sensitive data in configuration files?

Use the aspnet_regiis.exe tool with the **-pe** (provider encryption) option to encrypt sections of the configuration files.
For example, to encrypt the **connectionStrings** section**,** using the Windows Data Protection API (DPAPI) provider with the machine key store (the default configuration), run the following command from a command prompt:

```
aspnet_regiis -pe "connectionStrings" -app "/MachineDPAPI"
-prov "DataProtectionConfigurationProvider"
```

The aspnet_regiis configuration options are:

- **-pe** specifies the configuration section to encrypt.
- **-app** specifies your Web application's virtual path. If your application is nested, you need to specify the nested path from the root directory; for example, "/test/aspnet/MachineDPAPI"
- **-prov** specifies the provider name.

The .NET Framework supports the **RSAProtectedConfigurationProvider** and **DPAPIProtectedConfigurationProvider** protected configuration providers:

- **RSAProtectedConfigurationProvider.** This is the default provider and uses the RSA public key encryption to encrypt and decrypt data. Use this provider to encrypt configuration files for use on multiple WCF services in a Web farm.
- **DPAPIProtectedConfigurationProvider.** This provider uses DPAPI to encrypt and decrypt data. Use this provider to encrypt configuration files for use on a single Windows Server.

You do not need to take any special steps for decryption because the .NET run time takes care of this for you.

Additional Resources

- For more information on encrypting a configuration section using DPAPI, see "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI" at http://msdn2.microsoft.com/en-us/library/ms998280.aspx
- For more information on encrypting a configuration section using RSA, see "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA" at http://msdn2.microsoft.com/en-us/library/ms998283.aspx

How do I protect sensitive data in memory?

To minimize exposure of secret data in memory, consider the following measures:

- Avoid creating multiple copies of the secret data because this increases your attack surface. Pass references to secret data instead of making copies of the data. Also understand that if you store secret data in immutable objects such as **System.String**, a new copy is created after each object manipulation.
- Keep the secret data encrypted for as long as possible. Decrypt the data at the last possible moment before you need to use the sensitive information it contains.
- Clean the clear text version of the secret data as soon as you are done using it.

You can use the **SecureString** method to implement the above measures. The value of a **SecureString** object is automatically encrypted, can be modified until your application marks it as read-only, and can be deleted from computer memory by either your application or the .NET Framework garbage collector.

The following C# code creates an instance of the **SecureString** class and stores a data value in it.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace TestSecureString
{
  class Program
  {
      static void Main(string[] args)
      {
      System.Security.SecureString secstr = new
      System.Security.SecureString();
      secstr.AppendChar('W');
      secstr.AppendChar('C');
      secstr.AppendChar('F');
      secstr.MakeReadOnly();
      Console.WriteLine(secstr);
      }
  }
}
```

An exception is thrown if you attempt to alter the data because the code locks the string value with the **MakeReadOnly** method after the final character has been added. Therefore this string value may not be altered.

How do I protect my metadata?

You can protect the metadata of a service by creating a secure HTTPS GET metadata endpoint in its configuration. Set the **httpsGetEnabled** attribute of the <serviceMetadata> element to true and the **httpsGetUrl** attribute of the <serviceMetadata> element to the address of your metadata interface.

The following configuration code shows how to secure the metadata:

```
 <system.serviceModel>
  <behaviors>
   <serviceBehaviors>
    <behavior name="myServiceBehavior">
     <serviceMetadata httpsGetEnabled="true"
httpsGetUrl="https://localhost:1234/calcMetadata" />
    </behavior>
   </serviceBehaviors>
  </behaviors>
 <services>
  <service behaviorConfiguration="myServiceBehavior"
        name="MySamples.Calculator">
   <endpoint address="http://localhost:8037/Samples/calculator"
   binding="wsHttpBinding" bindingConfiguration=""
   contract="MySamples.ICalculator" />
  </service>
 </services>
 </system.serviceModel>
```

Additional Resources

- For more information on publishing metadata endpoints, see “Publishing Metadata Endpoints” at http://msdn.microsoft.com/en-us/library/ms788760.aspx

How do I protect sensitive data from being read on the wire?

Use message or transport security to encrypt your message and keep sensitive information from being sniffed off the network. Message security encrypts each individual message to protect sensitive data. Transport security secures the end-to-end network connection to protect the network traffic.

How do I protect sensitive data from being tampered with on the wire?

Use message or transport security to check the integrity of your message and keep the messages from being tampered with on the network. Message security checks integrity of each individual message. Transport security protects the end-to-end network connection to protect against tampering.

X.509 Certificates

- **How do I create X.509 certificates?**
- **Do I need to create a certificate signed by the root CA certificate?**
- **How do I use X.509 certificate revocation?**

How do I create X.509 certificates?

In a production environment, use an X.509 certificate issued by a certificate authority (CA) such as VeriSign. In a development environment, use the MakeCert utility to create a temporary X.509 certificate.

Note: Do not use temporary development certificates in a production environment as this will open your communication channel to malicious spoofing, sniffing, and tampering.

Additional Resources

- For more information on working with certificates in WCF, see "Working with Certificates" at http://msdn.microsoft.com/en-us/library/ms731899.aspx
- For more information on creating certificates for message security, see "How To – Create and Install Temporary Certificates in WCF for Message Security During Development"
- For more information on creating certificates for message security, see "How To – Create and Install Temporary Certificates in WCF for Transport Security during Development"

Do I need to create a certificate signed by the root CA certificate?

In a production environment, you can use an X.509 certificate issued by a CA such as VeriSign, this will be created off of the certificate authority's root certificate. In a development environment, you can create a temporary root certificate and then generate another certificate signed by the root for use by the service.

Note: Do not use temporary development certificates in a production environment as this will open your communication channel to malicious spoofing, sniffing, and tampering.

Additional Resources

- For more information on working with certificates in WCF, see "Working with Certificates" at http://msdn.microsoft.com/en-us/library/ms731899.aspx
- For more information on creating certificates for message security, see "How To – Create and Install Temporary Certificates in WCF for Message Security During Development"
- For more information on creating certificates for message security, see "How To – Create and Install Temporary Certificates in WCF for Transport Security during Development"

How do I use X.509 certificate revocation?

By default, WCF services are configured to check certificate revocation when using certificate authentication. To revoke the certificate used by your service, contact the CA who issued the certificate and ask them to perform a certificate revocation and issue you a new certificate.

Additional Resources

- For more information on working with certificates in WCF, see "Working with Certificates" at http://msdn.microsoft.com/en-us/library/ms731899.aspx

How To – Audit and Log Security Events in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of auditing and logging security events. The article shows you how to configure a WCF service for Auditing, Message Logging, and Tracing, and how to use the SvcTraceViewer tool to view the log files.

Contents

Objectives

- Learn to configure Auditing.
- Learn to configure Message Logging and Tracing.
- Learn to log the service events in the Event Log.
- Learn to use the SvcTraceViewer tool.

Overview

WCF Auditing allows you to audit security events such as authentication and authorization failures.WCF service auditing can allow you to detect an attack that has occurred or is in progress. In addition, auditing can help you debug security-related problems.

WCF Message Logging allows you to log malformed Simple Object Access Protocol (SOAP) messages or to trace incoming messages. It allows you to specify different logging levels that you can use to diagnose and analyze your applications in case of any problems.

In this How To article, you will create a sample WCF service in Visual Studio 2008. You will then configure the service to enable Auditing, Logging, and Tracing through the use of the WCF Configuration Editor. Next, you will create a test client to verify the security events in the Event Log. Finally, you will use the SvcTraceViewer tool to view and examine the log and trace files.

Summary of Steps

- **Step 1 – Create a Sample WCF Service**
- **Step 2 – Enable Auditing for Your WCF Service**
- **Step 3 – Enable Logging and Tracing for Your WCF Service**
- **Step 4 – Create a Windows Forms Test Client Application**
- **Step 5 – Add a WCF Service Reference to the Client**
- **Step 6 – Test the Client and WCF Service**
- **Step 7 – Verify the Service Events in the Event Log**
- **Step 8 – Trace the Log File Using the SvcTraceViewer**

Step 1 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio.

1. In Visual Studio, on the menu, click **File** -> **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to Http and specify the virtual directory to be created in the **Path (**e.g., http://localhost/WCFTestService).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample WCF service.
4. Browse to your WCF service (i.e., http://localhost/WCFTestService/Service.svc). You should see details of your WCF service.

Step 2 – Enable Auditing for Your WCF Service

In this step, you configure the WCF service to use Security Auditing.

1. In the Configuration Editor, expand the **Advanced** node and then expand the Service Behaviors folder.
2. Select the default behavior "**ServiceBehavior**".
3. In the **Behavior: ServiceBehavior** section, click **Add**.
4. In the **Adding Behavior Element Extension Sections** dialog box, select **serviceSecurityAudit** and then click **Add**.
5. In the **Configuration** section, under **Service Behaviors**, select the **serviceSecurityAudit** option.
6. Set the **AudtiLogLocation** attribute to by **Application** by choosing this option from the drop-down list.
7. Set the **MessageAuthenticationAuditLevel** attribute to **SuccessOrFailure** by choosing this option from the drop-down list.

8. Set the **ServiceAuthorizationAuditLevel** attribute to **SuccessOrFailure** by choosing this option from the drop-down list.
9. In the Configuration Editor, on the **File** menu, click **Save**.
10. In Visual Studio, verify your configuration. The configuration should look as follows:

```
…
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
        <serviceSecurityAudit auditLogLocation="Application"
serviceAuthorizationAuditLevel="SuccessOrFailure"
         messageAuthenticationAuditLevel="SuccessOrFailure" />
    </behavior>
  </serviceBehaviors>
</behaviors>
…
```

Step 3 – Enable Logging and Tracing for Your WCF Service

In this step, you configure the WCF service to use Message Logging and Tracing.

Configure Logging

1. In the Configuration Editor, select the **Diagnostics** node.
2. In the right pane, click **Enable MessageLogging**.
   This will create **ServiceModelMessageLoggingListener** and **System.ServiceModel.MessageLogging** nodes under the Listeners and Sources folders, respectively.
3. In the left pane, select **MessageLogging** under the **Diagnostics** node.
4. Set the **LogMessagesAtServiceLevel** attribute to **True by** choosing this option from the drop-down list.
5. In the left pane, select **ServiceModelMessageLoggingListener** under the **Listeners** node.
   Note the default value of the **InitData** attribute, which is set to c:\inetpub\wwwroot\WCFService\web_messages.svclog, the location where the message will be logged.

Configure Tracing

1. In the Configuration Editor, **s**elect the **Diagnostics** node.
2. In the right pane, click **Enable Tracing**.
   This will create **ServiceModelTraceListener** and **System.ServiceModel nodes** under the Listeners and Sources folders, respectively.
3. In the left pane, select **ServiceModeTraceListener** under the **Listeners** node.
   Note the default value of **the InitData attribute**, which is set to

c:\inetpub\wwwroot\WCFService\web_tracelog.svclog, the location where the trace message will be logged.

4. In the Configuration Editor, on the **File** menu, click **Save**.
5. In Visual Studio, verify your configuration. The c**onfi**guration should look as follows:

```
…
<configuration>
<system.diagnostics>
  <sources>
   <source name="System.ServiceModel.MessageLogging"
switchValue="Warning, ActivityTracing">
    <listeners>
     <add type="System.Diagnostics.DefaultTraceListener"
name="Default">
      <filter type="" />
     </add>
     <add name="ServiceModelMessageLoggingListener">
      <filter type="" />
     </add>
    </listeners>
   </source>
   <source name="System.ServiceModel" switchValue="Warning,
ActivityTracing"
    propagateActivity="true">
    <listeners>
     <add type="System.Diagnostics.DefaultTraceListener"
name="Default">
      <filter type="" />
     </add>
     <add name="ServiceModelTraceListener">
      <filter type="" />
     </add>
    </listeners>
   </source>
  </sources>
  <sharedListeners>
   <add
initializeData="c:\inetpub\wwwroot\WCFService\web_messages.svclog
"
    type="System.Diagnostics.XmlWriterTraceListener, System,
Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"
    name="ServiceModelMessageLoggingListener"
traceOutputOptions="Timestamp">
    <filter type="" />
   </add>
   <add
initializeData="c:\inetpub\wwwroot\WCFService\web_tracelog.svclog
"
    type="System.Diagnostics.XmlWriterTraceListener, System,
Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"
    name="ServiceModelTraceListener"
traceOutputOptions="Timestamp">
    <filter type="" />
```

```
    </add>
  </sharedListeners>
</system.diagnostics>
</configuration>
…
…
<system.serviceModel>
  <diagnostics>
   <messageLogging logEntireMessage="false"
logMalformedMessages="true"
    logMessagesAtServiceLevel="true"
logMessagesAtTransportLevel="true" />
  </diagnostics>
…
```

Note: Although enabling Logging and Tracing is not a mandatory step for auditing security events, it will provide detailed information about every activity in an event.

Step 4 – Create a Windows Forms Test Client Application

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your Solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.

Step 5 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service.

1. Right-click your client project and then click **Add Web Reference**.
2. In the **Add Web Reference** dialog box, set the URL to your WCF service, (e.g., http://localhost/WCFTestService/Service.svc) and then click **Go**.
3. In the Web reference name field, change ServiceReference1 to WCFTestService.
4. Click **Add Reference**.
   A reference to WCFTestService should now appear beneath Web References in your client project.

Step 6 – Test the Client and WCF Service

In this step, you access the WCF service, pass the user credentials, and make sure that the username authentication works.

1. In your client project, drag a Button control onto your form.
2. Double-click the Button control to show the underlying code.
3. Create an instance of the proxy and call the **GetData** operation of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
```

```
{
      WCFTestService.ServiceClient myService = new
                  WCFTestService.ServiceClient();
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5. When you click the button on the form, the message "**You entered: 123**" should appear.

Step 7 – Verify the Service Events in the Event Log

In this step, you verify the WCF service events in the Application Event Log.

1. On your Service host machine, click **Start** and then click **Run**.
2. In the command line, type **eventvwr** and then click **OK** to open the Event Viewer window.
3. In the left pane, select the **Application** node, which shows the list of application events in the right pane.
4. In the list, search for **Source ServiceModel Audit 3.0.0.0**.
   You will find four event entries for your service, one with a ServiceAuthorization category and others with MessageAuthentication categories.
5. Open the event with the **ServiceAuthorization** category. You will see the following message if your service authorizes a client:

   ```
   Service authorization succeeded.
   Service: <<service URI>>
   Action: http://tempuri.org/<<your service method info>>
   Client Identity: <<domain\user-id>>;
   …
   ```

6. Similarly for the **MessageAuthentication** events, if your service authenticates a client, you will see the following message for Security Negotiation and Message Authentication events:

   ```
   Message authentication succeeded.
   Service: <<service URI>>
   Action: http://tempuri.org/<<your service method info>>
   Client Identity: <<domain\user-id>>;
   …
   ```

7. If you enabled Logging and Tracing (followed step 3) for your service, you will see another event with the **MessageLogging** category in the application log:

   ```
   Message logging succeeded.
   Service: <<service URI>>
   Action: http://tempuri.org/<<your service method info>>
   Client Identity: <<domain\user-id>>;
   ```

Step 8 – Trace the Log File Using the SvcTraceViewer

In this step, you verify the log file by using the trace viewer tool, SvcTraceViewe.exe, which enables you to view both the message log files and the trace files.

1. On your Service host machine, go to C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin.
2. Open the **SvcTraceViewe.exe** tool.
3. On the tool's menu, click **File**, click **Open**, and then browse to the location of the message log file.
   The right pane shows the various activities that takes place during a host's life cycle. You can step through the activity messages by pressing F10 and F11**.**

Additional Resources

- For more information on WCF security auditing, see "Auditing Security Events" **a**t http://msdn2.microsoft.com/en-us/library/ms731669.aspx
- For more information on auditing security events in WCF, see "How To: Audit Windows Communication Foundation Security Events" at http://msdn2.microsoft.com/en-us/library/ms734737.aspx
- For more information on auditing security concerns, see "Security Concerns for Message Logging" at http://msdn.microsoft.com/en-us/library/ms730318.aspx

How To: Create and Install Temporary Certificates in WCF for Message Security During Development

Applies to

- Microsoft Windows Communication Foundation (WCF) 3.5
- Microsoft® Visual Studio® 2008

Summary

This How To article walks you through to the process of creating and installing temporary certificates to be used during the development and testing of WCF services that implement message security. The article explains the process of creating, configuring, and installing these temporary certificates to work with WCF.

Contents

Objectives

- Learn how to create a root certificate for the temporary certificate used for WCF message security
- Learn how to created a root certification revocation list file for the root certificate used to do the revocation validation
- Learn how to create a temporary certificate for WCF message security
- Learn how to install the temporary certificate
- Learn how to install the root certificate for the temporary certificate
- Learn how to install the root certification revocation list for the root certificate

Overview

When developing a WCF service that uses X.509 certificates to provide message security, it is necessary to work with temporary certificates. This is because production certificates are expensive and may not be readily available. There are two options for specifying trust on a certificate:

- **Peer trust** – Validates the certificate directly.
- **Chain trust** – Validates the certificate against the issuer of a certificate known as a root authority.

This How To article allows doing the chain trust option because it is the most commonly used approach in Business-to-Business (B2B) scenarios, and it is the default validation for WCF, when using message security.

Additionally there is a certificate revocation list validation performed during message security. This validation checks that list of certificates that were revoked by the root certificate. Three modes or revocation exist

- **Online** The CRL list is retrieved and the check happens on line requiring connectivity to the URLs
- **Offline** The CRL list is retrieved and check happens online then is cached for subsequent validation
- **NoCheck** No validation is performed

This How To article allows doing the CRL check, without configuration changes when using message security

To use chain trust validation during development time, you create a self-signed root certificate authority (CA) and install it in the Trusted Root Certification Authority in the Local Machine. The certificate used by WCF is then created and signed by the root self-signed certificate and installed in the Personal store of Local Machine. To allow CRL check to succeed you create a self-signed root CRL file and install it in the Trusted Root Certification Authority store of the Local Machine.

You will use makecert.exe to create a private key file and a certificate to act as your root certificate authority (CA). You will then create a certificate revocation list file from the private key that will act as your revocation list file for the root certificate authority. Then you install the root certificate and CRL file. Finally you will create and install the temporary certificate from the root certificate, using the private key to sign and generate the key.

Summary of Steps

- Step 1 – Create a Certificate to Act as Your Root Certificate Authority
- Step 2 – Create a Certificate Revocation List File from the Root Certificate
- Step 3 – Install Your Root Certificate Authority on the Server and Client Machines
- Step 4 – Install the Certificate Revocation List File on the Server and Client Machines
- Step 5 – Create and Install Your Temporary Service Certificate
- Step 6 – Give the WCF Process Identity Access to the Temporary Certificate's Private Key

Step 1 – Create a Certificate to Act as Your Root Certificate Authority

In this step, you use the makecert tool to create a root CA that will be used to sign your certificate. This certificate will be self signed and will only have the public key that will be used to do the trust chain validation, when encrypting and signing messages. Self signed certificate will act as a root certificate itself, instead of pointing to a Root authority in a chain of trust.

1. Open a Visual Studio command prompt and browse to the location where you want to save the certificate files.
2. Run the following command to create the root CA

   makecert -n "CN=RootCATest" -r -sv RootCATest.pvk RootCATest.cer

   In this command:

   - **-n** – Specifies the subject name for the root CA. The convention is to prefix the subject name with "CN = " for "Common Name".
   - **-r** – Specifies that the certificate will be self-signed. This means that certificates created with this switch will act as a root certificate.
   - **-sv** – Specifies the file that will contain the private key of the certificate. The file is always created, if it does not exist. This will allow creating certificates using the private key file for signing and key generation.
   - RootCATest.cer – Specifies the name of the file containing the public key of the certificate. The RootCATes.cer file will not have the private key. This is the certificate that will be installed in the store for trust chain validation on the client and server machines.

3. In the **Create Private Key Password** dialog box, enter a password, confirm the password, and then click **OK**. Optionally, you can click **None** without entering the password, but this is not recommended for security reasons.

4. In the **Enter Private Key Password** dialog box, enter the password again and then click **OK**.
   This is the password needed to access the private key file RootCATest.pvk in order to generate the file RootCATest.cer containing the public key.

This step creates a certificate named RootCATest.cer and a private key file named RootCATest.pvk

Step 2 – Create a Certificate Revocation List File from the Root Certificate

In this step you will create a certificate revocation list file that is going to be imported in the correct certificate stores of the client and service machines, so you will create a CRL(certificate revocation list) for the temporary root certificate. The CRL is necessary because WCF clients check for the CRL when doing certificate validation.

1. Open a Visual Studio command prompt and browse to the location where you want to save the CRL file for the root certificate.
2. Run the following command to create the CRL file.

```
makecert -crl -n "CN=RootCATest" -r -sv RootCATest.pvk RootCATest.crl
```

In this command:

- **-crl** – Specifies that you want to generate the CRL file for the root certificate
- **-n** – Specifies the subject name for the CRL. The convention is to prefix the subject name with "CN = " for "Common Name". You can name it with the same name of the root certificate authority
- **-r** – Specifies that the CRL file will be self-signed. This means certificates revocation list files are created with this switch, will act as revocation list files for the root certification authority.
- **-sv** – Specifies the file that will contain the private key for the CRL file generation. The file is not created, it already exists. This will allow creating certification revocation list files using the private key file for signing.
- RootCaTest.crl is the CRL file created with the command

Step 3 – Install Your Root Certificate Authority Certificate on the Server and Client Machines

In this step, you will install the certificate in the **Trusted Root Certification Authorities** location on both the server and client machines. All certificates that are signed with this certificate will be trusted by the client machine.

Important: Be sure to delete this certificate from the store after you have finished developing and testing for your application.

Repeat the following steps on both client and the server machines:

1. Copy the RootCATest.cer file to the client and server machines.
2. Click **Start** and then click **Run**.
3. In the command line, type **MMC** and **then** click **OK**.
4. In the Microsoft Management Console, on the **File** menu, click **Add/Remove Snap-in**.
5. In the **Add Remove Snap-in** dialog box, click **Add**.
6. In the **Add Standalone Snap-in** dialog box, select **Certificates** and then click **Add**.
7. In the **Certificates snap-in** dialog box, select the **Computer account** radio button because the certificate needs to be made available to all users, and then click **Next**.
8. In the **Select Computer** dialog box, leave the default **Local computer: (the computer this console is running on)** selected and then click **Finish**.
9. In the **Add Standalone Snap-in** dialog box, click **Close**.
10. In the **Add/Remove Snap-in** dialog box, click **OK**.
11. In the left pane, expand the **Certificates (Local Computer)** node, and then expand the Trusted Root Certification Authorities folder.
12. Under **Trusted Root Certification Authorities**, right-click the **Certificates** subfolder, select **All Tasks**, and then click **Import**.
13. On the **Certificate Import Wizard** welcome screen, click **Next**.
14. On the **File to Import** screen, click **Browse**.
15. Browse to the location of the signed Root Certificate Authority RootCATest.cer file copied in step 1, select the file, and then click **Open**.
16. On the File to Import screen, click **Next**.
17. On the Certificate Store screen, accept the default choice and then click **Next**.
18. On the Completing the Certificate Import Wizard screen, click **Finish**.

The signed root CA certificate is now installed in the Trusted Root Certification Authorities store. You can expand the Certificates subfolder under **Trusted Root Certification Authorities** to see the RootCATest certificate installed properly.

Step 4 – Install the Certificate Revocation List File on the Server and Client Machines

In this step, you will install the certificate revocation list (CRL) from the file in the **Trusted Root Certification Authorities** location on both the server and client machines. The certificate revocation list is checked during certificate validation process.

Important: Be sure to delete the certificate from the store after you have finished developing and testing for your application.

Repeat the following steps on both client and the server machines:

1. Copy the RootCATest.crl file to the client and server machines.
2. Click **Start** and then click **Run**.
3. In the command line, type **MMC** and **then** click **OK**.
4. In the Microsoft Management Console, on the **File** menu, click **Add/Remove Snap-in**.
5. In the **Add Remove Snap-in** dialog box, click **Add**.
6. In the **Add Standalone Snap-in** dialog box, select **Certificates** and then click **Add**.
7. In the **Certificates snap-in** dialog box, select the **Computer account** radio button because the certificate needs to be made available to all users, and then click **Next**.
8. In the **Select Computer** dialog box, leave the default **Local computer: (the computer this console is running on)** selected and then click **Finish**.
9. In the **Add Standalone Snap-in** dialog box, click **Close** .
10. In the **Add/Remove Snap-in** dialog box, click **OK** .
11. In the left pane, expand the **Certificates (Local Computer)** node, and then expand the Trusted Root Certification Authorities folder.
12. Under **Trusted Root Certification Authorities**, right-click the **Certificates** subfolder, select **All Tasks**, and then click **Import**.
13. On the **Certificate Import Wizard** welcome screen, click **Next**.
14. On the **File to Import** screen, click **Browse**.
15. On the **Files of Type** select **Certificate Revocation List**
16. Browse to the location of the signed Root Certificate Authority RootCATest.crl file copied in step 1, select the file, and then click **Open**.
17. On the File to Import screen, click **Next**.
18. On the Certificate Store screen, accept the default choice and then click **Next**.
19. On the Completing the Certificate Import Wizard screen, click **Finish**.

The certificate revocation list for the root CA certificate is now installed in the **Trusted Root Certification Authorities** store. You can click on **Trusted Root Certification Authorities** folder then press F5. A subfolder called **Certificate Revocation List** will be displayed. You can expand this folder and you will see the RootCATest certificate revocation list installed properly.

Step 5 – Create and Install Your Temporary Service Certificate

In this step, you create and install the temporary certificate on the server machine from the signed root CA created in the previous step.

1. Open a Visual Studio command prompt and browse to the location where you have the root CA certificate and private key file created.
2. Run following command for creating a certificate signed by the root CA certificate:

   makecert -sk MyKeyName -iv RootCATest.pvk -n "CN=tempCert" -ic RootCATest.cer -sr localmachine -ss my -sky exchange -pe

   In this command:

   - **-sk** – Specifies the key container name for the certificate. This needs to be unique for each certificate you create.
   - **-iv** – Specifies the private key file from which the temporary certificate will be created. You need to specify the root certificate private key file name that was created in previous step and make sure that it is available in the current directory. This will be used for signing the certificate and key generation.
   - **-n** – Specifies the key subject name for the temporary certificate. The convention is to prefix the subject name with "CN = " for "Common Name".
   - **-ic** – Specifies the file containing the root CA certificate file generated in previous step.
   - **-sr** – Specifies the store location where the certificate will be installed. The default location is Currentuser, but since the certificate needs to be available to all users, you should use the localmachine option.
   - **-ss** – Specifies the store name for the certificate. **My** is the personal store location of the certificate.
   - **-sky** – Specifies the key type, which could be either **signature** or **exchange**. Using **exchange** makes certificate capable of signing and encrypting the message.
   - **-pe** – Specifies that the private key is generated in the certificate and install with it in the certificate store. When you double click the certificate in the general tab you should see at the bottom a message **"You have a private key that corresponds to this certificate"**. For message security this is a requirement. If the certificate does not have the corresponding private key, it cannot be used for message security.

3. In the **Enter Private Key Password** dialog box, enter the password for the root CA privatekeyfile specified in STEP 2, and then click **OK**.

Step 6 – Give the WCF Process Identity Access to the Temporary Certificate's Private Key

In this step, you give the process identity of the WCF service access permissions to the certificate private key. If your service is hosted in Internet Information Services (IIS), the identity typically is "NT AUTHORITY\NETWORK SERVICE"; in a production scenario, or if your service is hosted in w
Windows service it could be a custom domain service account.

1. Open a Visual Studio command prompt.
2. Run the following command:

   FindPrivateKey.exe My LocalMachine -n "CN=tempCert"

   In this command:

   - **My** – the store name where you have installed your temporary certificate.
   - **LocalMachine** – the store location for your certificate.
   - **–n "CN=tempCert"** – the common name for your temporary certificate.

   Note: If FindPrivateKey is not on your machine, download the WCF samples, including the FindPrivateKey tool, at http://www.microsoft.com/downloads/details.aspx?FamilyId=2611A6FF-FD2D-4F5B-A672-C002F1C09CCD&displaylang=en

   FindPrivateKey returns the location of the private key for the certificate, similar to "**C:\Documents and Settings\All Users\Application Data\Microsoft\Crypto\RSA\Machinekeys\4d657b73466481beba7b0e1b5781db81_c225a308-d2ad-4e58-91a8-6e87f354b030**".
3. Run the following command to assign access permissions to the process identity of the WCF service.
   Note: You should give read-only permissions to the private key

   ```
   cacls.exe "C:\Documents and Settings\All Users\Application Data\Microsoft\Crypto\RSA\Machinekeys\4d657b73466481beba7b0e1b5781db81_c225a308-d2ad-4e58-91a8-6e87f354b030" /E /G "NT AUTHORITY\NETWORK SERVICE":R
   ```

   In this command:

   - **/E** – Edits the access control list (ACL) of the private key instead or replacing it. You should never replace the ACL but should only add the necessary permission to the process identity.
   - **/G** – Grants the permission to the process identity.

- **:R –** Gives read-only permissions to "NT AUTHORITY\NETWORK SERVICE".

4. Run the following command to verify the permissions on the private key. This will display all the identities and the permissions that have access to the private key

   ```
   cacls.exe "C:\Documents and Settings\All Users\Application Data\Microsoft\Crypto\RSA\Machinekeys\4d657b73466481beba7b0e1b5781db81_c225a308-d2ad-4e58-91a8-6e87f354b030"
   ```

   You should see the following in the output from this command:

   ```
   NT AUTHORITY\NETWORK SERVICE:R
   ```

Note: If you are running Microsoft Windows® XP, give the certificate permissions for the ASPNET identity instead of the NT Authority\Network Service identity, because the IIS process runs under the ASPNET account in Windows XP.

Deployment Considerations

Temporary certificates should only be used for development and testing purposes. For real-world production environments, use a certificate provided by a CA such as Microsoft Windows Server® 2003 Certificate Services or a third party.

Additional Resources

- For more information on how to work with temporary certificates, see “How to: Create Temporary Certificates for Use During Development” at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on how to view certificates using the MMC snap in, see “How to: View Certificates with the MMC Snap-in” at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see “Differences Between Service Certificate Validation Done by Internet Explorer and WCF” at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see “Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security” at http://msdn2.microsoft.com/en-us/library/aa702579.aspx
- For more information on how to obtain a certificate, see “How to: Obtain a Certificate (WCF)” at http://msdn2.microsoft.com/en-us/library/aa702761.aspx
- For more information on WCF command-line tools, see “Windows Communication Foundation Tools” at http://msdn2.microsoft.com/en-us/library/ms732015.aspx
- To download the WCF samples, including the FindPrivateKey tool, see “Windows Communication Foundation (WCF), Windows Workflow Foundation (WF) and

Windows CardSpace Samples" at
http://www.microsoft.com/downloads/details.aspx?FamilyId=2611A6FF-FD2D-4F5B-A672-C002F1C09CCD&displaylang=en

How To: Create and Install Temporary Certificates in WCF for Transport Security During Development

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft .NET Framework 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through to the process of creating and installing temporary certificates to be used during the development and testing of WCF services that implement transport security. The article explains the process of creating, configuring, and installing these temporary certificates to work with WCF.

Contents

Objectives

- Learn how to create a temporary root certificate authority to be used to sign your temporary service certificate.
- Learn how to create temporary certificates for transport security using makecert.exe utility.
- Learn where to store temporary certificates to be used by WCF when utilizing transport security.

Overview

When developing a WCF service that uses X.509 certificates to provide transport security, it is necessary to work with temporary certificates. This is because production certificates are expensive and may not be readily available. There are two options for specifying trust on a certificate:

- **Peer trust** – Validates the certificate directly.
- **Chain trust** – Validates the certificate against the issuer of a certificate known as a root authority.

This How To article discusses the chain trust option because it is the most commonly used approach in Business-to-Business (B2B) scenarios.

To use chain trust validation during development time, you create a self-signed root certificate authority (CA) and place it in the Trusted Root Certification Authority store. The certificate used by WCF is then created and signed by the root self-signed certificate and installed in the LocalMachine store.

You will use makecert.exe to create a certificate to act as your root certificate authority (CA). You will then use your root CA certificate to sign additional certificates for your WCF services. Finally, you will configure IIS to use your temporary certificate.

Summary of Steps

- Step 1 – Create a Certificate to Act as Your Root Certificate Authority
- Step 2 – Install Your Root Certificate Authority on the Server and Client Machines
- Step 3 – Create and Install Your Temporary Service Certificate
- Step 4 – Configure Your Temporary Service Certificate in IIS to Support SSL

Step 1 – Create a Certificate to Act as Your Root Certificate Authority

In this step, you use the makecert tool in the service host machine to create a root CA that will be used to sign your temporary certificate.

1. Open a Visual Studio command prompt and browse to the location where you want to save the certificate files.
2. Run the following command to create the root CA

   ```
   makecert -n "CN=RootCATest" -r -sv RootCATest.pvk RootCATest.cer
   ```

   In this command:

   - **-n** – Specifies the subject name for the root CA. The convention is to prefix the subject name with "CN = " for "Common Name".
   - **-r** – Specifies that the certificate will be self-signed.
   - **-sv** – Specifies the file that contains the private key of the certificate.
   - RootCATest.cer – Specifies the name of the file containing the public key of the certificate.

3. In the **Create Private Key Password** dialog box, enter a password, confirm the password, and then click **OK**. Optionally, you can click **None** without entering the password, but this is not recommended for security reasons.
4. In the **Enter Private Key Password** dialog box, enter the password again and then click **OK**. This is the password needed to access the private key file RootCATest.pvk in order to generate the file RootCATest.cer containing the public key.

This step creates a certificate named RootCATest.cer and a private key file named RootCATest.pvk.

Step 2 – Install Your Root Certificate Authority on the Server and Client Machines

In this step, you install the certificate in the **Trusted Root Certification Authorities** location on both the client and server machines. All certificates that are signed with this certificate will be trusted by the client and by the server.

Important: Be sure to delete this certificate from the store after you have finished developing and testing for your application.

Repeat the following steps on both the client and the server machines:

1. Copy the RootCATest.cer file to the client and server machines.
2. Click **Start** and then click **Run**.
3. In the command line, type **MMC** and then click **OK**.
4. In the Microsoft Management Console (MMC), on the **File** menu, click **Add/Remove Snap-in**.
5. In the **Add Remove Snap-in** dialog box, click **Add**.
6. In the **Add Standalone Snap-in** dialog box, select **Certificates** and then click **Add**.
7. In the **Certificates snap-in** dialog box, select the **Computer account** radio button because the certificate needs to be made available to all users, and then click **Next**.
8. In the **Select Computer** dialog box, leave the default **Local computer: (the computer this console is running on)** selected and then click **Finish**.
9. In the **Add Standalone Snap-in** dialog box, click **Close**.
10. In the **Add/Remove Snap-in** dialog box, click **OK**.
11. In the left pane, expand the **Certificates (Local Computer)** node, and then expand the Trusted Root Certification Authorities folder.
12. Under **Trusted Root Certification Authorities**, right-click the Certificates subfolder, select **All Tasks**, and then click **Import**.
13. On the Certificate Import Wizard welcome screen, click **Next**.
14. On the File to Import screen, click **Browse**.
15. Browse to the location of the signed Root Certificate Authority RootCATest.cer file copied in step 1, select the file, and then click **Open**.
16. On the File to Import screen, click **Next**.
17. On the Certificate Store screen, accept the default choice and then click **Next**.
18. On the Completing the Certificate Import Wizard screen, click **Finish**.

The signed root CA certificate is now installed in the Trusted Root Certification Authorities store. You can expand the Certificates subfolder under **Trusted Root Certification Authorities** to see the RootCATest certificate installed properly.

Important: If you do not install the self-signed root CA on the client machine, the proxy generation either from the UI or from the command prompt using svcutil will fail with the following error message:

```
“There was an error downloading
'https://MachineName/servicefolder/Service.svc'.
 The underlying connection was closed: Could not establish trust relationship
 for the SSL/TLS secure channel.
The remote certificate is invalid according to the validation procedure.”
```

Note: Unless the root certificate is present in the certificate authority store, the Http layer raises an exception when a client communicates over Https: **"**the remote server returned an error (403) forbidden**"**.

Step 3 – Create and Install Your Temporary Service Certificate

In this step, you create and install the temporary certificate on the server machine from the signed root CA created in the previous step.

1. Open a Visual Studio command prompt and browse to the location where you have the root CA certificate and private key file installed. The files will be named RootCATest.cer and RootCATest.pvk.
2. Run the following command for creating a certificate signed by the root CA certificate:

   ```
   makecert -sk <<UniqueKeyName>> -iv RootCATest.pvk -n
   "CN=<<MachineName>>" -ic RootCATest.cer -sr localmachine -ss my -sky
   exchange -pe
   ```

   In this command:

   - **-sk** – Specifies the key container name for the certificate. This name needs to be unique for each certificate you create.
   - **-iv** – Specifies the private key file from which the temporary certificate will be created. You need to specify the root certificate private key file name and make sure that it is available in the current directory.
   - **-n** – Specifies the key subject name for the temporary certificate. If the name of the certificate does not match the **DNS** or **netbios** name, later proxy generation will fail.
   - **-ic** – Specifies the file containing the root CA certificate file generated in the previous step.
   - **-sr** – Specifies the store location where the certificate will be installed. The default location is Currentuser, but since the certificate needs to be available to all users, you should use the localmachine option.
   - **-ss** – Specifies the store name for the certificate. Specify **My** as the personal store location of the certificate.
   - **-sky** – Specifies the key type, which could be either **signature** or **exchange**. Using **exchange** makes the private key exportable, which is required for message security.
   - **-pe** – Specifies that the private key is exportable. This is useful if you want to export the key and use it in another machine for development or testing purposes.

3. In the **Enter Private Key Password** dialog box, enter the password for the root CA private key file specified in STEP 2, and then click **OK**.

Important: The subject name and key file name have to match the name of the machine you are installing this certificate on. If the name does not match you will see a certificate security error when accessing the service.

Step 4 – Configure Your Temporary Service Certificate in IIS to Support SSL

In this step, you configure the Web site in IIS to use the temporary certificate for Secure Sockets Layer (SSL) communication. This will enable SSL for the transport communication. These instructions pertain to IIS version 6. IIS 7 requires different steps.

1. Click **Start and** then click **Run**.
2. In the **Run** dialog box, type **inetmgr** and then click **OK**.
3. In the **Internet Information Services (IIS) Manager** dialog box, expand the **(local computer)** node, and then expand the **Web Sites** node.
4. Right-click **Default Web Site** and then click **Properties**.
5. In the **Default Web Site Properties** dialog box, click the **Directory Security** tab, and then in the **Secure Communications** section, click **Server Certificate**.
6. On the Welcome screen of the Web Server Certificate Wizard, click **Next** to continue.
7. On the Server Certificate screen, select the **Assign an existing certificate** radio button option, and then click **Next**. If you have a preexisting certificate that you can remove, first remove the certificate using the "Remove the current certificate" option, then proceed with step 5.
8. On the Available Certificates screen, select the certificate you created and installed in previous step, and then click **Next**.
9. Verify the information on the certificate summary screen, and then click **Next.**
10. Click **Finish** to complete the certificate installation.
11. In the **Default Web Site Properties** dialog box, click **OK**.

Deployment Considerations

Temporary certificates should only be used for development and testing purposes. For real-world production environments, use a certificate provided by a CA such as Microsoft Windows Server® 2003 Certificate Services or a third party.

Additional Resources

- For more information on how to work with temporary certificates, see "How to: Create Temporary Certificates for Use During Development" at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on how to view certificates by using the MMC snap in, see "How to: View Certificates with the MMC Snap-in" at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see "Differences Between Service Certificate Validation Done by Internet Explorer and WCF" at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see "Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security" at http://msdn2.microsoft.com/en-us/library/aa702579.aspx

- For more information on how to obtain a certificate, see “How to: Obtain a Certificate (WCF)” at http://msdn2.microsoft.com/en-us/library/aa702761.aspx
- For more information on Windows Foundation command-line tools, see “Windows Communication Foundation Tools” at http://msdn2.microsoft.com/en-us/library/ms732015.aspx

How To – Create and Install Temporary Client Certificates in WCF During Development

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft .NET Framework 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of creating and installing temporary certificates to be used during the development and testing of WCF services that implement certificate client authentication. The article explains the process of creating, configuring, and installing these temporary certificates to work with WCF.

Contents

Objectives

- Learn how to create a root certificate for the temporary certificate used for certificate authentication in WCF.
- Learn how to create a root certification revocation list file for the root certificate used to validate the revocation.
- Learn how to create a temporary certificate for certificate authentication in WCF.
- Learn how to install the temporary certificate.
- Learn how to install the root certificate for the temporary certificate.
- Learn how to install the root certification revocation list for the root certificate.

Overview

When developing a WCF service that uses X.509 certificates to do certificate authentication, it is necessary to work with temporary certificates. This is because production certificates are expensive and may not be readily available. There are two options for specifying trust on a certificate:

- **Peer trust** – Validates the certificate directly.
- **Chain trust** – Validates the certificate against the issuer of a certificate known as a root authority.

This How To article describes the chain trust option because it is the most commonly used approach in Business-to-Business (B2B) scenarios, and it is the default validation for WCF, when using certificate authentication.

Additionally, a certificate revocation list (CRL) validation is performed during the certificate authentication process. This validation checks the list of certificates that were revoked by the root certificate. Three modes of revocation exist:

- **Online** – The CRL list is retrieved and checked online,this requires a network connection to retrieve the CRL and check each address listed.
- **Offline** – The CRL list is retrieved and checked online and is then cached for subsequent offline validation.
- **NoCheck** – No validation is performed.

For the purposes of this How To article, the CRL is checked without configuration changes when using certificate authentication.

To use chain trust validation during development time, you first create a self-signed root certificate authority (CA) and install it in the Trusted Root Certification Authority in the Local Machine. The certificate used by WCF is then created and signed by the root self-signed certificate and installed in the Personal store of the Local Machine. To allow CRL validation to succeed, you create a self-signed root CRL file and install it in the Trusted Root Certification Authority store of the Local Machine.

You will use makecert.exe to create a private key file and a certificate to act as your root CA. You will then create a CRL file from the private key that will act as your revocation list file for the root CA. You will have to install the root certificate and CRL file. Finally, you will create and install the temporary certificate from the root certificate, using the private key to sign and generate the key.

Summary of Steps

- **Step 1 – Create a Certificate to Act as Your Client Root Certificate Authority**
- **Step 2 – Create a Certificate Revocation List File from the Root Certificate**
- **Step 3 – Install Your Client Root Certificate Authority on the Client and Server Machines**
- **Step 4 – Install the Certificate Revocation List File on the Server and Client Machines**
- **Step 5 – Create and Install Your Temporary Client Certificate**

Step 1 – Create a Certificate to Act as Your Client Root Certificate Authority

In this step, you use the makecert tool to create a root CA that will be used to sign your certificate. This certificate will be self-signed and will only have the public key that will be used to perform trust chain validation, when authenticating clients with the certificate. The self-signed certificate will act as a root CA itself, instead of pointing to a Root authority in a chain of trust.

1. Open a Visual Studio command prompt and browse to the location where you want to save the certificate files.
2. Run the following command to create the root CA:

   ```
   makecert -n "CN=RootCaClientTest" -r -sv RootCaClientTest.pvk
   RootCaClientTest.cer
   ```

   In this command:

   - **-n** – Specifies the subject name for the root CA. The convention is to prefix the subject name with "CN = " for "Common Name".
   - **-r** – Specifies that the certificate will be self-signed. This means that certificates created with this switch will act as a root certificate.
   - **-sv** – Specifies the file that will contain the private key of the certificate. The file is always created, if it does not already exist. This will allow creating certificates using the private key file for signing and key generation.
   - **RootCaClientTest.cer** – Specifies the name of the file containing the public key of the certificate. The RootCATes.cer file will not have the private key. This is the certificate that will be installed in the store for trust chain validation on the client and server machines.

3. In the **Create Private Key Password** dialog box, enter a password, confirm the password, and then click **OK**.
   Optionally, you can click **None** without entering the password, but this is not recommended for security reasons.
4. In the **Enter Private Key Password** dialog box, enter the password again and then click **OK**.
   This is the password needed to access the private key file RootCaClientTest.pvk in order to generate the file RootCaClientTest.cer containing the public key.

This step creates a certificate named RootCaClientTest.cer and a private key file named RootCaClientTest.pvk.

Step 2 – Create a Certificate Revocation List File from the Root Certificate

In this step, you create a CRL file that will be imported into the correct certificate stores of the client and service machines. You create a CRL for the temporary root certificate; the CRL is necessary because WCF clients check for the CRL when validating certificates.

1. Open a Visual Studio command prompt and browse to the location where you want to save the CRL file for the root certificate.
2. Run the following command to create the CRL file:

```
makecert -crl -n "CN=RootCaClientTest" -r -sv RootCaClientTest.pvk
RootCaClientTest.crl
```

In this command:

- **-crl** – Specifies that you want to generate the CRL file for the root certificate.
- **-n** – Specifies the subject name for the CRL. The convention is to prefix the subject name with "CN = " for "Common Name". You can give it the same name as the root CA.
- **-r** – Specifies that the CRL file will be self-signed. This means that CRL files created with this switch will act as revocation list files for the root CA.
- **-sv** – Specifies the file that will contain the private key for CRL file generation. The file is not created since it already exists. This allows creation of CRL files using the private key file for signing.
- RootCaClientTest.crl – Is the CRL file created with the command.

Step 3 – Install Your Client Root Certificate Authority on the Client and Server Machines

In this step, you install the client root CA in the Trusted Root Certification Authorities location on both the server and client machines. All certificates that are signed with this certificate will be trusted by the client machine.

Important: Be sure to delete this certificate from the store after you have finished developing and testing your application.

Repeat the following steps on both the client and server machines:

1. Copy the RootCaClientTest.cer file to the client and server machines.
2. Click **Start** and then click **Run**.
3. In the command line, type **MMC** and then click **OK**.
4. In the Microsoft Management Console, on the **File** menu, click **Add/Remove Snap-in**.
5. In the **Add Remove Snap-in** dialog box, click **Add**.

6. In the **Add Standalone Snap-in** dialog box, select **Certificates** and then click **Add**.
7. In the **Certificates snap-in** dialog box, select the **Computer account** radio button (because the certificate needs to be made available to all users), and then click **Next**.
8. In the **Select Computer** dialog box, leave the default **Local computer: (the computer this console is running on)** selected and then click **Finish**.
9. In the **Add Standalone Snap-in** dialog box, click **Close**.
10. In the **Add/Remove Snap-in** dialog box, click **OK**.
11. In the left pane, expand the **Certificates (Local Computer)** node, and then expand the Trusted Root Certification Authorities folder.
12. Under **Trusted Root Certification Authorities**, right-click the **Certificates** subfolder, click **All Tasks**, and then click **Import**.
13. On the **Certificate Import Wizard** welcome screen, click **Next**.
14. On the **File to Import** screen, click **Browse**.
15. Browse to the location of the signed root CA RootCaClientTest.cer file copied in Step 1, select the file, and then click **Open**.
16. On the File to Import screen, click **Next**.
17. On the Certificate Store screen, accept the default choice and then click **Next**.
18. On the Completing the Certificate Import Wizard screen, click **Finish**.

The signed root CA certificate is now installed in the Trusted Root Certification Authorities store. You can expand the Certificates subfolder under **Trusted Root Certification Authorities** to see the RootCaClientTest certificate installed properly.

Step 4 – Install the Certificate Revocation List File on the Server and Client Machines

In this step, you install the CRL from the file in the Trusted Root Certification Authorities location on both the server and client machines. The CRL is checked during the certificate validation process.

Important: Be sure to delete the certificate from the store after you have finished developing and testing your application.

Repeat the following steps on both the client and server machines:

1. Copy the RootCaClientTest.crl file to the client and server machines.
2. Click **Start** and then click **Run**.
3. In the command line, type **MMC** and then click **OK**.
4. In the Microsoft Management Console, on the **File** menu, click **Add/Remove Snap-in**.
5. In the **Add Remove Snap-in** dialog box, click **Add**.
6. In the **Add Standalone Snap-in** dialog box, select **Certificates** and then click **Add**.
7. In the **Certificates snap-in** dialog box, select the **Computer account** radio button (because the certificate needs to be made available to all users), and then click **Next**.

8. In the **Select Computer** dialog box, leave the default **Local computer: (the computer this console is running on)** selected and then click **Finish**.
9. In the **Add Standalone Snap-in** dialog box, click **Close**.
10. In the **Add/Remove Snap-in** dialog box, click **OK**.
11. In the left pane, expand the **Certificates (Local Computer)** node, and then expand the Trusted Root Certification Authorities folder.
12. Under **Trusted Root Certification Authorities**, right-click the **Certificates** subfolder, select **All Tasks**, and then click **Import**.
13. On the Certificate Import Wizard welcome screen, click **Next**.
14. On the File to Import screen, click **Browse**.
15. On the Files of Type screen, select **Certificate Revocation List**.
16. Browse to the location of the signed root CA RootCaClientTest.crl file copied in Step 1, select the file, and then click **Open**.
17. On the File to Import screen, click **Next**.
18. On the Certificate Store screen, accept the default choice and then click **Next**.
19. On the Completing the Certificate Import Wizard screen, click **Finish**.

The CRL for the root CA certificate is now installed in the Trusted Root Certification Authorities store. You can click the Trusted Root Certification Authorities folder and then press F5 to display subfolder named Certificate Revocation List. You can expand this folder to see the RootCaClientTest certificate revocation list installed properly.

Step 5 – Create and Install Your Temporary Client Certificate

In this step, you create the temporary certificate from the signed root CA created in the previous step and install it on the server machine.

1. Open a Visual Studio command prompt and browse to the location where the root CA certificate and private key file you created are stored.
2. Run the following command for creating a certificate signed by the root CA certificate:

   ```
   makecert -sk MyKeyName -iv RootCaClientTest.pvk -n "CN=tempClientcert"
   -ic RootCaClientTest.cer -sr currentuser -ss my -sky signature -pe
   ```

   In this command:

   - **-sk** – Specifies the key container name for the certificate. This needs to be unique for each certificate you create.
   - **-iv** – Specifies the private key file from which the temporary certificate will be created. You need to specify the root certificate private key file name that was created in the previous step and make sure that it is available in the current directory. This will be used for signing the certificate and key generation.
   - **-n** – Specifies the key subject name for the temporary certificate. The convention is to prefix the subject name with "CN = " for "Common Name".

- **-ic** – Specifies the file containing the root CA certificate file generated in the previous step.
- **-sr** – Specifies the store location where the certificate will be installed. The default location is **currentuser**. For certificate authentication, this is the default location that Microsoft Internet Explorer uses for when browsing Web sites that require a client certificate.
- **-ss** – Specifies the store name for the certificate. **My** is the personal store location of the certificate.
- **-sky** – Specifies the key type, which could be either **signature** or **exchange**. Using **signature** makes the certificate capable of signing and enables certificate authentication.
- **-pe** – Specifies that the private key is generated in the certificate and installed with it in the certificate store. When you double-click the certificate on the **General** tab, you should see the message "You have a private key that corresponds to this certificate" displayed at the bottom. This is a requirement for certificate authentication. If the certificate does not have the corresponding private key, it cannot be used for certificate authentication.

3. In the **Enter Private Key Password** dialog box, enter the password for the root CA privatekeyfile specified in Step 2, and then click **OK**.

Deployment Considerations

Temporary certificates should only be used for development and testing purposes. In real-world production environments, use a certificate provided by a CA such as Microsoft Windows Server® 2003 Certificate Server or a third party.

Additional Resources

- For more information on working with temporary certificates, see "How to: Create Temporary Certificates for Use During Development" at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on viewing certificates by using the Microsoft Management Console (MMC) snap in, see "How to: View Certificates with the MMC Snap-in" at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see "Differences Between Service Certificate Validation Done by Internet Explorer and WCF" at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see "Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security" at http://msdn2.microsoft.com/en-us/library/aa702579.aspx
- For more information on obtaining a certificate, see "How to: Obtain a Certificate (WCF)" at http://msdn2.microsoft.com/en-us/library/aa702761.aspx

- For more information on Windows Communication Foundation command-line tools, see "Windows Communication Foundation Tools" at http://msdn2.microsoft.com/en-us/library/ms732015.aspx

How To – Host WCF in a Windows Service Using TCP

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of hosting a WCF service within a Microsoft Windows® service.

Contents

Objectives

- Create a simple WCF service.
- Host your WCF service in a Windows service using Transmission Control Protocol (TCP).
- Create a simple client to consume your service.

Overview

WCF services can be self-hosted in an application (such as a console or a Windows Forms application), in a Windows service, in Internet Information Services (IIS) 6.0, or in IIS 7.0 with Windows Activation Services (WAS).

The advantages of hosting in a Windows service are:

- **Start on boot.** The service will automatically be started when the hosting computer is rebooted.
- **Recovery.** The service will be restarted by the Windows Service Control Manager if there is a failure.

- **Administration.** Administrators already know how to manage Windows services.
- **Security Identity.** Windows Service Control Manager allows you to choose an identity under which the process will run.
- **Binding flexibility.** Hosting in a Windows service allows you to choose any binding protocol. IIS 6.0 only allows HTTP bindings.

The disadvantages of hosting in a Windows service are:

- **Installation.** You must use a custom installer action or the .NET utility Installutil.exe.
- **Lack of enterprise features.** Windows services do not have the security, manageability, scalability, and administrative features that are included in IIS.

To host WCF in a Windows service, you need to create the WCF service, create a Windows service to host the WCF service, and then install and run the Windows service. For the purposes of this How To article, you will use installutil.exe on the command line to install the service. In a production environment, you can use a setup program to install the service.

Summary of Steps

- **Step 1 - Create a WCF Service**
- **Step 2 - Configure the WCF Endpoints to Use TCP and Set the Base Address**
- **Step 3 - Create a Windows Service**
- **Step 4 - Add the Service Installers to the Windows Service**
- **Step 5 - Modify the Windows Service to Host the WCF Service**
- **Step 6 – Install the Windows Service**
- **Step 7 – Create a Windows Forms Test Client Application**
- **Step 8 - Add a WCF Service Reference to the Client**
- **Step 9 – Test the Client and WCF Service**

Step 1 – Create a WCF service

In this step, you create a WCF service to test hosting in a Windows service.

1. In Visual Studio, click **File**, click **New**, and then click **Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **WCF Service Library**.
3. In the **Add New Project** dialog box, click **OK** to create the WCF Service Library project WcfServiceLibrary1.

Step 2 – Configure the WCF Endpoints to Use TCP and Set the Base Address

In this step, you modify the WCF configuration so that the endpoints use TCP instead of the default Hypertext Transfer Protocol (HTTP). You then set the base address for your service. Finally, you set **HttpGetEnabled** to **false**, since you will be running under TCP.

1. Right-click the App.config file of the WCF Service Library project and then click **Edit WCF Configuration**.
   If you do not see the Edit WCF Configuration option, on the **Tools** menu, click **WCF Service Configuration Editor.** Close the WCF Service Configuration Editor tool that appears. The option should now appear on the App.config context menu.
2. In the Configuration Editor, in the configuration section, expand **Services** and then expand **Endpoints**.
3. Select the first endpoint. Under **Endpoint Properties**, change the Binding from **wsHttpBinding** to **netTcpBinding**.
4. Select the second endpoint. Under **Endpoint Properties**, change the Binding from **mexHttpBinding** to **mexTcpBinding**.
5. Under **Service,** select the **Host** node, select the default address under the **BaseAddress** list, and then click **Edit**.
6. Set the base address to the following and then click **OK**:

```
net.tcp://localhost:8523/Service1
```

7. Under **Advanced**, expand the tree under **Service Behaviors**. Select **serviceMetadata** and change **HttpGetEnabled** from True to **False**.
8. Click **File** and then click **Save** to save your configuration changes.
9. In Visual Studio, verify your configuration, which should look as follows:

```
  <system.serviceModel>
    <services>
      <service behaviorConfiguration="WcfServiceLibrary1.Service1Behavior"
        name="WcfServiceLibrary1.Service1">
        <endpoint address="" binding="netTcpBinding" bindingConfiguration=""
          contract="WcfServiceLibrary1.IService1">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexTcpBinding"
bindingConfiguration=""
          contract="IMetadataExchange" />
        <host>
          <baseAddresses>
            <add baseAddress="net.tcp://localhost:8523/Service1" />
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
```

```
      <serviceBehaviors>
        <behavior name="WcfServiceLibrary1.Service1Behavior">
          <serviceMetadata httpGetEnabled="false" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
```

Note:

- The port 8523 and Service1 are arbitrary for this example. If you run into a conflict where the port is in use, change it.
- If you do not set **HttpGetEnabled** to False, you will get an exception in the Event Log when the service tries to start.

Step 3 – Create a Windows Service

In this step, you add a Windows Service project to your solution.

1. Right-click your solution, cllick **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, select **Windows**, and then select **Windows Service**.
3. In the **Name** field, leave the default name **WindowsService1** and then click **OK** to create a Windows service application.
4. Copy **App.config** from your WCF Service Library project to your Windows service project. In the WCF Service Library project, right-click the **App.config** file, click **Copy**, and then right-click your Windows service project and click **Paste**.

Step 4 – Add the Service Installers to the Windows Service

In this step, you add service installers to your Windows service.

1. Right-click **Service1.cs** and then click **View Designer**.
2. Right-click the designer view and then click **Add Installer**.
   This adds the **ProjectInstaller.cs** file with two objects, **serviceProcessInstaller1** and **serviceInstaller1**.
3. In the Design view of **ProjectInstaller.cs**, right-click **serviceProcessInstaller1** and then click **Properties**.
4. In the Properties pane, set the **Account** attribute to **NetworkService**.
5. Right-click **serviceInstaller1** and then click **Properties**.
6. In the Properties pane, set the **StartType** attribute to **Automatic**.

Step 5 – Modify the Windows Service to Host the WCF Service

In this step, you override the **OnStart()** and **OnStop()** methods to start and stop the WCF service inside the Windows service process.

1. Add a reference to **System.ServiceModel** to your Windows Service project. To do so, in your Windows service project, right-click the **References** node and then click **Add References**. In the **Add Reference** dialog box, select **System.ServiceModel** and then click **OK**.
2. Add a reference to your WCF Service Library project from your Windows service. To do so, in your Windows service project, right-click the **References** node and then click **Add References**. In the **Add Reference** dialog box, select the **Projects** tab. Select the WCF Service Library project, **WcfServiceLibrary1**, and then click **OK**.
3. Add the following **using** statements to the **Service1.cs** file in your Windows service project.

```
using System.ServiceModel;
using WcfServiceLibrary1;
```

4. Select **Service1.cs** and switch to code view.
5. Declare an internal static member of ServiceHost type, as follows:

```
internal static ServiceHost myServiceHost = null;
```

6. Override the **OnStart** method of the Windows service, to open the service host as follows:

```
protected override void OnStart(string[] args)
{
   if (myServiceHost != null)
   {
       myServiceHost.Close();
   }
   myServiceHost = new ServiceHost(typeof(Service1));
   myServiceHost.Open();
}
```

7. Override the **OnStop** method of the Windows service, to close the service host as follows:

```
protected override void OnStop()
{
   if (myServiceHost != null)
   {
      myServiceHost.Close();
      myServiceHost = null;
   }
}
```

8. Verify that your Service1.cs resembles the following:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
```

```
using System.Linq;
using System.ServiceProcess;
using System.Text;
using System.ServiceModel;
using WcfServiceLibrary1;

namespace WindowsService1
{
    public partial class Service1: ServiceBase
    {
        internal static ServiceHost myServiceHost = null;

        public WCFServiceHost1()
        {
            InitializeComponent();
        }
        protected override void OnStart(string[] args)
        {
            if (myServiceHost != null)
            {
                myServiceHost.Close();
            }
            myServiceHost = new ServiceHost(typeof(Service1));
            myServiceHost.Open();
        }
        protected override void OnStop()
        {
            if (myServiceHost != null)
            {
                myServiceHost.Close();
                myServiceHost = null;
            }
        }
    }
}
```

9. In the Solution Explorer, copy the App.config file from the WCF service project to the Windows service project so that the config file will be in both service binary folders after compiling.
10. Build your solution and verify that your project produces **WindowsService1.exe** in your project **\bin\debug** directory of your **WindowsService1** project.

Step 6 – Install the Windows Service

In this step, you install the Windows service and run it from the Services console.

1. Rebuild the solution and open a Visual Studio command prompt**.**
2. Browse to the **bin** directory of the project where WindowsService1.exe is located.
3. Run the following command to install the service:

```
Installutil WindowsService1.exe
```

4. Start your service. To do so, click **Start**, click **Run**, type **services.msc** and then click **OK**. Right-click your service and then click **Start**.

Note: If you have modified the service that is already installed, you can uninstall it by using following command:

```
Installutil /u WindowsService1.exe
```

Step 7 – Create a Windows Forms Test Client Application

In this step, you create a Windows Forms application named **Test Client** that you will use to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Application**.
3. In the **Name** field, type **Test Client** and then click **OK** to create a Windows Forms application.

Step 8 – Add a WCF Service Reference to the Client

In this step, you add a reference from your test client to your WCF service

1. Right-click your **Test client** project and select **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the **Address** to the following and then click **OK**

```
net.tcp://localhost:8523/Service1
```

Note: net.tcp://localhost:8523/Service1 is the base address that you set in Step 3 above.

Step 9 – Test the Client and WCF Service

In this step, you use the test client to ensure that the WCF service is running properly.

1. In your Client project, drag a button control onto your form.
2. Double-click the button control to show the underlying code.
3. In the code behind the button click, create an instance of the proxy, and call **GetData** of your WCF service. When you call the service, your current user security context will automatically be passed to your WCF Service. The code should look as follows:

```
        private void button1_Click(object sender, EventArgs e)
        {
            ServiceReference1.Service1Client myService = new
ServiceReference1.Service1Client();
            MessageBox.Show(myService.GetData(123), “My Service”);
            myService.Close();
        }
```

4. Right-click your client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5.
   When you click the button on the form, the message “You entered: 123” should appear.

Additional Resources

- For more information on hosting WCF in a Windows service, see “Hosting in a Windows Service Application” at http://msdn.microsoft.com/en-us/library/ms734781.aspx
- For more information on hosting WCF in a Windows service, see “How to: Host a WCF Service in a Managed Windows Service” at http://msdn.microsoft.com/en-us/library/ms733069.aspx

How To: Impersonate the Original Caller in WCF Calling from a Web Application

Applies To

- Microsoft Windows Communication Foundation (WCF) 3.5
- Microsoft® Visual Studio® 2008

Summary

This How To article shows you how to impersonate the original caller in a WCF service that has been called from a Web application. The article shows you how to configure the WCF service, implement impersonation, and test the service with a sample Web client.

Contents

Objectives

- Learn how to impersonate the original caller declaratively.
- Learn how to impersonate the original caller programmatically.
- Learn how to impersonate for specific WCF operations.
- Learn how to impersonate for all WCF operations.

Overview

WCF service code can make calls by using the security identity of the service (usually the host process identity or the identity of a service account), or by using the security identity of the original caller. The original caller may be an ASP.NET service account, or it may be the end user of the client application. You impersonate the original caller

whenever downstream code needs to authorize based on the original caller's identity. For instance, you may have authorization checks in business logic called by WCF, or you may want to access resources that have access control lists (ACLs) allowing specific user access.

You can impersonate the original caller either declaratively or programmatically, depending on the following circumstances:

- Impersonate the original caller declaratively when you want to access Microsoft Windows® resources that are protected with ACLs configured for your application's domain user accounts.
- Impersonate the original caller programmatically when you want to access resources predominantly by using the application's process identity, but specific sections of the operation need to use the original caller's identity.

Configure WCF to run using the identity of a lower-privilege account, such as the Network Service account, when it is not impersonating. Use the **OperationBehavior** attribute to impersonate declaratively on specific operations. Use the **Impersonate()** method in your code to impersonate programmatically.

In order to reduce attack surface, it is more secure to impersonate only on those operations in which it is necessary to do so. If you do want to impersonate on all operations, set the **ImpersonateCallerForAllOperations** attribute of ServiceAuthorizationBehavior to **True** in your application's configuration file.

Summary of Steps

- Step 1 – Create a Sample WCF Service
- Step 2 – Configure the WCF Service to Use Windows Authentication
- Step 3 – Configure the SPN Identity for the WCF Service Endpoint
- Step 4 – Implement Impersonation in the WCF Service
- Step 5 – Create a Web Application Test Client
- Step 6 – Add a WCF Service Reference to the Client
- Step 7 – Impersonate the Original Caller When Calling the WCF Service
- Step 8 – Configure the Web Application for Constrained Delegation
- Step 9 – Test the Client and WCF Service

Before You Begin

Before you can configure WCF to impersonate the original caller from a Web application, you must ensure that you have the following prerequisites in place:

- You must have Visual Studio 2008 installed.
- You must have Internet Information Services (IIS) installed and running.

- You must be in a Microsoft Active Directory® environment.
- You must have access to your Active Directory domain controller.

Step 1 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio, hosted in an IIS virtual directory.

1. In Visual Studio, select **File** > **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http** and specify the virtual directory to be created in the **Path (**e.g., http://localhost/WCFServiceImpersonation).
3. In the **New Web Site** dialog box, click **OK**to create a virtual directory, a solution file, and a sample WCF service for the solution.
4. In Microsoft Internet Explorer, browse to your WCF Service at http://localhost/WCFServiceImpersonation/Service.svc. You should see details of your WCF service in the browser.

Step 2 – Configure the WCF Service to Use Windows Authentication

By default, Visual Studio configures your WCF service to use wsHttpBinding with Windows Authentication and Message Security.

In Visual Studio, verify your configuration settings in Web.config. The configuration should look as follows:

```
…
<services>
  <service name="Service" behaviorConfiguration="ServiceBehavior">
    <endpoint address="" binding="wsHttpBinding" contract="IService">
      <identity>
        <dns value="localhost"/>
      </identity>
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
         contract="IMetadataExchange"/>
  </service>
</services>
…
```

Step 3 – Configure the SPN Identity for the WCF Service Endpoint

In this step, you configure the service principle name (SPN) identity under which the WCF service will run. This identity is usually the lower-privilege Network Service account. Use of this account will reduce the attack surface when your application is not impersonating.

1. Right-click the Web.config file and then and select the **Edit WCF Configuration** option.
2. If you do not see the **Edit WCF Configuration** option, click the **Tools** menu and select **WCF Service Configuration Editor**. Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu.
3. Expand the **Services** node, expand the **Service** node, and then expand the **Endpoints** node.
4. Select the first endpoint and verify it is configured to use **wsHttpBinding**.
5. Select the **Identity** tab and delete the **Dns** attribute value, which by default is set to "localhost".
6. Set the **ServicePrincipalName** attribute to "HOST/YourMachineName". This value depends on the identity which is used for running the WCF service. By default, the WCF service runs under the Network Service identity and is identified by the machine account in the network, hence you can use your machine name.

   Note: If WCF was running under a domain account, which will be true in a real-world production scenario, you will have to create a SPN for that identity and set the **ServicePrincipalName** attribute to the SPN appropriately.

7. In the configuration editor dialog box, on the **File** menu, click **Save**.
8. In Visual Studio, verify your configuration settings in Web.config. The configuration should look as follows:

```
...
<services>
  <service name="Service"
behaviorConfiguration="ServiceBehavior">
    <!-- Service Endpoints -->
    <endpoint address="" binding="wsHttpBinding"
contract="IService">
      <identity>
        <servicePrincipalName value="HOST/YourMachineName" />
        <dns value="" />
      </identity>
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange"/>
  </service>
</services>
...
```

Step 4 – Implement Impersonation in the WCF Service

Perform the following steps to declaratively impersonate specific operations:

1. In the Solution Explorer, expand the App_Code folder under your WCF Service project, and then open the Service.cs file.
2. Add a using statement for the **System.Security.Principal** namespace.
3. Set the impersonation required on the operation implementation of the specific operation as follows:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
  return string.Format("Hi, {0}, you have entered: {1}",
                       WindowsIdentity.GetCurrent().Name,
value);
}
```

Step 5 – Create a Web Application Test Client

In this step, you create a Web application that you will use to test the WCF service. In order to more closely emulate a production scenario, you should create the Web application on a separate physical machine.

1. In Visual Studio, select **File** > **New Web Site**.
2. In the **Templates** section, select **ASP.NET Web Site**. Make sure that the **Location** is set to **Http** and specify the virtual directory to be created in the **Path (**e.g., http://localhost/TestClientWebSite).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample ASP.NET Web site.
4. Open Internet Information Services (IIS) Manager by running the **inetmgr** command from the command line.
5. Expand the **Default Website** node, right-click the new **TestClientWebSite** virtual directory, and then select **Properties**.
6. In the **Properties** dialog box, click the **Directory Security** tab.
7. In the **Anonymous access and authentication control** section, click **Edit**.
8. In the **Authentication Methods** dialog box, clear the **Anonymous access** check box, and then select the **Integrated Windows authentication** check box.
9. In the **Authentication Methods** dialog box, click **OK**.
10. In the **Properties** dialog box, click **Apply** and then click **OK**.
11. Run the **iisreset** command from the command line.

Step 6 – Add a WCF Service reference to the client

In this step, you add a reference to your WCF service.

1. Right-click your client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF service (for example, http://localhost/WCFServiceImpersonation/Service.svc) and then click **Go**.
3. In the **Namespace** field, change “ServiceReference1” to “WCFTestService”.

4. Click **OK**.
   A reference to WCFTestService should appear beneath Service References in your client project.

Step 7 – Impersonate the Original Caller When Calling the WCF Service

In this step, you impersonate the original caller from the Web application and then call the WCF service.

1. View the designer for Default.aspx in your Web application.
2. Drag a **Button** control into the designer.
3. Double-click the **Button** control to show the underlying code.
4. Add a using statement for the **System.Security.Principle** namespace.
5. Use the **Impersonate()** method to impersonate the original caller.
6. Create an instance of the proxy and then call the **GetData** method of your WCF service. The code should look as follows:

```
Using System.Security.Principal;
…
protected void Button1_Click(object sender, EventArgs e)
{
    // Obtain the authenticated user's Identity and impersonate
the original caller
    using
(((WindowsIdentity)HttpContext.Current.User.Identity).Impersonate
())
    {
        WCFTestService.ServiceClient myService = new
WCFTestService.ServiceClient();
        Response.Write(myService.GetData(123) + "<br/>");
        myService.Close();
    }
}
…
```

Step 8 – Configure the Web Application for Constrained Delegation

In this step, you configure Active Directory to allow your Web application to use constrained delegation to access a remote WCF service. *Constrained delegation* allows the Web application to pass the identity of the original user to the WCF service.

If your ASP.NET application runs using the Network Service machine account, you must enable constrained delegation for your Web server computer. However, if your ASP.NET application runs under a custom domain account, you must enable protocol transition and constrained delegation for the custom domain account.

This How To article assumes that you are running your Web application under the Network Service machine account.

1. Start the **Microsoft Management Console (MMC) Active Directory Users and Computers** snap-in.
2. In the left pane of the MMC snap-in, click the **Computers** node.
3. In the right pane, double-click your Web server computer to display the **Properties** dialog box.
4. On the **Delegation** tab of the Properties window for the Web server computer, **Do not trust the computer for delegation** is selected by default. To use constrained delegation, select **Trust this computer for delegation to specified services only**.
   You specify precisely which service or services can be accessed in the bottom pane.
5. Beneath **Trust this computer for delegation to specified services only**, select **Use Kerberos only**.
6. Click **Add**. The **Add Services** dialog box appears.
7. Click **Users or computers**.
8. In the **Select Users or Computers** dialog box, type the name of your WCF service computer if you are running using Network Service. Alternatively, if you are running WCF by using a custom domain account, enter that account name instead. Click **OK**.
   You will see all the SPNs configured for the selected user or computer account.
9. To restrict access to the WCF service, select the **HOST** service, and then click **OK**.

For more information on constrained delegation, see “How To: Use Protocol Transition and Constrained Delegation in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998355.aspx

Step 9 – Test the Client and WCF Service

In this step, you access the WCF service via the ASP.NET Web site and ensure that it impersonates as expected.

1. Rebuild both your WCF Service and Web Application projects.
2. From the client machine, access the Web application and click the button.
3. The browser should display the message “Hi, <<logged in user id>>, you have entered: 123”.

Notice that if you remove impersonation from your service and run the client again, the user ID changes from your identity to the ASP.NET identity.

Additional Information

There are two options for impersonation:

- Impersonating the original caller declaratively
- Impersonating the original caller programmatically

This How To article showed how to impersonate specific operations declaratively because this is the most common and secure mechanism for impersonation. The following sections detail the complete set of options available for impersonation.

Impersonating the original caller declaratively

You can impersonate declaratively by applying the **OperationBehaviorAttribute** attribute on any operation that requires client impersonation. You can impersonate for all operations in the service, or limit the scope to specific operations. Impersonating all operations may increase the attack surface and negatively impact the security of your application.

Impersonating for specific operations

Perform the following steps to impersonate specific operations:

1. In the Solution Explorer, expand the App_Code folder under your WCF Service project, and then open the Service.cs file.
2. Add a using statement for the **System.Security.Principal** namespace.
3. Set the impersonation required on the operation implementation of the specific operation as follows:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
  return string.Format("Hi, {0}, you have entered: {1}",
                       WindowsIdentity.GetCurrent().Name,
value);
}
```

Impersonating all operations

Perform the following steps to impersonate all operations:

1. Right-click the Web.config file and then select the **Edit WCF Configuration** option.
2. Expand the **Advanced** node and then expand the **Service Behaviors** node.
3. Select the **ServiceBehavior** service behavior, and then click the **Add** button.
4. In the **Adding Behavior Extension Element Sections** dialog box, choose **serviceAuthorization** and then click **Add**.

5. Select the **serviceAuthorization** node and then set the **ImpersonateCallerForAllOperations** attribute to **True**.
6. In the configuration editor dialog box, on the **File** menu, click **Save**.
7. In Visual Studio, verify your configuration settings in Web.config. The configuration should look as follows:

```
...
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceAuthorization
impersonateCallerForAllOperations="true" />
    </behavior>
  </serviceBehaviors>
</behaviors>
...
```

Note: When impersonating for all operations, the **Impersonation** property of the **OperationBehaviorAttribute** applied to each method must also be set to either **Allowed** or **Required**.

Impersonating the original caller programmatically

Perform the following steps to impersonate the original caller programmatically:

1. In the Solution Explorer, expand the App_Code folder under your WCF Service project, and then open the Service.cs file.
2. Add a using statement for the **System.Security.Principal** namespace.
3. Use the **Impersonate**() call to impersonate the original caller, and then use **GetCurrent()** to revert back to the previous state, as follows:

```
public string GetData(int value)
{
 using
(ServiceSecurityContext.Current.WindowsIdentity.Impersonate())
 {
     // return the impersonated user (original users identity)
     return string.Format("Hi, {0}, you have entered: {1}",
          WindowsIdentity.GetCurrent().Name, value);
 }
}
```

Additional Resources

- For more information on impersonation, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms731090.aspx.
- For further information on impersonation, see “How to: Impersonate a Client on a Service” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx.

- For more information on constrained delegation, see “How To: Use Protocol Transition and Constrained Delegation in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998355.aspx

How To: Impersonate the Original Caller in WCF Calling from Windows Forms

Applies To

- Microsoft Windows Communication Foundation (WCF) 3.5
- Microsoft® Visual Studio® 2008

Summary

This How To article shows you how to impersonate the original caller in a WCF service that has been called from a Windows Forms application. The article shows you how to configure the WCF service, implement impersonation, and test the service with a sample Windows Forms client.

Contents

Objectives

- Learn how to impersonate the original caller declaratively.
- Learn how to impersonate the original caller programmatically.
- Learn how to impersonate for specific WCF operations.
- Learn how to impersonate for all WCF operations.

Overview

WCF service code can make calls by using the security identity of the service (usually the process identity or the identity of a service account), or by using the security identity of the original caller. The original caller may be an ASP.NET service account, or it may be the end user of the client application. You impersonate the original caller whenever downstream code needs to authorize based on the original caller's identity. For instance, you may have authorization checks in business logic called by WCF, or you may

want to access resources that have access control lists (ACLs) allowing specific user access.

You can impersonate the original caller either declaratively or programmatically, depending on the following circumstances:

- Impersonate the original caller declaratively when you want to access Microsoft Windows® resources that are protected with ACLs configured for your application's domain user accounts.
- Impersonate the original caller programmatically when you want to access resources predominantly by using the application's process identity, but specific sections of the operation need to use the original caller's identity.

Use the **OperationBehavior** attribute to impersonate declaratively on specific operations. Use the **Impersonate()** method in your code to impersonate programmatically.

In order to reduce attack surface, it is more secure to impersonate only on those operations in which it is necessary to do so. If you do want to impersonate on all operations, set the **ImpersonateCallerForAllOperations** attribute to **True** in your application's configuration file.

Summary of Steps

- Step 1 – Create a Sample WCF Service
- Step 2 – Configure the WCF Service to Use Windows Authentication
- Step 3 – Implement Impersonation in the WCF Service
- Step 4 – Create a Test Client Application
- Step 5 – Add a WCF Service Reference to the Client
- Step 6 – Test the Client and WCF Service

Step 1 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio, hosted in an Internet Information Services (IIS) virtual directory.

1. In Visual Studio, select **File** > **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http** and specify the virtual directory to be created in the **Path (**e.g., http://localhost/WCFServiceImpersonation).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory, a solution file, and a sample WCF service for the solution.

4. In Microsoft Internet Explorer, browse to your WCF Service at http://localhost/WCFServiceImpersonation/Service.svc. You should see details of your WCF service in the browser.

Step 2 – Configure the WCF Service to Use Windows Authentication

By default, Visual Studio configures your WCF service to use wsHttpBinding with Windows Authentication and Message Security.

In Visual Studio, verify your configuration settings in Web.config. The configuration should look as follows:

```
…
<services>
  <service name="Service" behaviorConfiguration="ServiceBehavior">
    <endpoint address="" binding="wsHttpBinding" contract="IService">
      <identity>
        <dns value="localhost"/>
      </identity>
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
         contract="IMetadataExchange"/>
  </service>
</services>
…
```

Step 3 – Implement Impersonation in the WCF Service

Perform the following steps to declaratively impersonate specific operations:

1. In the Solution Explorer, expand the App_Code folder under your WCF Service project, and then open the Service.cs file.
2. Add a using statement for the **System.Security.Principal** namespace.
3. Set the impersonation required on the operation implementation of the specific operation as follows:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
  return string.Format("Hi, {0}, you have entered: {1}",
                       WindowsIdentity.GetCurrent().Name,
value);
}
```

Step 4 – Create a Test Client Application

In this step, you create a Windows Forms application that you will use to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK** to create a Windows Forms application for testing.

Step 5 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF Service.

1. Right-click your client project and select **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF service: http://localhost/WCFServiceImpersonation/Service.svc
3. In the **Namespace** field, change ServiceReference1 to **WCFTestService**.
4. Click **OK**.
   A reference to WCFTestService should appear beneath Service References in your client project.

Step 6 – Test the Client and WCF Service

In this step, you access the WCF service and make sure that it impersonates as expected.

1. In your client project, drag a **Button** control onto your form.
2. Double-click the **Button** control to show the underlying code.
3. Create an instance of the proxy and call the **GetData** method of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                           WCFTestService.ServiceClient();
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the client project and select **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5. When you click the button on the form, it should display the message “Hi, <<logged in user id>>, you have entered: 123”.

Note that if you remove impersonation from your service and run the client again, the user ID changes from your identity to the ASP.NET identity.

Additional Information

There are two options for impersonation:

- Impersonating the original caller declaratively
- Impersonating the original caller programmatically

This How To article showed how to impersonate specific operations declaratively because this is the most common and secure mechanism for impersonation. The following sections detail the complete set of options available for impersonation.

Impersonating the original caller declaratively

You can impersonate declaratively by applying the **OperationBehaviorAttribute** attribute on any operation that requires client impersonation. You can impersonate for all operations in the service, or limit the scope to specific operations. Impersonating all operations may increase the attack surface and negatively impact the security of your application.

Impersonating for specific operations

Perform the following steps to impersonate specific operations:

1. In the Solution Explorer, expand the App_Code folder under your WCF Service project, and then open the Service.cs file.
2. Add a using statement for the **System.Security.Principal** namespace.
3. Set the impersonation required on the operation implementation of the specific operation as follows:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
  return string.Format("Hi, {0}, you have entered: {1}",
                       WindowsIdentity.GetCurrent().Name,
value);
}
```

Impersonating all operations

Perform the following steps to impersonate all operations:

1. Right-click the Web.config file and then select the **Edit WCF Configuration** option.
2. Expand the **Advanced** node and then expand the **Service Behaviors** node.
3. Select the **ServiceBehavior** service behavior, and then click **Add**.
4. In the **Adding Behavior Extension Element Sections** dialog box, choose **serviceAuthorization** and then click the **Add**.
5. Select the **serviceAuthorization** node and then set the **ImpersonateCallerForAllOperations** attribute to **True**.
6. In the configuration editor dialog box, on the **File** menu, click **Save**.
7. In Visual Studio, verify your configuration settings in Web.config. The configuration should look as follows:

```
...
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceAuthorization
impersonateCallerForAllOperations="true" />
    </behavior>
  </serviceBehaviors>
</behaviors>
...
```

Note: When impersonating for all operations, the **Impersonation** property of the **OperationBehaviorAttribute** applied to each method must also be set to either **Allowed** or **Required**.

Impersonating the original caller programmatically

Perform the following steps to impersonate the original caller programmatically:

1. In the Solution Explorer, expand the App_Code folder under your WCF Service project, and then open the Service.cs file.
2. Add a using statement for the **System.Security.Principal** namespace.
3. Use the **Impersonate**() call to impersonate the original caller, and then use **GetCurrent()** to revert back to the previous state, as follows:

```
public string GetData(int value)
{
 using
(ServiceSecurityContext.Current.WindowsIdentity.Impersonate())
 {
     // return the impersonated user (original users identity)
     return string.Format("Hi, {0}, you have entered: {1}",
          WindowsIdentity.GetCurrent().Name, value);
 }
}
```

Note: It is important to revert to impersonation. Failure to do so can form the basis for denial of service and elevation of privilege attacks. In the example above the **using** statement ensures that the impersonation is reverted after execution of the **using** block.

Additional Resources

- For more information on impersonation, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx.
- For further information on impersonation, see “How to: Impersonate a Client on a Service” athttp://msdn2.microsoft.com/en-us/library/ms731090.aspx.

How To – Perform Input Validation in WCF

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft .NET Framework 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article shows you how to perform input and data validation on parameters in WCF operations. The article shows you how to create a custom parameter inspector that can be used to validate input on both the server and on the client.

Contents

Objectives

- Learn how to create a custom parameter inspector to validate parameters in the operations of the service.
- Learn how to create a custom endpoint behavior that will consume the parameter inspector.
- Learn how to create a custom configuration element that will allow exposing of the custom endpoint behavior in the configuration file.

Overview

Input and data validation represents one important line of defense in the protection of your WCF application. You should validate all parameters exposed in WCF service operations to protect the service from attack by a malicious client. Conversely, you should also validate all return values received by the client to protect the client from attack by a malicious service.

WCF provides different extensibility points that allow you to customize the WCF runtime behavior by creating custom extensions. Message Inspectors and Parameter Inspectors are two extensibility mechanisms to gain higher control over the data passing between a client and a service. You should use parameter inspectors for input validation and message inspectors should be used only when you need to inspect the entire message flowing in and out of a service.

To perform input validation, you will build a .NET class and implement a custom parameter inspector in order to validate parameters on operations in your service. You will then implement a custom endpoint behavior to enable validation on both the client and the service. Finally, you will implement a custom configuration element on the class that allows you to expose the extended custom endpoint behavior in the configuration file of the service or the client.

For the purpose of this How-To we will create a WCF Service with wsHttpBinding and host it in IIS.

Summary of Steps

- Step 1 – Create a Sample WCF Service
- Step 2 – Configure the WCF Service to Use wsHttpBinding with Windows Authentication and Message Security
- Step 3 – Create a Windows Class Library Project That Will Contain the Three Classes Necessary for Parameter Validation
- Step 4 – Create a Class That Implements the Validation Logic
- Step 5 – Create a Class That Implements a Custom Endpoint Behavior
- Step 6 – Create a Class That Implements a Custom Configuration Element
- Step 7 – Add the Custom Behavior to the Configuration File
- Step 8 – Create an Endpoint Behavior and Map It to Use the Custom Behavior
- Step 9 – Configure the Service Endpoint to Use the Endpoint Behavior
- Step 10 – Test the Parameter Validator

Step 1 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio, hosted in an Internet Information Services (IIS) virtual directory.

1. In Visual Studio, on the **File** menu, click **New Web Site**.
2. In the **New Web Site** dialog box, in the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to Http.
3. In the **New Web Site** dialog box, set the new Web site address to **http://localhost/WCFTestParameterValidation** and then click **OK**.
4. By def**ault, your WC**F service will be configured to use **wsHttpBinding binding with message security and Win**dows Authenticat**io**n. Verify that your web.config configuration file looks as follows:

```
…
<services>
  <service name="Service" behaviorConfiguration="ServiceBehavior">
    <!-- Service Endpoints -->
```

```
    <endpoint address="" binding="wsHttpBinding" contract="IService">
      <identity>
        <dns value="localhost"/>
      </identity>
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange"/>
  </service>
</services>
…
```

Step 2 – Create a Windows Class Library for Parameter Validation

In this step, you create a Microsoft Windows® class library project that will include three classes for the custom parameter validation:

- One class to implement the parameter validation logic
- A second class to implement the endpoint behavior that will use the custom parameter class
- A third class to implement a behavior extension so that the validator will be visible in the service and client configuration files

Perform the following steps:

1. Open a new instance of Visual Studio, leaving your WCF service solution open.
2. In the new instance of Visual Studio, on the click **File** menu, click **New** and then click **Project**.
3. Expand **Visual C#**, click **Windows**, **and** then select **Class Library**.
4. In the **Name** field, type MyParamaterValidator and then click **OK**.
5. In the Solution Explorer, right click **References**, click **Add Reference**, click the **.NET** tab, select **System.ServiceModel**, and then click **OK**.
6. In the Solution Explorer, right-click **References**, click **Add Reference**, click the **.NET** tab, select **System.Configuration**, and then click **OK**.
7. Open the Class1.cs file and rename the class name from Class1 to **Validation**.
8. Add the following using statements to the top of the Class1.cs file.

```
using System.Configuration;
using System.ServiceModel;
using System.ServiceModel.Configuration;
using System.ServiceModel.Description;
using System.ServiceModel.Dispatcher;
```

Step 3 – Create a Class That Implements the Validation Logic

In this step, you create a new class, derived from **IParameterInspector**, to implement the validation logic.

The newly created class has the following characteristics:

- It implements **AfterCall()** and **BeforeCall()** methods.
- When used as part of the service, **BeforeCall()** will be invoked before the parameters are dispatched to the service operation. **AfterCall()** will be invoked after the service has processed the call and is returning a the response to the client. Use **BeforeCall()** to validate your input parameters and **AfterCall()** to validate your output parameters.
- When used as part of the client, **BeforeCall()** will be invoked before calling the service, and **AfterCall()** before the service's response is dispatched to the client code. Use **AfterCall()** to validate the response from the service, and **BeforeCall()** to validate the return from the service.

This example uses simple validation logic to check if the parameter passed to the operation is within the values 1 and 5. If the validation fails, an exception is thrown with the Validation Input Error message.

Perform the following steps:

1. Open the Class1.cs file and rename the class name from Class1 to **Validation**. Click Yes on the dialog box which pops up.
2. Add the following using statements to the top of the **Validation.cs** file.

```
using System.ServiceModel;
using System.ServiceModel.Dispatcher;
```

3. Add the following code to implement the **AfterCall** method in the **ValidationParameterInspector** class .

```
public class ValidationParameterInspector : IParameterInspector
 {
     public void AfterCall(string operationName, object[] outputs,
                     object returnValue, object correlationState)
     {

         if (operationName == "GetData")
         {
             for (int index = 0; index < outputs.Length; index++)
             {
                 if (index == 0)
                 {

                   // execute the method level validators
                  if (((int)outputs[index] < 0) ||
                                       ((int)outputs[index] > 5))
                   throw new FaultException("Your Error Message");
                 }

             }

         }
     }
 }
```

4. Add the following code to implement the **BeforeCall** method in the **ValidationParameterInspector** class.

```
public class ValidationParameterInspector : IParameterInspector
 {
     public void AfterCall(string operationName, object[] outputs,
                   object returnValue, object correlationState)
     {
     …

     }

     public object BeforeCall(string operationName, object[] inputs)
     {

         if (operationName == "GetData")
         {

             for (int index = 0; index < inputs.Length ; index++)
             {
                 if(index==0)
                 {
                 // execute the method level validators
                 if (((int)inputs[index]<0)||((int)inputs[index] > 5))
                  throw new FaultException("Validation Input  Error");
                 }

             }

         }

         return null;
     }
    }
```

Step 4 – Create a Class That Implements a Custom Endpoint Behavior

In this step, you create a new class, derived from **IEndpointBehavior**, that implements a custom endpoint behavior.

The newly created class has the following characteristics:

- It implements **ApplyClientBehavior()** to add the **ValidationParamaterInspector** to the client operation and enable client-side validation.
- It implements **ApplyDispatchBehavior()** toadd the **ValidationParameterInspector** to the dispatch operation and enable service-side validation.
- It verifies that it is enabled in the configuration before adding the **ValidationParameterInspector** to the client or dispatch run time.

Perform the following step:

- Copy the below code snippet and paste it into the Class1.cs file, inside the **Validation** class that already exists:

```
class ValidationBehavior : IEndpointBehavior
  {
      private bool enabled;
      #region IEndpointBehavior Members

      internal ValidationBehavior(bool enabled)
      {
          this.enabled = enabled;
      }

      public bool Enabled
      {
          get { return enabled; }
          set { enabled = value; }
      }

      public void AddBindingParameters(ServiceEndpoint serviceEndpoint,
System.ServiceModel.Channels.BindingParameterCollection bindingParameters)
      { }

      public void ApplyClientBehavior(
        ServiceEndpoint endpoint,
        ClientRuntime clientRuntime)
      {
          //If enable is not true in the config we do not apply the
Parameter Inspector
         if (false == this.enabled)
         {
              return;
         }

          foreach (ClientOperation clientOperation in
clientRuntime.Operations)
          {
              clientOperation.ParameterInspectors.Add(
                  new ValidationParameterInspector());
          }

      }

      public void ApplyDispatchBehavior(
         ServiceEndpoint endpoint,
         EndpointDispatcher endpointDispatcher)
      {
          //If enable is not true in the config we do not apply the
Parameter Inspector

          if (false == this.enabled)
          {
```

```
                return;
            }

            foreach (DispatchOperation dispatchOperation in
endpointDispatcher.DispatchRuntime.Operations)
            {

                dispatchOperation.ParameterInspectors.Add(
                    new ValidationParameterInspector());
            }

        }

        public void Validate(ServiceEndpoint serviceEndpoint)
        {

        }

        #endregion
    }
```

Step 5 – Create a Class That Implements a Custom Configuration Element

In this step, you create a new class, derived from **BehaviorExtensionElement**, that implements a custom configuration element.

The newly created class has the following characteristics:

- It implements **CreateBehavior()** to create an instance of the **ValidationBehavior** class.
- It implements **BehaviorType()** to return the **ValidationBehavior** type. This will allow the custom behavior to be exposed in the service or client configuration sections.
- It implements **ConfigurationProperty** to allow the behavior to be enabled or disabled in the WCF configuration files.

Perform the following step:

- Copy the below code snippet and paste it into the Class1.cs file, inside the **Validation** class that already exists:

```
  public class CustomBehaviorSection : BehaviorExtensionElement
  {

        private const string EnabledAttributeName = "enabled";

        [ConfigurationProperty(EnabledAttributeName, DefaultValue = true,
IsRequired = false)]
        public bool Enabled
        {
            get { return (bool)base[EnabledAttributeName]; }
            set { base[EnabledAttributeName] = value; }
        }
```

```
            protected override object CreateBehavior()
            {
                return new ValidationBehavior(this.Enabled);

            }

            public override Type BehaviorType
            {

                get { return typeof(ValidationBehavior); }


            }
    }
```

Step 6 – Add the Custom Behavior to the Configuration File

In this step, you add the custom behavior to the behavior element extension in the WCF configuration file so that it can be used by the WCF endpoint.

1. Compile your validation class library solution to create MyClassValidation.dll.
2. Return to the original instance of Visual Studio that contains your WCF service solution.
3. Right-click the **WCF Web site** project and then click **Add Reference**. Navigate to the folder containing MyClassValidation.dll and then click **Add**.
4. Right-click web.config and then click **Edit WCF configuration**.
   If you do not see the Edit WCF Configuration option, on the **Tools** menu, click **WCF Service Configuration Editor**. Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu.
5. Expand the **Advanced** node and the **Extensions** node, and then click **behavior element extensions**.
6. Click **New**.
7. In the **Name** field, type Validator
8. Select the **Type** field, click the button that appears to the right, navigate to the folder containing MyClassValidation.dll, and then double-click the dll file.
9. Double-click the type name **MyParamaterValidator. CustomBehaviorSection** and then click **OK**.
10. In the WCF Configuration Editor, on the **File** menu, click **Save**.
    Verify that your configuration file contains the following:

```
<system.serviceModel>
  ...
    <extensions>
      <behaviorExtensions>
        <add name="Validator" type="MyParamaterValidator.
CustomBehaviorSection, MyParamaterValidator, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null" />
      </behaviorExtensions>
   </extensions>
 ...
 <system.serviceModel>
```

Step 7 – Create an Endpoint Behavior and Map It to Use the Custom Behavior

In this step, you create an endpoint behavior and map it to the custom behavior created in Step 7.

1. In the WCF Configuration Editor, expand the **Advanced** node, right-click **Endpoint Behavior,** and then click **New Endpoint Behavior Configuration**.
2. Select the new behavior and then in the **Name** field, type MyEndPointBehavior
3. Click **Add**, select the Validator custom behavior, and then click **Add**.
4. In the WCF Configuration Editor, on the **File** menu, click **Save**.
   Verify that your configuration file contains the following:

```
<behaviors>
  ...
     <endpointBehaviors>
        <behavior name="MyEndPointBehavior">
          <Validator />
        </behavior>
      </endpointBehaviors>
  ...
 </behaviors>
```

Step 8 – Configure the Service Endpoint to Use the Endpoint Behavior

In this step, you configure the service to use the endpoint behavior in order to consume the custom validator.

1. In the WCF Configuration Editor, expand the **Services** node, then expand the **Service** node and then expand **Endpoints**.
2. Select the first **[Empty Name**] node.
3. In the **BehaviorConfiguration** field, select **MyEndPointBehavior**.
4. In the WCF Configuration Editor, on the file **File** menu, click **Save**.
   Verify that your configuration file contains the following:

```
<endpoint address="" behaviorConfiguration="MyEndPointBehavior"
          binding="wsHttpBinding" contract="IService">
          <identity>
            <dns value="localhost" />
          </identity>
</endpoint>
```

Step 9 - Test the Parameter Validator

In this step, you create a sample WCF client to test your validator.

1. Right-click your WCF service solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.
4. Right-click your client project and then click **Add Service Reference**.
5. In the **Add Service Reference** dialog box, set the **Address** field to **http://localhost/WCFTestParameterValidation/Service.svc** and then click **Go**.
6. Set the **Namespace** field to **WCFTestService** and then click **OK**.
7. Open the designer for your new Windows form.
8. Drag a button control into the designer.
9. Drag a textbox control into the designer.
10. Double-click the button to show the underlying code.
11. In the code behind the button, create an instance of the WCF service proxy, and then call the **DoWork()** method on your WCF service based on the value that is in the textbox control.
    When you call the service, your current user security context will automatically be passed to your WCF Service.

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                              WCFTestService.ServiceClient();
      MessageBox.Show(myService.GetData(int.Parse(textBox1.Text)));
      myService.Close();
}
```

12. Right click the client prOject and then click **Set as Startup Project**.
13. Run the client application by pressing F5 or Ctrl+F5, and then click the button. In the text box, enter a value and then click the button.
    The application will display the message “You entered: value**”** for the correct value (input between 1 and 5) or a validation error for the incorrect value.

Deployment Considerations

Do not divulge exception errors to clients in production. Instead, develop a fault contract and return it to your client inside the BeforeCall() and AfterCall() methods of the ValidationParameterInspector class. For client-side validation, follow the same steps detailed in this How To article, but instead use the app.config of the client consuming the service.

Additional Resources

- For additional information on configuring and extending the runtime with behaviors, see “Configuring and Extending the Runtime with Behaviors” at http://msdn2.microsoft.com/en-us/library/ms730137.aspx

- For additional information about the IEndPoint interface , see “IEndpointBehavior Interface” at http://msdn2.microsoft.com/en-us/library/system.servicemodel.description.iendpointbehavior.aspx
- For additional information about IParameterInspector interface, see “IParameterInspector Interface” at http://msdn2.microsoft.com/en-us/library/system.servicemodel.dispatcher.iparameterinspector.aspx
- For additional information about the BehaviorExtensionElement class, see “BehaviorExtensionElement Class” at http://msdn2.microsoft.com/en-us/library/system.servicemodel.configuration.behaviorextensionelement.aspx

How To – Perform Message Validation with Schema Validation in WCF

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft .NET Framework 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article shows you how to perform message validation using a schema in WCF. You will learn how to create a custom client message inspector and dispatcher message inspector that can be used to validate messages on both the server and the client.

Contents

Objectives

- Learn how to create a custom configuration element that will allow exposing the custom endpoint behavior in the configuration file.
- Learn how to create a custom endpoint behavior that will consume the client and dispatcher message inspectors.
- Learn how to create custom client and dispatcher message inspectors to validate messages using schemas.

Overview

Message validation represents one line of defense in the protection of your WCF application. With this approach, you validate messages using schemas to protect WCF service operations from attack by a malicious client. Validate all messages received by the client to protect the client from attack by a malicious service. Message validation makes it possible to validate messages when operations consume message contracts or data contracts, which cannot be done using parameter validation. Message validation allows you to create validation logic inside schemas, thereby providing more flexibility and reducing development time. Schemas can be reused across different applications inside the organization, creating standards for data representation. Additionally, message validation allows you to protect operations when they consume more complex data types involving contracts representing business logic.

To perform message validation, you first build a schema that represents the operations of your service and the data types consumed by those operations. You then create a .NET class that implements a custom client message inspector and custom dispatcher message inspector to validate the messages sent/received to/from the service. Next, you implement a custom endpoint behavior to enable message validation on both the client and the service. Finally, you implement a custom configuration element on the class that allows you to expose the extended custom endpoint behavior in the configuration file of the service or the client.

Summary of Steps

- Step 1 – Create a Sample WCF Service
- Step 2 – Configure the WCF Service to Use wsHttpBinding with Windows Authentication and Message Security
- Step 3 – Create the Schema to Validate the Message
- Step 4 – Create a Windows Class Library Project That Will Contain the Three Classes Necessary for Schema Validation
- Step 5 – Create a Class That Implements the Schema Validation Logic
- Step 6 – Create a Class That Implements a Custom Endpoint Behavior
- Step 7 – Create a Class That Implements a Custom Configuration Element
- Step 8 – Add the Custom Behavior to the Configuration File
- Step 9 – Create an Endpoint Behavior and Map It to Use the Custom Behavior
- Step 10 – Configure the Service Endpoint to Use the Endpoint Behavior
- Step 11 – Test the Schema Validator

Step 1 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio, hosted in an Internet Information Services (IIS) virtual directory.

1. In Visual Studio, on the menu, click **File** -> **New Web Site.**

2. In the **New Web Site** dialog box, in the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http**.
3. In the **New Web Site dialog** box, set the new Web site address to http://localhost/WCFTestSchemaValidation and then click **OK**.
4. Create a data contract with the class **CustomerData**. This data contract will be consumed by the operation of your service. To do this, in Visual Studio, double-click the **IService.cs** file, copy the following code snippet, and paste it into the bottom of the IService.cs file:

```
[DataContract(Namespace =
"http://Microsoft.PatternPractices.WCFGuide")]
public class CustomerData
{
    [DataMember]
    string text;
    [DataMember]
    int CustomerID;

    public string Text
    {
        get { return text; }
        set { text = value; }
    }

    public int customerid
    {
        get { return CustomerID; }
        set { CustomerID = value; }
    }
}
```

5. Change the definition of the **GetData** operation contract to use the CustomerData type. To do this, double-click **IService.cs**, go to the top of the file, and replace the previous definition with the new one as follows.

   Previous operation contract:

```
[OperationContract]
string GetData(int value);
```

   New operation contract:

```
[OperationContract]
CustomerData GetData(CustomerData CustomerInfo);
```

6. Change the implementation of the **GetData** operation contract to use the CustomerData type. Double-click I**Service.cs**, go to the top of the file, and replace the previous definition with the new one as follows.

   Previous operation contract implementation:

```
public string GetData(int value)
{
    return string.Format("You entered: {0}", value);
}
```

New operation contract implementation:

```
public CustomerData GetData(CustomerData CustomerInfo)
{
   CustomerData CustomerInfoResponse = new CustomerData();
   CustomerInfoResponse.Text = CustomerInfo.Text;
   CustomerInfoResponse.customerid = CustomerInfo.customerid+1;
   return CustomerInfoResponse;
}
```

7. Add the namespace definition to conform to the schema you are going to create. Double-click **IService.cs** and add the string (Namespace = "http://Microsoft.PatternPractices.WCFGuide") after the ServiceContract entry in that file. Your service contract entry will look like the following code fragment.

```
[ServiceContract(Namespace =
"http://Microsoft.PatternPractices.WCFGuide")]
```

Step 2 – Configure the WCF Service to Use wsHttpBinding with Windows Authentication and Message Security

By default, your WCF service will be configured to use **wsHttpBinding** with message security and Windows authentication. Verify that your web.config configuration file looks as follows:

```
…
<services>
  <service name="Service" behaviorConfiguration="ServiceBehavior">
    <!-- Service Endpoints -->
    <endpoint address="" binding="wsHttpBinding" contract="IService">
      <identity>
        <dns value="localhost"/>
      </identity>
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange"/>
  </service>
</services>
…
```

Step 3 – Create the Schema to Validate the Message

In this step, you create the schema to validate the message.

1. Right-click the http://localhost/WCFTestSchemaValidation project and then click **Add New Item**.

2. Select **Xml Schema** from the Visual Studio installed templates and enter the name **SchemaValidation** in the **Name** text box. Select **Visual C#** as the language and then click **Add**.
3. In the Visual Studio editor, delete the entire contents of this file. Copy the following schema definition and paste it into the file:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
targetNamespace="http://Microsoft.PatternPractices.WCFGuide"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://Microsoft.PatternPractices.WCFGuide">
  <xs:element name="GetData">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="CustomerInfo" nillable="false"
type="tns:CustomerData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="CustomerData">
    <xs:sequence>
      <xs:element name="CustomerID" type="tns:CustIDLimiter">
      </xs:element>
      <xs:element name="text" type="tns:CustomerN">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="CustomerN">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
      <xs:maxLength value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CustIDLimiter">
    <xs:restriction base="xs:int">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="GetDataResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="GetDataResult" nillable="false"
type="tns:CustomerData" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The schema first defines the **GetData** operation contract that takes the **CustomerData** type as a parameter being instantiated by **CustomerInfo** being passed to the operation call. **CustomerData** must exist and cannot be null.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
targetNamespace="http://Microsoft.PatternPractices.WCFGuide"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://Microsoft.PatternPractices.WCFGuide">
  <xs:element name="GetData">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="CustomerInfo" nillable="false"
type="tns:CustomerData"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

CustomerData is a complex type that has **CustomerID** and **text** as data members. **CustomerID** and **text** are of type **CustIDLimiter** and **CustomerN**.

```
  <xs:complexType name="CustomerData">
    <xs:sequence>
      <xs:element name="CustomerID" type="tns:CustIDLimiter">
      </xs:element>
      <xs:element name="text" type="tns:CustomerN">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

CustomerN and **CustIDLimiter** are just schema facets: they are simple types – string and integer – that serve to limit the length and value of the string and integer. In the fragment below, string is limited to 1 and 5 characters in length and integer is limited to range values of 1 and 5.

```
  <xs:simpleType name="CustomerN">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
      <xs:maxLength value="5" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CustIDLimiter">
    <xs:restriction base="xs:int">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="5" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

The **GetDataResponse** element is a response from **GetData**. It returns a CustomerInfoResponse with type CustomerData, so the response will be also tested.

```
  <xs:element name="GetDataResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" name="GetDataResult" nillable="false"
type="tns:CustomerData" />
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Step 4 – Create a Windows Class Library Project That Will Contain the Three Classes Necessary for Schema Validation

In this step, you create a Microsoft Windows® class library project that will include three classes for the schema validation:

- One class to implement the schema validation logic
- A second class to implement the endpoint behavior that will use the schema validation class
- A third class to implement a behavior extension so that the validator will be visible in the service and client configuration files

Perform the following steps to create a Windows class library project with these three classes:

1. Open a new instance of Visual Studio, leaving your WCF service solution open.
2. In the new instance of Visual Studio, click **File**, click **New**, and then click **Project**.
3. Expand **Visual C#,** click **Windows**, and then select **Class Library**.
4. In the **Name** field, type **MySchemaValidationClass** and then click **OK**.
5. In the Solution Explorer, right-click **References and then** click **Add Reference**., Click the **.NET** tab, click **System.ServiceModel**, and then click **OK**.
6. In the Solution Explorer, right-click **References** and then click **Add Reference**. Click the **.NET** tab, click **System.Configuration**, and then click **OK**.
7. Open the Class1.cs file and rename the class name from Class1 to **Validation**.
8. Add the following **using** statements to the top of the Class1.cs file:

```
using System.Configuration;
using System.ServiceModel;
using System.ServiceModel.Configuration;
using System.ServiceModel.Description;
using System.ServiceModel.Dispatcher;
```

Step 5 – Create a Class That Implements the Schema Validation Logic

In this step, you create a new class, derived from interfaces **IClientMessageInspector** and **IDispatchMessageInspector**, to implement the schema validation logic for both the client and the dispatcher.

The newly created class implements the **AfterReceiveRequest()**, **BeforeSendReply()**, **BeforeSendRequest()**, and **AfterReceiveReply()** methods and has the following characteristics:.

- **On the dispatcher: AfterReceiveRequest** will happen when inbound messages are received by the dispatcher, before the operation is invoked and deserialization of messages has occurred. If the message is encrypted, decryption will take place first. **BeforeSendReply** will happen when outbound messages are to be sent back to the client. It will happen after the operation is invoked, and after serialization has occurred. If the message is encrypted, encryption will not take place.
- **On the client: BeforeSendRequest** will happen when outbound messages are sent by the client, after serialization has occurred. If the message is encrypted, encryption will not take place.
  AfterReceiveReply will happen when inbound messages are received by the client before deserialization of the message has occurred. If the message is encrypted, decryption will take place first.
- **ValidateMessage. ValidateMessage** will validate the message with regard to the schema definition. If validation succeeds, a new message is constructed and returned to the caller. On the dispatcher side, either a new message is returned before the operation is invoked or before a response is sent to the client. On the client side, ValidateMessage is called before sending the message to the service or before returning to the application.

In the following example, a simple schema validation logic is implemented by simply traversing the **XmlReader**. If validation fails, a fault exception or a message is returned to the client.

Copy and paste the following code snippet to the class1.cs file:

```
  public class SchemaValidation
    {
       public class SchemaValidationMessageInspector :
IClientMessageInspector,IDispatchMessageInspector
        {
            XmlSchemaSet schemas;
            public SchemaValidationMessageInspector(XmlSchemaSet schemas)
            {
                this.schemas = schemas;
            }
            void validateMessage(ref System.ServiceModel.Channels.Message
message)
            {
                XmlDocument bodyDoc = new XmlDocument();
                bodyDoc.Load(message.GetReaderAtBodyContents());
                XmlReaderSettings settings = new XmlReaderSettings();
                settings.Schemas.Add(schemas);
                settings.ValidationType = ValidationType.Schema;
                XmlReader r = XmlReader.Create(new XmlNodeReader(bodyDoc),
settings);
                while (r.Read()) ; // do nothing, just validate
                // Create new message
                Message newMsg = Message.CreateMessage(message.Version, null,
                    new XmlNodeReader(bodyDoc.DocumentElement));
                newMsg.Headers.CopyHeadersFrom(message);
```

```
                foreach (string propertyKey in message.Properties.Keys)
                    newMsg.Properties.Add(propertyKey,
message.Properties[propertyKey]);
                // Close the original message and return new message
                message.Close();
                message = newMsg;
            }

            object IDispatchMessageInspector.AfterReceiveRequest(ref
System.ServiceModel.Channels.Message request,
System.ServiceModel.IClientChannel channel,
System.ServiceModel.InstanceContext instanceContext)
            {
                try{
                    validateMessage(ref request);
                }
            catch (FaultException e)
            {
                throw new FaultException<string>(e.Message);
            }
            return null;

            }

            void IDispatchMessageInspector.BeforeSendReply(ref
System.ServiceModel.Channels.Message reply, object correlationState)
            {
                try
                {
                    validateMessage(ref reply);
                }
                catch (FaultException fault)
                {
                    // if a validation error occurred, the message is
replaced
                    // with the validation fault.
                    reply = Message.CreateMessage(reply.Version, new
FaultException("validation error in reply message").CreateMessageFault() ,
reply.Headers.Action);
                }

            }

            void IClientMessageInspector.AfterReceiveReply(ref
System.ServiceModel.Channels.Message reply, object correlationState)
            {
                validateMessage(ref reply);
            }

            object IClientMessageInspector.BeforeSendRequest(ref
System.ServiceModel.Channels.Message request,
System.ServiceModel.IClientChannel channel)
            {
                validateMessage(ref request);
                return null;
            }
        }
```

```
    }
```

Step 6 – Create a Class That Implements a Custom Endpoint Behavior

In this step, you create a new class, derived from **IEndpointBehavior**, that implements a custom endpoint behavior.

The newly created class has the following characteristics:

1. It implements **ApplyClientBehavior()** to add the **SchemaValidationMessageInspector** to the client operation and enable client-side validation.
2. It implements **ApplyDispatchBehavior()** to add the **SchemaValidationMessageInspector** to the dispatch operation and enable service-side validation.
3. It verifies that it is enabled in the configuration before adding the **SchemaValidationMessageInspector** to the client or dispatch run time.

Copy and paste the following code snippet to the Class1.cs file, inside the **Validation** class that already exists:

```
  class SchemaValidationBehavior : IEndpointBehavior
        {
            private bool enabled;
            private XmlSchemaSet schemaSet;


            internal SchemaValidationBehavior(bool enabled,XmlSchemaSet
schemaSet)
            {
                this.enabled = enabled;
                this.schemaSet = schemaSet;
            }

            public bool Enabled
            {
                get { return enabled; }
                set { enabled = value; }
            }

            public void AddBindingParameters(ServiceEndpoint serviceEndpoint,
System.ServiceModel.Channels.BindingParameterCollection bindingParameters)
            { }

            public void ApplyClientBehavior(
              ServiceEndpoint endpoint,
              ClientRuntime clientRuntime)
            {
                //If enable is not true in the config we do not apply the
Parameter Inspector
                if (false == this.enabled)
                {
                    return;
```

```
                }
                SchemaValidationMessageInspector inspector = new
SchemaValidationMessageInspector(schemaSet);
                clientRuntime.MessageInspectors.Add(inspector);
            }

            public void ApplyDispatchBehavior(
               ServiceEndpoint endpoint,
               EndpointDispatcher endpointDispatcher)
            {
                //If enable is not true in the config we do not apply the
Parameter Inspector
                if (false == this.enabled)
                {
                    return;
                }
                SchemaValidationMessageInspector inspector = new
SchemaValidationMessageInspector(schemaSet);

endpointDispatcher.DispatchRuntime.MessageInspectors.Add(inspector);
            }

            public void Validate(ServiceEndpoint serviceEndpoint)
            {

            }


        }
```

Step 7 – Create a Class That Implements a Custom Configuration Element

In this step, you create a new class, derived from **BehaviorExtensionElement**, that implements a custom configuration element.

The newly created class has the following characteristics:

1. It implements **CreateBehavior()** to create an instance of the **SchemaValidationBehavior** class. Inside the method, a schema set is initialized with the schema that reads the base directory of the application loading the SchemaValidation.xsd file created previously. The schema set is passed to the **Schemavalidation** behavior that will pass it to the schema inspector.
2. It implements **BehaviorType()** to return the **SchemaValidationBehavior** type. This will allow the custom behavior to be exposed in the service or client configuration sections.
3. It implements **ConfigurationProperty** to allow the behavior to be enabled or disabled in the WCF configuration files.

Copy and paste the following code snippet to the Class1.cs file, inside the **Validation** class that already exists:

```
      public class CustomBehaviorSection : BehaviorExtensionElement
      {

            private const string EnabledAttributeName = "enabled";

            [ConfigurationProperty(EnabledAttributeName, DefaultValue = true,
IsRequired = false)]
            public bool Enabled
            {
                get { return (bool)base[EnabledAttributeName]; }
                set { base[EnabledAttributeName] = value; }
            }

            protected override object CreateBehavior()
            {
               XmlSchemaSet schemaSet = new XmlSchemaSet();
               Uri baseSchema = new
Uri(AppDomain.CurrentDomain.BaseDirectory);
               string mySchema = new
Uri(baseSchema,"SchemaValidation.xsd").ToString();
               XmlSchema schema = XmlSchema.Read(new XmlTextReader(mySchema),
null);
               schemaSet.Add(schema);
               return new SchemaValidationBehavior(this.Enabled,schemaSet);

            }

            public override Type BehaviorType
            {

                get { return typeof(SchemaValidationBehavior); }


            }
        }
```

Step 8 – Add the Custom Behavior to the Configuration File

In this step, you add the custom behavior to the behavior element extension in the WCF configuration file so that it can be used by the WCF endpoint.

1. Compile your schema validation class library solution to create MySchemaClassValidation.dll.
2. Return to the original instance of Visual Studio that contains your WCF service solution.
3. Right-click the WCF Web site project and then click **Add Reference**. Navigate to the folder containing the MySchemaClassValidation.dll and click **Add**.
4. Right-click **web.config** and then click **Edit WCF configuration**.
   If you do not see the Edit WCF Configuration option, on the **Tools** menu, click **WCF Service Configuration Editor**. Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu.
5. Expand the **Services** node and the **Extensions** node and then click **Behavior Element Extensions**.

6. Click **New**.
7. In the **Name** field, type **SchemaValidator**
8. Select the **Type** field, click the button that appears to the right, navigate to the folder containing MySchemaClassValidation.dll,and then double-click the .dll file.
9. Double-click the type name **MySchemaValidationClass.SchemaValidation+CustomBehaviorSection** and then click **OK**.
10. In the WCF Configuration Editor, click **File** and then click **Save**.

Verify that your configuration file contains the following:

```
<system.serviceModel>
  ...
<extensions>
      <behaviorExtensions>
        <add name="SchemaValidator"
type="MySchemaValidationClass.SchemaValidation+CustomBehaviorSection,
MySchemaValidationClass, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null" />
      </behaviorExtensions>
    </extensions>
 ...
 <system.serviceModel>
```

Step 9 – Create an Endpoint Behavior and Map It to Use the Custom Behavior

In this step, you create an endpoint behavior and map it to the custom behavior created in Step 7.

1. In the WCF Configuration Editor, expand the **Advanced** node, right-click **Endpoint Behavior,** and then click **New Endpoint Behavior Configuration.**
2. Select the new behavior and then in the **Name** field, type **MyEndPointBehavior**
3. Click **Add**, select the SchemaValidator custom behavior, and then click **Add** again.
4. In the WCF Configuration Editor, click **File** and then click **Save**.

Verify that your configuration file contains the following:

```
<behaviors>
  ...
<endpointBehaviors>
        <behavior name="MyEndPointBehavior">
          <SchemaValidator />
        </behavior>
      </endpointBehaviors>
 ...
 </behaviors>
```

Step 10 – Configure the Service Endpoint to Use the Endpoint Behavior

In this step, you configure the service to use the endpoint behavior to consume the custom validator.

1. In the WCF Configuration Editor, expand the **Service** node and then expand **Endpoints**.
2. Select the first **[Empty Name]** node.
3. In the **BehaviorConfiguration** field, select **MyEndPointBehavior**.
4. In the WCF Configuration Editor, click **File** and then click **Save**.

Verify that your configuration file contains the following:

```
<endpoint address="" behaviorConfiguration="MyEndPointBehavior"
          binding="wsHttpBinding" contract="IService">
          <identity>
            <dns value="localhost" />
          </identity>
</endpoint>
```

Step 11 - Test the Schema Validator

In this step, you create a sample WCF client to test your validator.

1. Right-click your WCF service solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section. select **Windows Forms Application** .
3. In the **Name** field, type **Test Client** and then click **OK**.
4. Right-click your client project and then click **Add Service Reference**.
5. In the **Add Service Reference** dialog box, set the **Address** field to http://localhost/WCFTestSchemaValidation and then click **Go**.
6. Set the **Namespace** field to WCFTestService and then click **OK**.
7. Open the designer for your new Windows form.
8. Drag three text box controls into the designer.
9. Drag a button control into the designer.
10. Double-click the button to show the underlying code.
11. In the code behind the button, create an instance of the WCF service proxy, and then call the **GetData()** method on your WCF service as follows:

```
      try
      {
            WCFTestService.CustomerData CustomerInfo = new
WindowsFormsApplication1. WCFTestService.CustomerData();
            CustomerInfo.text = textBox1.Text;
            CustomerInfo.CustomerID = int.Parse(textBox2.Text);
```

```
            WCFTestService.ServiceClient proxy = new
WindowsFormsApplication1. WCFTestService.ServiceClient();
            WCFTestService.CustomerData CustomerInfoResponse =
proxy.GetData(CustomerInfo);
            textBox3.Text = CustomerInfoResponse.text;
            proxy.Close();
      }

      catch (FaultException ex)
      {
           textBox3.Text = ex.Message;
      }
```

12. Right-click the client project and then click **Set as Startup Project**.
13. Run the client application by pressing F5 or Ctrl+F5 and then click the button.
14. In **textbox1** enter a string with a maximum size of 5 characters.
15. In **textbox2** enter an integer value between 1 and 4.
16. Next try entering values outside of these validation ranges and you'll see the application displays a validation error.

Deployment Considerations

Consider the following key points before deployment:

- Do not divulge exception errors to clients in production. Instead, develop a fault contract and return it to your client inside **AfterReceiveRequest()**
- Do not divulge exception errors after **BeforeSendReply()**. Instead, develop a fault contract and build an error message with the fault contract and return it to the client.
- For client-side validation, follow the same steps detailed in this How To article, but instead use the app.config file of the client consuming the service.
- Consider caching the schema for performance benefits.
- Consider the more advanced schema validation logic provided in the download sample at http://msdn2.microsoft.com/en-us/library/aa717047.aspx

Additional Resources

- For additional information on configuring and extending the run time with behaviors, see "Configuring and Extending the Runtime with Behaviors" at http://msdn2.microsoft.com/en-us/library/ms730137.aspx
- For additional information about the **IEndPointBehavior** interface, see "IEndpointBehavior Interface" at
- For additional information about the **IDispatchMessageInspector** interface, see "IDispatchMessageInspector Interface" at http://msdn2.microsoft.com/en-us/library/system.servicemodel.dispatcher.idispatchmessageinspector.aspx

- For additional information about the **IClientMessageInspector** interface, see "IClientMessageInspector Interface" at http://msdn2.microsoft.com/en-us/library/system.servicemodel.dispatcher.iclientmessageinspector.aspx
- For additional information about the **BehaviorExtensionElement** class, see "BehaviorExtensionElement Class" at http://msdn2.microsoft.com/en-us/library/system.servicemodel.configuration.behaviorextensionelement.aspx
- For more information on message inspectors, see "Message Inspectors" at http://msdn2.microsoft.com/en-us/library/aa717047.aspx

How To: Use basicHttpBinding with Windows Authentication and TransportCredentialOnly in WCF from Windows Forms

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of using Windows Authentication over **basicHttpBinding** binding using **TransportCredentialsOnly** security mode. This article shows you how to configure WCF, configure Internet Information Services (IIS) for Windows Authentication, and test the service with a sample WCF client.

Contents

Objectives

- Create a WCF service hosted in IIS.
- Expose the WCF service as a legacy Web service through basicHttpBinding.
- Call the service from a test client.

Overview

Windows Authentication is suited for scenarios in which your users have domain credentials. In the scenario described in this How To article, users are authenticated by using Windows Authentication. The **basicHttpBinding** binding is used in order to provide support for older clients that expect a legacy ASMX Web service. The **TransportCredentialOnly** security mode option passes the user credentials without encrypting or signing the messages. Use this mode with caution as it will not protect the credentials being

transmitted and they will have to be protected by some other means, such as Internet Protocol Security (IPSec).

In this How To article, you will create a sample WCF service in Visual Studio 2008. You will then configure the service to use **basicHttpBinding** with **TransportCredentialOnly** security through the use of the WCF Configuration Editor. You will enable Windows Authentication in IIS to allow your users to authenticate to the service. Finally, you will create a test client to verify that the service is working properly.

Summary of Steps

- Step 1 – Create a Sample WCF Service
- Step 2 – Configure the WCF Service to Use basicHttpBinding
- Step 3 – Configure the basicHttpBinding to use Windows Authentication with TransportCredentialOnly
- Step 4 – Enable Windows Authentication on IIS
- Step 5 – Create a Windows Forms Test Client Application
- Step 6 – Add a WCF Service Reference to the Client
- Step 7 – Test the Client and WCF Service

Step 1 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio, hosted in an IIS virtual directory.

1. In Visual Studio, on the **File** menu, click **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http,** and specify **http://localhost/WCFServiceBasicHttp** as the **Path.** Click **OK** in the **New Web Site** dialog box to create a virtual directory and a sample WCF service.
3. Browse to your WCF service at http://localhost/WCFServiceBasicHttp/Service.svc.
   You should see your WCF service respond with details of the service.

Step 2 – Configure the WCF Service to Use basicHttpBinding

In this step, you configure your WCF service endpoint to use **basicHttpBinding**.

1. Right-click the Web.config file of the WCF service and then click **Edit WCF Configuration**.
   If you do not see the Edit WCF Configuration option, on the **Tools** menu, click **WCF Service Configuration Editor.** Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu.
2. In the Configuration Editor, in the Configuration section, expand **Service** and then expand **Endpoints**.
3. Select the first node **[Empty Name].** Set the name attribute to **BasicHttpEndpoint**.
   By default, the name will be empty because it is an optional attribute.
4. In the **Service Endpoint** section, set the **Binding** attribute to **basicHttpBinding** by choosing this option from the drop-down list.
5. In the **Configuration Editor**, on the **File** menu, click **Save**.

6. In Visual Studio, verify your configuration settings in Web.config. The configuration should look as follows:

```
…
<services>
  <service behaviorConfiguration="ServiceBehavior" name="Service">
    <endpoint address="" binding="basicHttpBinding"
        name="BasicHttpEndpoint"
        bindingConfiguration=""
        contract="IService">
      <identity>
        <dns value="localhost" />
      </identity>
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
       contract="IMetadataExchange" />
  </service>
</services> …
```

Step 3 – Configure basicHttpBinding to use Windows Authentication with TransportCredentialOnly

By default, the **basicHttpBinding** security mode is None. This default setting means that you do not have authentication and that neither transport nor message security is enabled. By enabling Windows Authentication with **TransportCredentialOnly**, you will get authentication, but no message protection; this is similar to how an ASMX Web service works.

1. In the Configuration Editor, in the **Configuration** section, select the Bindings folder.
2. In the **Bindings** section, choose **New Binding Configuration.**
3. In the **Create a New Binding** dialog box, select **basicHttpBinding**.
4. Click **OK**.
5. Set the **Name** of the binding configuration to some logical and recognizable name; for example, **BasicHttpEndpointBinding**.
6. Click the **Security** tab.
7. Set the **Mode** attribute to **TransportCredentialOnly** by choosing this option from the drop-down menu.
8. Set the **TransportClientCredentialType** to **Windows** by choosing this option from the drop-down list.
   In this case, the **Windows** option represents Kerberos.
9. In the Configuration section, select **BasicHttpEndpoint**.
10. Set the **BindingConfiguration** attribute to **BasicHttpEndpointBinding** by choosing this option from the drop-down list.
    This associates the binding configuration setting with the binding.
11. In the **Configuration Editor**, on the **File** menu, click **Save**.
12. In Visual Studio, verify your configuration, which should look as follows:

```
...
<bindings>
```

```
    <basicHttpBinding>
      <binding name="BasicHttpEndpointBinding">
        <security mode="TransportCredentialOnly">
          <transport clientCredentialType="Windows" />
        </security>
      </binding>
    </basicHttpBinding>
  </bindings>
  <services>
    <service behaviorConfiguration="ServiceBehavior" name="Service">
      <endpoint address="" binding="basicHttpBinding"
        bindingConfiguration="BasicHttpEndpointBinding"
        name="BasicHttpEndpoint" contract="IService">
        <identity>
          <dns value="localhost" />
        </identity>
      </endpoint>
      <endpoint address="mex" binding="mexHttpBinding"
          contract="IMetadataExchange" />
    </service>
  </services>
  ...
```

Step 4 – Enable Windows Authentication on IIS

In this step, you enable IIS for Windows Authentication to match the authentication scheme used in your WCF service.

1. Open Internet Information Services (IIS) Manager by running the **inetmgr** command from the command line.
2. Browse to the WCF Service virtual directory created in Step 1.
3. Right-click the virtual directory and then click **Properties**.
4. In the **Properties** dialog box, click the **Directory Security** tab.
5. In the **Authentication and access control** section, click **Edit**.
6. In the **Authentication Methods** dialog box, clear the **Enable anonymous access** check box, and then select the **Integrated Windows authentication** check box.
7. In the **Authentication Methods** dialog box, click **OK** button.
8. In the **Properties** dialog box, click **Apply** and then click **OK**.
9. Run the **iisreset** command from the command line.
10. Verify that your service is working correctly. In IIS Manager, browse to your service (Service.svc).

Important: Make sure that you have installed ASP.NET on your machine; if not or if in doubt, run the following command:

```
> c:\Widows\Microsoft.NET\Framework\vX.X.XXXXX\aspnet_regiis.exe /i
```

Step 5 – Create a Windows Forms Test Client Application

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Application**.
3. In the **Name** field, type **Test Client** and then click **OK** to create a Windows Forms application.

Step 6 – Add a WCF Service Reference to the Client

In this step, you add a Web reference of the WCF service to your Client application. This How To article uses a Web reference to show the usage of a WCF service as a legacy Web service; otherwise, you can add it as a service reference.

1. Right-click your Client project and then click **Add Service References**.
2. Click **Advanced** and then click **Add Web Reference** under the Compatibility section.
3. In the **Add Web References** dialog box, set the URL to your WCF Service - http://localhost/WCFServiceBasicHttp/Service.svc
4. Click **Go**.
5. In the **Web reference name:** field, change **localhost** to **WCFTestService**.
6. Click **Add Reference**.
   A Web reference to **WCFTestService** should now appear in your Client project.

Step 7 – Test the Client and WCF Service

In this step, you access the WCF service as a legacy ASMX Web service and make sure that it works.

1. In your Client project, drag a button control onto your Form.
2. Double-click the button control to show the underlying code.
3. In the code behind the button click, create an instance of the proxy, pass the default user credentials, and call MyOperation1 of your WCF Service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.Service myService = new
                              WCFTestService.Service();
      myService.Credentials =
                        System.Net.CredentialCache.DefaultCredentials;
      MessageBox.Show(myService.GetData(123, true));
      myService.Dispose();
}
```

4. Right-click the Client project and then click **Set as Startup Project**.
5. Run the Client application ny pressing F5 or Ctrl+F5. When you click the button on the form, it the message “You entered: 123” should appear.

Additional Resources

- For more information on using the **DefaultCredentials** property, see “How To: Pass Current Credentials to an ASP.NET Web Service” at http://support.microsoft.com/kb/813834

How To – Use Certificate Authentication and Message Security in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of using client certificates and message security to authenticate your users. The article shows you how to create and install client and service certificates during development, configure the WCF service and client to use the respective certificates, and test the service with a sample WCF client.

Contents

Objectives

- Learn how to create and use a temporary certificate for authentication and message security.
- Learn where to store the temporary certificate.
- Learn how to troubleshoot common errors related to temporary certificates, authentication, and message security in WCF.

Overview

When developing a WCF service that uses X.509 certificates to provide client authentication and message security, it is necessary to work with temporary certificates.

This is because production certificates are expensive and may not be readily available. There are two options for specifying trust on a certificate:

- **Peer trust** – Validates the certificate directly.
- **Chain trust** – Validates the certificate against the issuer of a certificate known as a root authority.

This How To article discusses the chain trust option because it is the most commonly used approach in Business-to-Business (B2B) scenarios.

To use chain trust validation during development time, you create a self-signed root certificate authority (CA) and place it in the Trusted Root Certification Authority store of the client and service machines. The certificate used by the WCF client for client authentication and the WCF service for service authentication and message protection is then created and signed by the root self-signed certificate and installed in the LocalMachine store.

You will use makecert.exe to create a certificate to act as your root CA. You will then use your root CA certificate to sign additional certificates for your WCF service and client. Finally, you will configure the WCF client and service to use your temporary certificate.

Summary of Steps

- **Step 1 – Create a Sample WCF Service**
- **Step 2 – Configure wsHttpBinding with Certificate Authentication and Message Security**
- **Step 3 – Create and Install a Service Certificate**
- **Step 4 – Configure the Service Certificate for the WCF Service**
- **Step 5 – Create a Test Client**
- **Step 6 – Add a WCF Service Reference to the Client**
- **Step 7 – Create and Install the Client Certificate for Authentication**
- **Step 8 – Configure the Client Certificate in the WCF Client Application**
- **Step 9 – Test the Client and WCF Service**

Step 1 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio.

1. In Visual Studio, from the **File** menu, click **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to Http and specify the virtual directory to be created in the **Path (**e.g., http://localhost/WCFTestService).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample WCF service.

4. Browse to your WCF service (i.e., http://localhost/WCFTestService/Service.svc). You should see details of your WCF service.

Step 2 – Configure wsHttpBinding with Certificate Authentication and Message Security

In this step, you configure the WCF service to use certificate authentication and message security.

1. Right-click the Web.config file of the WCF service and then choose the **Edit WCF Configuration** option.
2. In the Configuration Editor, in the **Configuration** section, expand **Service** and then expand **Endpoints**.
3. Select the first node **[Empty Name] and** Set the **Name** attribute to **wsHttpEndpoint**.
   By default, the name will be empty because it is an optional attribute.
4. Click the **Identity** tab and then delete the **Dns** attribute value.
5. In the Configuration Editor, select the Bindings folder.
6. In the **Bindings** section, choose **New Binding Configuration**.
7. In the **Create a New Binding** dialog box, select **wsHttpBinding**.
8. Click **OK**.
9. Set the **Name** of the binding configuration to some logical and recognizable name; for example, **wsHttpEndpointBinding**.
10. Click the **Security** tab.
11. Make sure that the **Mode** attribute is set to **Message**, which is the default setting.
12. Set the **MessageClientCredentialType** to **Certificate** by selecting this option from the drop-down list.
13. In the **Configuration** section, select the **wsHttpEndpoint** node.
14. Set the **BindingConfiguration** attribute to **wsHttpEndpointBinding** by selecting this option from the drop-down list.
    This associates the binding configuration setting with the binding.
15. In the Configuration Editor, on the **File** menu, select **Save**.
16. In Visual Studio, open your configuration and comment out the identity element. It should look as follows:

```
<!--<identity>
  <dns value="" />
</identity>-->
```

17. In Visual Studio, verify your configuration. The configuration should look as follows:

```
…
<bindings>
  <wsHttpBinding>
```

```
    <binding name="wsHttpEndpointBinding">
      <security>
        <message clientCredentialType="Certificate" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
<services>
  <service behaviorConfiguration="ServiceBehavior" name="Service">
    <endpoint address="" binding="wsHttpBinding"
      bindingConfiguration="wsHttpEndpointBinding"
      name="wsHttpEndpoint" contract="IService">
      <!--<identity>
        <dns value="" />
      </identity>-->
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />
  </service>
</services>
…
```

Step 3 – Create and Install a Service Certificate

In this step, you create a temporary service certificate and install it in the local store. This certificate will be used for service authentication and to encrypt the message, thereby protecting any other sensitive data.

Creating and installing the certificate is outside the scope of this How To article. For detailed steps on how to do this, see "How To - Create and Install Temporary Certificates in WCF for Message Security During Development."

Note:

- If you are running on Microsoft Windows® XP, give the certificate permissions for the ASPNET identity instead of the NT Authority\Network Service identity because the Internet Information Services (IIS) process runs under the ASPNET account.
- The temporary certificate should be used for development and testing purposes only. For actual production deployment, you will need to obtain a valid certificate from a certificate authority (CA).

Step 4 – Configure the Service Certificate for the WCF Service

In this step, you configure the WCF service to use the temporary certificate you created in the previous step.

1. In the Configuration Editor, expand the **Advanced** node, and then expand the **Service Behaviors** node.
2. Click **Add**.

3. In the **Service Behavior Element Extensions dialog** box, select the **serviceCredentials** option and then click **Add**.
4. Expand the **serviceCredentials** node and then select the **serviceCertificate** node.
5. Set the **FindValue** attribute to the name of the service certificate that you have created; for example, "CN=tempCertServer".
6. Leave the default settings for StoreLocation and StoreName.
7. In the Configuration Editor, on the **File** menu, click **Save**.
8. In Visual Studio, verify your configuration. The configuration should look as follows.

```
...
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceCredentials>
        <serviceCertificate findValue="CN=tempCertServer" />
      </serviceCredentials>
    </behavior>
  </serviceBehaviors>
</behaviors>
...
```

Step 5 – Create a Test Client

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.

Step 6 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service.

1. Right-click your Client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF Service (e.g., http://localhost/WCFTestService/Service.svc) and then click **Go**.
3. In the **Web reference name** field, change ServiceReference1 to **WCFTestService**.
4. Click **Add Reference**.
   A reference to WCFTestService should appear beneath Web References in your Client project.

Step 7 – Create and Install the Client Certificate for Authentication

In this step, you create a temporary client certificate by using the Root CA created in Step 3 above, and install it in the local store. This certificate will be used for client authentication and to encrypt the message, thereby protecting any other sensitive data.

1. Copy the root CA certificate (RootCATest.cer) and private key file (RootCATest.pvk), created as part of Step 3, to the client machine.
2. Open a Visual Studio command prompt and browse to the location where you copied the root CA certificate and private key file.
3. Run following command for creating a certificate signed by the root CA certificate:

```
makecert -sk MyKeyName -iv RootCATest.pvk -n "CN=tempCert" -ic
RootCATest.cer -sr CurrentUser -ss my -sky signature -pe tempCert.cer
```

4. In the **Enter Private Key Password** dialog box, enter the password for the root CA private key file created as part of the Step 3 above, and then click **OK**.

For more information and detailed steps, see “How To - Create and Install Temporary Certificates in WCF for Message Security During Development.”

Step 8 – Configure the Client Certificate in the WCF Client Application

In this step, you configure the WCF client to use the temporary certificate you created in the previous step.

1. In your test client, right-click the App.config file and then click **Edit WCF Configuration**.
2. In the Configuration Editor, expand the **Advanced** node, select **Endpoint Behaviors**, and then select **New Endpoint Behavior Configuration**.
3. Click **Add**.
4. In the **Adding Behavior Element Extension Sections** dialog box, select **clientCredentials** and then click **Add**.
5. Expand the **clientCredentials** node, Select the **clientCertificate** node, and then set the **FindValue** attribute to the subject name of the client certificate that you created and installed in Step 7; for example, "CN=tempCertClient".
6. Leave the default **StoreLocation** attribute set to **CurrentUser** as is.
7. In the Configuration Editor, expand the **Client** node, expand the **Endpoints** node, and then select the **WsHttpEndpoint** node.

8. Set the **BehaviorConfiguration** attribute to **NewBehavior** by choosing this option from the drop-down list.
   This is the endpoint behavior you just created.
9. In the Configuration Editor, on the **File** menu, click **Save**.
10. In Visual Studio, verify your configuration. The configuration should look as follows.

```
<system.serviceModel>
    <behaviors>
        <endpointBehaviors>
            <behavior name="NewBehavior">
                <clientCredentials>
                    <clientCertificate findValue="CN=tempCertClient"/>
                </clientCredentials>
            </behavior>
        </endpointBehaviors>
    </behaviors>
    ...
    <client>
        <endpoint address="http://<<service address>>"
            behaviorConfiguration="NewBehavior" binding="wsHttpBinding"
            bindingConfiguration="wsHttpEnpoint1"
contract="ServiceReference1.IService"
            name="wsHttpEnpoint">
            <identity>
                <certificate encodedValue="<<Encode Value>>" />
            </identity>
        </endpoint>
    </client>
</system.serviceModel>
```

Step 9 – Test the Client and WCF Service

In this step, you access the WCF service, pass the user credentials, and make sure that the username authentication works.

1. In your Client project, drag a Button control onto your form.
2. Double-click the **Button** control to show the underlying code.
3. Create an instance of the proxy and call the **GetData** operation of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                    WCFTestService.ServiceClient();
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the Client project and then click **Set as Startup Project**.

5. Run the Client application by pressing F5 or Ctrl+F5.
   When you click the button on the form, the message "**You entered: 123**" should appear.

Additional Resources

- For more information on how to work with temporary certificates, see "How to: Create Temporary Certificates for Use During Development" at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on how to view certificates using the Microsoft Management Console (MMC) snap in, see "How to: View Certificates with the MMC Snap-in" at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see "Differences Between Service Certificate Validation Done by Internet Explorer and WCF" at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see "Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security" at http://msdn2.microsoft.com/en-us/library/aa702579.aspx

How To – Use Certificate Authentication and Transport Security in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of using client certificates to authenticate your users with transport security. First, you will learn how to create and install a client certificate for authentication and a service certificate for transport security, during development. You will then learn how to configure a binding that implements **IMetadataExchange** in a WCF service, and how to create a svcutil.exe.config file to allow proxy creation from the client, which is necessary when implementing transport security and certificate authentication when a WCF service is hosted in Internet Information Services (IIS). Finally, you will learn how to correctly configure security settings in IIS, and how to test the service with a sample WCF client.

Contents

Objectives

- Learn how to create and use a temporary certificate for authentication and transport security.

- Learn where to store the temporary certificate.
- Learn how to configure a custom binding that implements **IMetadataExchange** in a WCF service to allow publishing of metadata with certificate authentication and transport security.
- Learn how to configure a svcutil.exe.config file to be able to create a proxy to the WCF service with transport security and certificate authentication.

Overview

When developing a WCF service that uses X.509 certificates to provide client authentication and transport security, it is necessary to work with temporary certificates. This is because production certificates are expensive and may not be readily available. There are two options for specifying trust on a certificate:

- **Peer trust** – Validates the certificate directly.
- **Chain trust** – Validates the certificate against the issuer of a certificate known as a root authority.

Additionally, a certificate revocation list (CRL) validation is performed during certificate authentication. This validation checks the list of certificates that were revoked by the root certificate. Three modes of revocation exist:

- **Online –** The CRL list is retrieved and checked online, requiring connectivity to the computers that contains the CRL.
- **Offline –** The CRL list is retrieved and checked online and is then cached offline for subsequent validation.
- **NoCheck –** No validation is performed.

For the purposes of this How To article, the CRL is checked without configuration changes when using certificate authentication., The article also allows for chain trust validation when using transport security.

To use chain trust validation during development time, you first create a self-signed root certificate authority (CA) and install it in the Trusted Root Certification Authority in the Local Machine. The certificate used by WCF is then created and signed by the root self-signed certificate and installed in the Personal store of the Computer Account. To allow the CRL validation to succeed, you create a self-signed root CRL file and install it in the Trusted Root Certification Authority store of the Local Machine.

You will use makecert.exe to create a private key file and a certificate to act as your root certificate authority (CA). You will then create a certificate revocation list file from the private key that will act as your revocation list file for the root CA. You will have to install the root certificate and CRL file. Finally, you will create and install the temporary certificate from the root certificate, using the private key to sign and generate the key.

Summary of Steps

- **Step 1 – Create and Install a Temporary Certificate for Transport Security**
- **Step 2 – Create and Install a Temporary Client Certificate for Certificate Authentication**
- **Step 3 – Create a Sample WCF Service**
- **Step 4 – Configure wsHttpBinding with Certificate Authentication and Transport Security**
- **Step 5 – Configure the mex Endpoint to Use wsHttpbinding with Certificate Authentication Configuration**
- **Step 6 – Configure the Virtual Directory to Use SSL and Require Client Certificates**
- **Step 7 – Create a Test Client**
- **Step 8 – Create a Svcutil Configuration File in the Client Machine**
- **Step 9 – Create a Proxy with the svcutil.exe Tool**
- **Step 10 – Test the Client and WCF Service**

Step 1 – Create and Install a Temporary Certificate for Transport Security

In this step, you create and install a temporary certificate for transport security on the server. You are also be required to install the root CA on the client for trust chain validation to succeed when browsing the service in Microsoft Internet Explorer, creating the proxy to the service, and calling the service from the proxy. For a complete set of steps on how to create certificates for transport security, refer to the document "How To – Create and Install Temporary Certificates in WCF for Transport Security During Development."

Step 2 – Create and Install a Temporary Client Certificate for Certificate Authentication

In this step, you create and install a temporary client certificate for certificate authentication on the client. You are also required to install the root certificate authority (CA) and the CRL on the server in order for trust chain and revocation validation to succeed. For a complete set of steps on how to create certificates for certificate authentication, refer to the document "How To: Create and Install Temporary Client Certificates in WCF for Certificate Authentication During Development."

Step 3 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio.

1. In Visual Studio, on the menu, click **File** and then click **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http** and specify the virtual directory to be created in the **Path** with https

(e.g., https://ServerName/WCFTestService).
Note that the server name needs to match the certificate name either by NetBIOS or Domain Name System (DNS) name.

3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample WCF service.

Step 4 – Configure wsHttpBinding with Certificate Authentication and Transport Security

In this step, you configure the WCF service to use certificate authentication and transport security.

1. Right-click the Web.config file of the WCF service and then click **Edit WCF Configuration**.
2. In the Configuration Editor, in the **Configuration** section, expand **Service** and then expand **Endpoints**.
3. Select the first node **[Empty Name]** and set the **Name** attribute to **wsHttpEndpoint**.
   By default, the name will be empty because it is an optional attribute.
4. Click the **Identity** tab and then delete the **DNS** attribute value.
5. In the Configuration Editor, select the Bindings folder.
6. In the **Bindings** section, choose **New Binding Configuration**.
7. In the **Create a New Binding** dialog box, select **wsHttpBinding**.
8. Click **OK**.
9. Set the **Name** of the binding configuration to some logical and recognizable name; for example, wsHttpEndpointBinding.
10. Click the **Security** tab.
11. Make sure that the **Mode** attribute is set to **Transport**.
12. Set the **TransportClientCredentialType** to **Certificate** by selecting this option from the drop-down list.
13. In the **Configuration** section, select the **wsHttpEndpoint** node.
14. Set the **BindingConfiguration** attribute to **wsHttpEndpointBinding** by selecting this option from the drop-down list.
    This associates the binding configuration setting with the binding.
15. In the Configuration Editor, on the **File** menu, select **Save**.
16. In Visual Studio, open your configuration and comment out the identity element. It should look as follows:

```
<!--<identity>
  <dns value="" />
</identity>-->
```

17. In Visual Studio, verify your configuration. The configuration should look as follows:

```
…
<system.serviceModel>
<bindings>
    <wsHttpBinding>
      <binding name="wsHttpEndpointBinding">
        <security mode="Transport">
          <transport clientCredentialType="Certificate" />
        </security>
      </binding>
    </wsHttpBinding>
  </bindings>
<client/>
<services>
    <service behaviorConfiguration="ServiceBehavior"
name="MyService">
      <endpoint binding="wsHttpBinding"
bindingConfiguration="wsHttpEndpointBinding"
        name="wsHttpEndpoint" contract="IService" />
<endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange"/>
  </service>
  </services>…
```

Step 5 – Configure the mex Endpoint to Use wsHttpbinding with Certificate Authentication Configuration

In this step, you change the configuration of the **mex** endpoint from **mexHttpBinding** (the default) to use **wsHttpbinding** with the configuration you created in the previous step in order to use certificate authentication. **mexHttpendpoint** cannot be used for certificate authentication because the Web site requires Secure Sockets Layer (SSL), and **mexHttpsendpoint** cannot be used either because it does not support certificate authentication configuration in IIS. To create a proxy to a WCF service hosted in IIS with a certificate authentication schema, you need an endpoint that implements **IMetadataExchange** with **wsHttpbinding** with a security configuration that allows certificate authentication.

1. In the Configuration Editor, in the **Configuration** section, expand **Service** and then expand **Endpoints**.
2. Select the second node **[Empty Name]** and set the **Name** attribute to **mexEndpoint**
   By default, the name will be empty because it is an optional attribute.
3. Click the **Binding** attribute and change it to **wsHttpbinding** by selecting this option from the drop-down list.
4. Click the **BindingConfiguration** attribute and change it to **wsHttpEndpointBinding** by selecting this option from the drop-down list.
   This associates the **mex** endpoint with the binding configuration setting that configures certificate authentication.
5. In the Configuration Editor, on the **File** menu, select **Save**.

6. In Visual Studio, verify your configuration. The configuration should look as follows. Notice that that the endpoint is now a **wsHttpbinding** endpoint that implements **IMedataExchange** (shown in bold).

```
…
<bindings>
    <wsHttpBinding>
      <binding name="wsHttpEndpointBinding">
        <security mode="Transport">
          <transport clientCredentialType="Certificate" />
        </security>
      </binding>
    </wsHttpBinding>
  </bindings>
            <client/>
            <services>
    <service behaviorConfiguration="returnFaults"
name="MyService">
      <endpoint binding="wsHttpBinding"
bindingConfiguration="wsHttpEndpointBinding"
        name="wsHttpEndpoint" contract="IService" />
      <endpoint address="mex" binding="wsHttpBinding"
bindingConfiguration="wsHttpEndpointBinding"
        name="mexEndpoint" contract="IMetadataExchange" />
    </service>
  </services>…
```

Step 6 – Configure the Virtual Directory to Use SSL and Require Client Certificates

In this step, you configure the virtual directory in IIS to use SSL security and to require client certificates.

1. Click **Start** and then click **Run**.
2. In the **Run** dialog box, type **inetmgr** and then click **OK**.
3. In the **Internet Information Services (IIS) Manager** dialog box, expand the **(local computer)** node, and then expand the **Web Sites** node.
4. Expand **Default Web Site** and then right-click the virtual directory.
5. In the **Virtual Directory Properties** dialog box, click the **Directory Security** tab, and then click **edit** on secure communications. Click **Require Secure Channel(SSL)** and then click **Require Client Certificates**.
   You can now browse the service using Internet Explorer by navigating to https://ServerName/WCFTestService. Internet Explorer will prompt you to choose a certificate from a list of certificates installed in the user and personal stores.

Step 7 – Create a Test Client

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.

Step 8 – Create a Svcutil Configuration File in the Client Machine

In this step, you create the svcutil.exe.config file that you need in order to create a proxy to the service.

1. Right-click the project in Visual Studio, click **Add**, and then click **New item.** Select **text file** and name the file **svcutil.exe.config**.
2. Copy the configuration file below, paste it into the svcutil.exe.config file, click **File**, and then click **Save**.
   Verify the following in your configuration file:

   a. The client certificate name is correctly specified. The certificate name is determined by the attribute **findValue "CN=…"** under the **client credentials** node.
   b. The client certificate location is correctly specified. The certificate location is determined by the attributes **storeLocation="CurrenUser" storeName="My"**. **CurrentUser** and **My** represent the current user and personal store. This is the default location of the client certificate as specified in Step 2 above.

```
<configuration>
  <system.serviceModel>
    <client>
      <endpoint
behaviorConfiguration="ClientCertificateBehavior"
   binding="wsHttpBinding"
        bindingConfiguration="Binding1"
contract="IMetadataExchange"
     name="https" />
    </client>
    <bindings>
      <wsHttpBinding>
        <binding name="Binding1">
          <security mode="Transport">
            <transport clientCredentialType="Certificate" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <behaviors>
      <endpointBehaviors>
        <behavior name="ClientCertificateBehavior">
```

```
            <clientCredentials>
              <clientCertificate findValue="CN=clienttempcert"
            storeLocation="CurrentUser"
                 storeName="My"
            x509FindType="FindBySubjectDistinguishedName" />
             </clientCredentials>
           </behavior>
         </endpointBehaviors>
       </behaviors>
     </system.serviceModel>
   </configuration>
```

Step 9 – Create a Proxy with the svcutil.exe Tool

In this step, you create a proxy to the service by using the svcutil.exe tool and the svcutil.exe.config file.

1. Copy svcutil.exe from C:\Program Files\Microsoft Visual Studio 8\Common7\IDE to the same location as the svcutil.exe.config file created in previous step.
2. Open a command prompt, navigate to the same directory as the svcutil.exe.config file, and run the following command:

   ```
   .\svcutil https://ServerName/WCFTestService /config:app.config.
   ```

   This will generate two files: MyService.cs and app.config.
3. In Visual Studio, right-click the project , click **Add**, and then click **Existing item.** The location defaults to the same directory as the app.config file created in the previous step.
4. Press the CTRL key and then select the app.config and MyService.cs files.
5. Right-click the app.config file of the WCF service and then click **Edit WCF Configuration**.
6. In the Configuration Editor, in the **Configuration** section, expand **Advanced**, click **EndpointsBehaviors**, and then click **New Endpoint Behavior**.
7. In the **Name** text box, type **ClientEndPointBehavior** and then select **client credentials.**
8. Double-click **clientCredentials**, expand **clientCredentials**, and then click **client certificates**.
9. In the **client certificates**, click **findValue**, enter the name of the certificate, click the store location and store name, and then select the correct values for your certificate.
   These values should be the same as in the svc.util.exe.config file.
10. Under **Endpoints**, click **wsHttpEndpoint B,** select **BehaviorConfiguration**, and then select **ClientEndPointBehavior**.

Step 10 – Test the Client and WCF Service

In this step, you access the WCF service, pass the user credentials, and make sure that the username authentication works.

1. In your client project, drag a Button control onto your form.
2. Double-click the Button control to show the underlying code.
3. Create an instance of the proxy and call the **GetData** operation of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      ServiceClient proxy = new ServiceClient();
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5.
   When you click the button on the form, the message “You entered: 123” should appear.

Additional Resources

- For more information on working with temporary certificates, see “How to: Create Temporary Certificates for Use During Development” at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on viewing certificates using the Microsoft Management Console (MMC) snap in, see “How to: View Certificates with the MMC Snap-in” at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see “Differences Between Service Certificate Validation Done by Internet Explorer and WCF” at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see “Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security” at http://msdn2.microsoft.com/en-us/library/aa702579.aspx

How To – Use Delegation for Flowing the Original Caller Credentials to the Back End in WCF Calling from Windows Forms

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article shows you how to flow the original caller credentials to the back end in a WCF service that has been called from a Windows Forms application. The article shows you how to configure the WCF service, implement delegation, and test the service with a sample Windows Forms client.

Contents

Objectives

- Learn how to configure a WCF process identity trusted for delegation.
- Learn how to constrain the delegation.
- Learn how to impersonate the original caller.

Overview

When a WCF Service impersonates the original caller, it accesses resources by using the security context of the authenticated user. However, the application can only access local resources. To access network resources while impersonating an original caller, your service must use delegation. If your service uses Kerberos authentication to

authenticate its users, you can use delegation to pass the caller's identity through the layers of your application, and to access network resources.

Note: If your application does not use Kerberos authentication, you can use protocol transition to switch from a non-Kerberos authentication mechanism to Kerberos, and then use delegation to pass on the identity.

Kerberos delegation by default is unconstrained, and servers that are configured as trusted for delegation in Microsoft Active Directory® can access any network resources or any machine on the network while using the impersonated user's security context. This represents a potential security threat, particularly if the Web server is compromised.

To address this issue, you should use constrained delegation. This allows administrators to specify exactly which services on a downstream server or a domain account can be accessed when using an impersonated user's security context.

Note The list of services that can be accessed by delegation is maintained in an Active Directory list referred to as the A2D2 list.

Summary of Steps

- **Step 1 – Create a Sample WCF Service**
- **Step 2 – Configure the WCF Service to Use Windows Authentication**
- **Step 3 – Identify and Configure the Remote Service to Be Accessed**
- **Step 4 – Configure the WCF Service Identity Trusted for Constrained Delegation**
- **Step 5 – Impersonate the Original Caller in the WCF Service**
- **Step 6 – Create a Test Client Application**
- **Step 7 – Add a WCF Service Reference to the Client**
- **Step 8 – Test the Client and WCF Service**

Step 1 – Create a Sample WCF Service

In this step, you create a sample WCF service in Visual Studio, hosted in an Internet Information Services (IIS) virtual directory.

1. In Visual Studio, on the menu, click **File** > **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http** and specify the virtual directory to be created in the **Path (**e.g., http://localhost/WCFServiceDelegation).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory, a solution file, and a sample WCF service for the solution.
4. In Microsoft Internet Explorer, browse to your WCF service at http://localhost/WCFServiceDelegation/Service.svc.
   You should see details of your WCF service in the browser.

Step 2 – Configure the WCF Service to Use Windows Authentication

By default, Visual Studio configures your WCF service to use wsHttpBinding with Windows authentication and Message Security.

- In Visual Studio, verify your configuration settings in Web.config. The configuration should look as follows:

```
…
<services>
  <service name="Service" behaviorConfiguration="ServiceBehavior">
    <endpoint address="" binding="wsHttpBinding" contract="IService">
      <identity>
        <dns value="localhost"/>
      </identity>
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
        contract="IMetadataExchange"/>
  </service>
</services>
…
```

Step 3 – Identify and Configure the Remote Service to Be Accessed

In this step, you identify the remote service to be accessed on behalf of the original caller. This service needs to be enabled for Windows authentication and configured with access rights to the original caller.

For the purposes of this exercise, you will access the Microsoft SQL Server® database on a remote server on behalf of the original caller.

1. If you use a custom domain account to run SQL Server, you must create a service principal name (SPN) for this account. You can do this by using the following command:

```
setspn -A MSSQLSvc/ databaseservername.fullyqualifieddomainname
domain\customAccountName
```

   If you run SQL Server by using the System account (which is not recommended because of the associated high privileges that an attacker could exploit), an SPN is created automatically for you.

2. To allow access to SQL Server, you must create a SQL Server login for each of your application's end users or for a set of groups that the users belong to, and grant them read access to the target database.

Step 4 – Configure the WCF Service Identity Trusted for Constrained Delegation

In this step, you configure Active Directory to allow your WCF service to use constrained delegation to access a remote database server.

If your WCF Service runs using the Network Service machine account, you must enable constrained delegation for your WCF server computer. However, if your WCF service runs under a custom domain account, you must enable constrained delegation for the custom domain account.

Note If you use a custom domain account for running your WCF service, create an SPN for your custom domain account. Kerberos requires an SPN to support mutual authentication.

To configure constrained delegation for the machine account

This procedure assumes that you are running your WCF service under the Network Service machine account.

1. On the domain controller, start the Microsoft Management Console (MMC) Active Directory Users and Computers snap-in.
2. In the left pane of the MMC snap-in, click the **Computers** node.
3. In the right pane, double-click your WCF server computer to display the Properties dialog box.
4. On the **Delegation** tab of the Properties window for the WCF server computer, **Do not trust the computer for delegation** is selected by default. To use constrained delegation, select **Trust this computer for delegation to specified services only**. You specify precisely which service or services can be accessed in the bottom pane.
5. Beneath **Trust this computer for delegation to specified services only**, keep the default option **Use Kerberos only** selected.
6. Click the **Add** button to display the **Add Services** dialog box.
7. Click the **Users or computers** button.
8. In the **Select Users or Computers** dialog box, type the name of your database server computer if you are running SQL Server as System or Network Service. Alternatively, if you are running SQL Server by using a custom domain account, enter that account name instead and then click **OK**.

9. You will see all the service principal names configured for the selected user or computer account. To restrict access to SQL Server, select the **MSSQLSvc** service, and then click **OK**.

Note If you want to delegate to a file on a file share, you need to select the Common Internet File System (CIFS) service.

To configure constrained delegation for a custom domain account

This procedure assumes that you are running your Web application under a custom domain account.

1. Create an SPN for your custom domain account. Kerberos requires an SPN to support mutual authentication. To create an SPN for the domain account:
   i. Install the Windows Server 2003 Tools from the Microsoft Windows Server® 2003 CD.
   ii. From a command prompt, run the **Setspn** tool twice from the **C:\Program Files\Support Tools** directory as shown below:

   setspn -A HTTP/*wcfservername domaincustomAccountName***

   setspn -A HTTP/*wcfservername.fullyqualifieddomainname*

   domain**customAccountName

   Note You can only have a single SPN associated with any HTTP service (DNS) name, which means you cannot create SPNs for different service accounts mapped to the same HTTP server unless they are on different ports. The SPN can include a port number.

2. On the domain controller, start the Microsoft Management Console (MMC) Active Directory Users and Computers snap-in.
3. In the left pane of the MMC snap in, click the **Users** node.
4. In the right pane, double-click the user account you are using to run the WCF service.
   This displays the user account properties.
5. On the **Delegation** tab of the Properties window for the WCF server computer, **Do not trust the computer for delegation** is selected by default. To use constrained delegation, select **Trust this computer for delegation to specified services only**. You specify precisely which service or services can be accessed in the bottom pane.
6. Beneath **Trust this computer for delegation to specified services only**, keep the default option **Use Kerberos only** selected.
7. Click the **Add** button to display the **Add Services** dialog box.

8. Click the **Users or computers** button.
9. In the **Select Users or Computers** dialog box, type the name of your database server computer if you are running SQL Server as System or Network Service. Alternatively, if you are running SQL Server by using a custom domain account, enter that account name instead and then click **OK**.
10. You will see all the service principal names configured for the selected user or computer account. To restrict access to SQL Server, select the **MSSQLSvc** service, and then click **OK**.

Step 5 – Impersonate the Original Caller in the WCF Service

Perform the following steps to declaratively impersonate specific operations:

1. In the Solution Explorer, expand the App_Code folder under your WCF Service project, and then open the Service.cs file.
2. Add a **using** statement for the **System.Security.Principal** namespace.
3. Set the impersonation required on the operation implementation of the specific operation as follows:

```
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetData(int value)
{
  return string.Format("Hi, {0}, you have entered: {1}",
                       WindowsIdentity.GetCurrent().Name, value);
}
```

4. Add the database access code to the WCF Service operation implementation. The remote database is accessed using the original caller's security context.

```
public string GetData(int value)
{
   // Access the database
    using (SqlConnection conn = new SqlConnection())
    {
      conn.ConnectionString = "Connection String";
      conn.Open();
      SqlCommand cmd = new SqlCommand("Select * from <<tableName>>",
                                      conn);
      SqlDataAdapter da = new SqlDataAdapter(cmd);
      da.Fill(dt);
    }

   return string.Format("Hi, {0}, you have entered: {1}",
                       WindowsIdentity.GetCurrent().Name, value);
}
```

Step 6 – Create a Test Client Application

In this step, you create a Windows Forms application that you will use to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK** to create a Windows Forms application for testing.

Step 7 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service.

1. Right-click your client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF service: http://localhost/WCFServiceDelegation/Service.svc
3. In the **Namespace** field, change ServiceReference1 to **WCFTestService**.
4. Click **OK**.
   A reference to WCFTestService should appear beneath Service References in your client project.

Step 8 – Test the Client and WCF Service

In this step, you access the WCF service and make sure that it impersonates as expected.

1. In your client project, drag a **Button** control onto your form.
2. Double-click the **Button** control to show the underlying code.
3. Create an instance of the proxy and call the **GetData** method of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                              WCFTestService.ServiceClient();
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5. When you click the button on the form, it should display the message “Hi, <<logged in user id>>, you have entered: 123”.

Additional Resources

- For more information on impersonation, see “Delegation and Impersonation with WCF” at http://msdn2.microsoft.com/en-us/library/ms731090.aspx.
- For further information on impersonation, see “How to: Impersonate a Client on a Service” at http://msdn2.microsoft.com/en-us/library/ms730088.aspx.

How To – Use Health Monitoring to Instrument a WCF Service for Security

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of configuring a WCF service for Health Monitoring in order to instrument a custom event. The article shows you how to create a custom web event, configure a WCF service for Health Monitoring, instrument a WCF service for security events, and create a test client application to verify the events in the Event Log.

Contents

Objectives

- Learn to create a custom Web event.
- Learn to configure a WCF service for Health Monitoring.
- Learn to instrument a WCF service.

Overview

The Health Monitoring feature in WCF supports many standard events that you can use to check the health of your WCF service. This feature supports an event provider model. It allows you to instrument your WCF service and monitors user management events around authentication and authorization. You can track access to sensitive operations such as financial transactions or access to sensitive data by using the Health Monitoring feature to detect and react to potentially suspicious behavior.

This article shows you how to create a custom Web event in a class library project in Visual Studio 2008. You will then create a sample WCF service project and configure the service to use the Health Monitoring feature. Next, you will instrument the service by raising the custom event. Finally, you will create an ASP.NET test client project to verify the security events in the Event Log by invoking the custom event.

Summary of Steps

- **Step 1 – Create a Custom Web Event**
- **Step 2 – Create a WCF Service for Monitoring**
- **Step 3 – Configure Your WCF Service for Health Monitoring**
- **Step 4 – Instrument Your WCF Service**
- **Step 5 – Create a Test Client**
- **Step 6 – Add a WCF Service Reference to the Client**
- **Step 7 – Test the Client and WCF Service**
- **Step 8 – Verify the Service Events in the Event Log**

Step 1 – Create a Custom Web Event

In this step, you create a custom Web event by creating a class that inherits from **System.Web.Management.WebAuditEvent**.

1. In Visual Studio, on the menu, click **File** -> **New Project**.
2. In the **Templates** section, select **Class Library**. Specify the name of the project and the location to be created in the **Path (**e.g., C:/Projects/MyEventLibrary).
3. In the **New Project** dialog box, click **OK** to create a Class Library project and sample class file named (Class1.cs).
4. Rename Class1.cs as **MyEvent.cs**.
5. Add a reference to your new project to **System.Web** and add the **System.Web.Management** namespace to the top of MyEvent.cs.
6. Derive MyEvent from **WebAuditEvent** and create appropriate public constructors that call the protected equivalents in the parent **WebAuditEvent** class, as follows:

```
using System.Web.Management;

 public class MyEvent : WebAuditEvent
    {

        public MyEvent(string msg, object eventSource, int
eventCode)
            : base(msg, eventSource, eventCode)
        {
        }

        public MyEvent(string msg, object eventSource, int
eventCode, int eventDetailCode)
            : base(msg, eventSource, eventCode, eventDetailCode)
        {
```

```
        }
```

7. To log some custom details, override the **FormatCustomEventDetails** method as follows:

```
   public override void
FormatCustomEventDetails(WebEventFormatter formatter)
        {
            base.FormatCustomEventDetails(formatter);

            // Add some custom data.
            formatter.AppendLine("");
            formatter.IndentationLevel += 1;
            formatter.AppendLine("******** SampleWebAuditEvent
Start ********");

            formatter.AppendLine(string.Format("Request path:
{0}",
                RequestInformation.RequestPath));

            formatter.AppendLine(string.Format("Request Url:
{0}",
                RequestInformation.RequestUrl));

            // Display some custom event message
            formatter.AppendLine("Some Critical Event Fired");

            formatter.AppendLine("******** SampleWebAuditEvent
End ********");

            formatter.IndentationLevel -= 1;
        }
```

8. Build the assembly by compiling the project.

Step 2 – Create a WCF Service for Monitoring

In this step, you create a WCF service in Visual Studio.

1. In Visual Studio, on the menu, click **File** -> **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to Http and specify the virtual directory to be created in the **Path (**e.g., http://localhost/HMWCFService).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample WCF service.
4. Browse to your WCF service (i.e., http://localhost/HMWCFService/Service.svc). You should see details of your WCF service.

Step 3 – Configure Your WCF Service for Health Monitoring

In this step, you configure the WCF service to use Health Monitoring. You can configure your application to use any of the three default providers. For this exercise, you will use the **EventLogWebEventProvider**, which uses the **EventLogWebEventProvider** class to write entries to the Windows application Event Log.

- In the Web.config file of your service application, add the following code, which specifies the event mapping and the rules for using the **EventLogProvider** for the custom event type **MyEventLibrary.MyEvent**.

```
…
<system.web>
      <healthMonitoring>
            <eventMappings>
                  <add name="Some Custom Event"
type="MyEventLibrary.MyEvent, MyEventLibrary"/>
            </eventMappings>
            <rules>
                  <add name="Custom event" eventName="Some Custom
Event" provider="EventLogProvider" minInterval="00:00:01"/>
            </rules>
      </healthMonitoring>
</system.web>
…
```

Step 4 – Instrument Your WCF Service

In this step, you instrument your WCF service to raise custom events.

1. In the Solution Explorer, select the WCF Service project and add a reference to the **Class Library project** created in step1.
2. Expand the App_Code folder, open **IService.cs**, and add the following operation contract:

```
 [OperationContract]
 string InvokeCriticalEvent();
```

3. Add a reference to **System.Web** and then add the **System.Web.Management** namespace to the top of the **Service.cs** as follows:

```
using CustomEvents;
using System.Web.Management;
```

4. Implement the above contract by creating a new custom event object of type **MyEvent** and calling its **Raise** method to fire the event as follows:

```
public string InvokeCriticalEvent()
 {
     MyEvent obj = new MyEvent("Invoking Some Custom Event",
this, WebEventCodes.WebExtendedBase + 1);
```

```
        obj.Raise();
        return "Critical event invoked";
    }
```

5. Compile the WCF service project and test the service.

Note: When you raise a custom Web event, you must specify an event code that is greater than **System.Web.Management.WebEventCodes.WebExtendedBase**. Codes that are less than this value are reserved for system-generated events.

Step 5 – Create a Test Client

In this step, you create an ASP.NET application to monitor your WCF service.

1. In the Solution Explorer, right-click your solution and then click **New Website**.
2. In the **Templates** section, select **ASP.NET Website**. Make sure that the **Location** is set to Http and specify the virtual directory to be created in the **Path (**e.g., http://localhost/HMWCFService).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample ASP.NET Web site.

Step 6 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service.

1. Right-click your ASP.NET client application and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF service, (e.g, http://localhost/HMWCFService/Service.svc) and then click **Go**.
3. In the **Web reference name field**, change ServiceReference1 to HMWCFService.
4. Click **Add Reference**.
   A reference to HMWCFService should appear beneath Web References in your client project.

Step 7 – Test the Client and WCF Service

In this step, you access the WCF service and invoke the custom event.

1. In your ASP.NET test application project, drag a Button control onto your Web form.
2. Double-click the Button control to show the underlying code.
3. Create an instance of the proxy, and then call the **InvokeCriticalEvent** operation of your WCF service. The code should look as follows:

```
protected void button1_Click(object sender, EventArgs e)
{
      HMWCFService.ServiceClient myService = new
                    HMWCFService.ServiceClient();
```

```
        Response.Write(myService.InvokeCriticalEvent());
        myService.Close();
}
```

4. Right-click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5. When you click the button on the form, the message "**Critical event invoked**" should appear.

Step 8 – Verify the Service Events in the Event Log

In this step, you verify the WCF service events in the Application Event Log.

1. On your Service host machine, click **Start** and then click **Run**.
2. In the command line, type **eventvwr** and then click **OK** to open the Event Viewer window.
3. In the left pane, select the **Application** node to view a list of application events in the right pane.
4. In the list, search for the latest event. You will see an event of **Web Event** category, **ASP.NET <<version no>>** source, and **Information** type.
5. Open the event and view the following information, which appends your custom message event:

```
Event code: 100001
Event message: Invoking Some Custom Event
Event time: 3/31/2008 10:55:42 AM
Event time (UTC): 3/31/2008 5:25:42 AM
Event ID: 1515c05420ea46e189f83e1550cb1f8a
Event sequence: 10
Event occurrence: 2
Event detail code: 0

Application information:
…
Process information:
…

Request information:
…
Custom event details:
    ******** SampleWebAuditEvent Start ********
    Request path: /Health/Default.aspx
    Request Url: http://localhost/HMWCFService/Default.aspx
    Password changed
    ******** SampleWebAuditEvent End ********
…
```

Additional Resources

- For more information on instrumenting ASP.NET applications, see "How To: Instrument ASP.NET 2.0 Applications for Security" at http://msdn2.microsoft.com/en-us/library/ms998325.aspx

- For more information on Health Monitoring, see “How To: Use Health Monitoring in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998306.aspx

How To: Use netTcpBinding with Windows Authentication and Message Security in WCF Calling from Windows Forms

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Windows Forms
- Microsoft Visual Studio® 2008

Summary

This how to shows you how to use the **netTcpBinding** with Windows Authentication and Message security. **netTcpBinding** is used for communicating with WCF clients in an intranet and provides transport security with windows authentication by default. This how to shows you how to configure the service to use message security instead of transport security. In this how to, the WCF service is hosted in a Windows service.

Contents

Objectives

- Create a WCF service hosted in a Windows service.
- Learn how to expose the WCF service with message security.
- Learn how to use Windows tokens for encrypting and signing your messages.
- Learn why you need service principle names (SPNs) and how to create them

Overview

Windows Authentication is suited for scenarios in which your users have domain credentials. In the scenario described in this How To article, users are authenticated by Windows Authentication. The scenario described in this How To article uses the **netTcpBinding** binding to expose a WCF service to WCF-enabled clients. The

netTcpBinding binding offers improved performance over an HTTP binding. In this scenario WCF is hosted in a Windows service. The WCF service with **netTcpBinding** can be consumed by a WCF-enabled .NET application through the use of a service reference. The Visual Studio service reference generates a proxy class to abstract the underlying message-based communication. WCF message security is used to support a secure communication channel in a end-to-end scenario. In general, you should always use transport security unless you need the additional flexibility that message security affords you. For example, you would use message security for scenarios in which there are intermediaries who need to inspect and re-route the message.

In this How To, you will create a Windows service to host your WCF service. You will then create sample WCF service in Visual Studio 2008 and configure the service to use **netTcpBinding** with message security through the use of the WCF Configuration Editor. Next, you will configure a **mexTcpBinding** so that the service can expose its metadata to clients from which they can generate a WCF proxy and call your service. Finally, you will create a test client to verify that the service is working properly.

Solution Summary

- **Binding**: By default, netTcpBinding offers improved performance over an HTTP binding and is the ideal choice for cross machine communication between WCF clients and a WCF service, in an intranet.

- **Security Mode**: *Transport security* is the default security mode for netTcpBinding and should be preferred over *Message* security for better performance. If needed, message security can provide greater control over signing and encryption of the message.

- **Client Authentication**: Since this binding is used inside an intranet, *Windows* is the recommended client authentication mechanism though the default is *UserName.* The other possible values are *None*, *Certificate* and *IssuedToken*.

- **Algorithm Suite**: The default message encryption algorithm used is *Basic256* and should suffice for most scenarios. A stronger encryption algorithm can be chosen for increased security.

- **Hosting Consideration**: This how-to hosts WCF in a Windows service. In general, netTcpBinding services can be hosted in a windows service, IIS 7.0 (not IIS 6.0 or lower), WAS or can be self-hosted. The choice should be based on the deployment requirements of the service.

Summary of Steps

- **Step 1 – Create a Windows Service**
- **Step 2 – Create a Sample WCF Service**
- **Step 3 – Modify the Windows Service to Host the WCF Service**
- **Step 4 – Configure the WCF Service to Use netTcpBinding with Message Security**
- **Step 5 – Configure the WCF Service to Publish Metadata**
- **Step 6 – Install the Windows Service**
- **Step 7 – Create a Test Client Application**
- **Step 8 – Test the Client and WCF Service**

Step 1 – Create a Windows Service

In this step, you create a Windows service to host your WCF service.

1. In Visual Studio, on the **File** menu, click **New** and then click **Project**.
2. In the **New Project** dialog box, in the **Project Types** section, select **Windows** under **Visual C#.**
3. In the **Templates** section, select **Windows Service**, and type the name of your project (WCFServiceHost) in the **Name** field.
4. In the **Add a Project** dialog box, click **OK** to add a sample Windows service to the solution.
5. Right-click **Service1.cs** and then click **View Designer**.
6. Right-click the designer and then click **Add Installer** to add the ProjectInstaller.cs file with two objects, **serviceProcessInstaller1** and **serviceInstaller1**.
7. In the designer view of ProjectInstaller.cs, right-click **serviceProcessInstaller1** and then click **Properties**.
8. In the **Properties** pane, set the **Account** attribute to **NetworkService**.
   This will run your Windows service under the Network Service account.

Step 2 – Create a Sample WCF Service

In this step, you add a WCF service to the Windows service that will host it.

1. Right-click the Windows service project, click **Add**, and then click **New Item**.
2. In the **Add New Item** dialog box, select **WCF Service**.
3. Set the **Name** as **MyService.cs** and then click the **Add**.
   Note that the configuration file, App.config, gets added automatically.
4. Modify the **DoWork()** method signature in **IMyService.cs** to accept a string parameter and return a string data type as shown below:

   ```
   string DoWork(string value);
   ```
5. Modify the **DoWork()** method in **MyService.cs** to accept a string parameter and return a string data type as below.

```
public string DoWork(string value)
{
      return "Welcome " + value;
}
```

Step 3 – Modify the Windows Service to Host the WCF Service

In this step, you add code to the **OnStart()** and **OnStop()** methods to start and stop the WCF Service inside the Windows service process.

1. In the Solution Explorer, right-click **Service1.cs** and then click **View Code**.
2. In the Service1.cs file, add a using statement as follows:

```
using System.ServiceModel;
```

3. Declare an internal static member of **ServiceHost** type as follows:

```
internal static ServiceHost myServiceHost = null;
```

4. Add code to the **OnStart** method of the Windows service, to open the service:

```
protected override void OnStart(string[] args)
{
   if (myServiceHost != null)
   {
       myServiceHost.Close();
   }

   myServiceHost = new ServiceHost(typeof(MyService));
   myServiceHost.Open();
}
```

5. Add code to the **OnStop** method of the Windows service, to close the service host

```
protected override void OnStop()
{
   if (myServiceHost != null)
   {
      myServiceHost.Close();
      myServiceHost = null;
   }
}
```

6. Build the solution and verify that your project produces WCFServiceHost.exe in your project \bin directory.

Step 4 – Configure the WCF Service to Use netTcpBinding with Message Security

In this step, you configure your WCF service to use **netTcpBinding** and message security.

1. In the Solution Explorer, Right-click the App.config file and then click **Edit WCF Configuration**.
   If you do not see the Edit WCF Configuration option, on the **Tools** menu, click **WCF Service Configuration Editor**. Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu.
2. In the Configuration Editor, expand the **Services** node and then expand **WCFHostService.MyService**.
3. Select the **Host** node, select the default **BaseAddress** in the **Base addresses** section, and then click **Delete**.
4. Click **New** and then in the **Base Address Editor** dialog box, set the **Base address:** to **net.tcp://localhost:8523/WCFTestService**.
5. Expand the **Endpoints** node, select the first **[Empty Name]** node, and then set the set the **Name** attribute to **NetTcpBindingEndpoint**.
6. Set the **Binding** attribute to **netTcpBinding**.
7. In the Configuration Editor dialog box, on the **File** menu, select **Save**.
8. In Visual Studio, verify the configuration in your App.config, which should look as follows:

```
<services>
    <service
behaviorConfiguration="WCFHostService.MyServiceBehavior"
        name="WCFHostService.MyService">
        <endpoint address="" binding="netTcpBinding"
               bindingConfiguration=""
               name="NetTcpBindingEndpoint"
               contract="WCFHostService.IMyService">
            <identity>
                <dns value="localhost" />
            </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding"
                  contract="IMetadataExchange" />
        <host>
            <baseAddresses>
                <add
baseAddress="net.tcp://localhost:8523/WCFTestService" />
            </baseAddresses>
        </host>
    </service>
</services>
```

9. In the Configuration Editor, select the **Bindings** node and then click the **New Binding Configuration** link.

10. In the **Create a New Binding** dialog box, select **netTcpBinding** and then click **OK**.
11. Set the **Name** attribute to **NetTcpBindingEndpointConfig** on the newly created binding configuration.
12. Click the **Security** tab and then set the **Mode** attribute to **Message**.
13. Verify that the **MessageClientCredentials** attribute is set to Windows.
14. In the **NetTcpBindingEndpoint** binding created above, set the **BindingConfiguration** to **NetTcpBindingEndpointConfig** by selecting it from the dropdown.
15. In the Configuration Editor dialog box, on the **File** menu, select **Save**.
16. In Visual Studio, verify your configuration, which should look as follows:

```
...
<bindings>
    <netTcpBinding>
        <binding name="NetTcpBindingEndpointConfig">
            <security mode="Message" />
        </binding>
    </netTcpBinding>
</bindings>
...
<services>
    <service
behaviorConfiguration="WCFHostService.MyServiceBehavior"
        name="WCFHostService.MyService">
        <endpoint address="" binding="netTcpBinding"

bindingConfiguration="NetTcpBindingEndpointConfig"
                  name="NetTcpBindingEndpoint"
contract="WCFHostService.IMyService">
            <identity>
                <dns value="localhost" />
            </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding"
bindingConfiguration=""
            contract="IMetadataExchange" />
        <host>
            <baseAddresses>
                <add
baseAddress="net.tcp://localhost:8523/WCFTestService" />
            </baseAddresses>
        </host>
    </service>
</services>
...
```

Step 5 – Configure the WCF Service to Publish Metadata

In this step, you configure your WCF service to publish metadata. Publishing the metadata will allow your client to add a reference to the WCF service.

1. In the Configuration Editor, expand the **Services** node and then expand the **WCFHostService.MyService** node.

2. Expand the **Endpoints** node, select the remaining **[Empty Name]** node, and then set the **Name** attribute to **mexTcpBindingEndpoint**.
3. Set the **Binding** attribute to **mexTcpBinding**.
4. In the Configuration Editor dialog box, on the **File** menu, select **Save**.
5. In Visual Studio, verify the configuration in your App.config file. The configuration should look as follows:

```
...
<services>
    <service
behaviorConfiguration="WCFHostService.MyServiceBehavior"
        name="WCFHostService.MyService">
        <endpoint address="" binding="netTcpBinding"

bindingConfiguration="NetTcpBindingEndpointConfig"
                  name="NetTcpBindingEndpoint"
contract="WCFHostService.IMyService">
            <identity>
                <dns value="localhost" />
            </identity>
        </endpoint>
        <endpoint address="mex" binding="mexTcpBinding"
bindingConfiguration=""
            name="mexTcpBindingEndpoint"
contract="IMetadataExchange" />
        <host>
            <baseAddresses>                <add
baseAddress="net.tcp://localhost:8523/WCFTestService" />
            </baseAddresses>
        </host>
    </service>
</services>
...
```

6. In the Configuration Editor, expand the **Advanced** node and then expand the **Service Behaviors** node.
7. Expand the **WCFHostService.MyServiceBehavior** node and then select the **serviceMetadata** node.
8. Set the **HttpGetEnabled** attribute to **False**.
9. In the Configuration Editor dialog box, on the **File** menu, select **Save**.
10. In Visual Studio, verify the configuration in your App.config file. The configuration should look as follows:

```
<behaviors>
    <serviceBehaviors>
        <behavior name="WCFHostService.MyServiceBehavior">
            <serviceMetadata httpGetEnabled="false" />
            <serviceDebug includeExceptionDetailInFaults="false"
/>
        </behavior>
    </serviceBehaviors>
</behaviors>
```

Step 6 – Install the Windows Service

In this step, you install the Windows service and run it from the Services console.

1. Rebuild the solution and open a Visual Studio command prompt.
2. Navigate to the bin directory of the project where **WCFServiceHost.exe** was copied.
3. On the command line, execute the following command to install the service:

```
Installutil WCFServiceHost.exe
```

4. After the service has installed successfully, open the services console by executing services.msc on the command line.
5. In the services console, search for the name of the service, **Service1**, and start it.

Note: If you have modified the service that is already installed, you can uninstall it by using following command:

```
Installutil /u WCFServiceHost.exe
```

Step 7 – Create a Test Client Application

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.
4. Right-click your client project and then click **Add Service Reference**.
5. In the **Add Service Reference** dialog box, set the **Service URI:** to **net.tcp://localhost:8523/WCFTestService** and then click **Go**.
6. Set the **Service reference name:** to WCFTestService and then click **OK**.

Step 8 – Test the Client and WCF Service

In this step, you use the test client to ensure that the WCF service is running properly.

1. In your Client project, drag a button control onto your form.
2. Double-click the button control to show the underlying code.
3. Create an instance of the proxy and call the **DoWork** method on your WCF service.
   When you call the service, your current user security context will automatically be passed to your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
```

```
{
      WCFTestService.MyServiceClient myService = new
                              WCFTestService.MyServiceClient();
      MessageBox.Show(myService.DoWork("Hello World!"));
      myService.Close();
}
```

4. Right-click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5.
   When you click the button on the form, it should display the message “Welcome Hello World!”.

Additional Resources

- For more information on security authentication best practices, see “Best Practices for Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms731059.aspx
- For additional information on message security, see “Message Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For more information on hosting in a Windows service, see “How to: Host a WCF Service in a Managed Windows Service.”
- For more information on WCF hosing considerations, see “Hosting Services” at http://msdn2.microsoft.com/en-us/library/ms730158.aspx
- For more information on netTcpBinding configuration options see “<netTcpBinding>” at http://msdn2.microsoft.com/en-us/library/ms731343.aspx

How To – Use netTcpBinding with Windows Authentication and Transport Security in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Windows Forms
- Microsoft Visual Studio® 2008

Summary

This How To article shows you how to use the **netTcpBinding** binding with Windows authentication and transport security. **netTcpBinding** is used for communicating with WCF clients in an intranet environment and provides transport security and Windows authentication by default. In this article, the WCF service is hosted in a Windows service.

Contents

Objectives

- Create a WCF service hosted in a Windows service.
- Expose the WCF service over **netTcpBinding** to WCF-enabled clients.
- Run the WCF service in the Network Service security context.
- Call the service from a Windows Forms test client.

Overview

Windows authentication is suited for scenarios in which your users have domain credentials. In the scenario described in this How To article, users are authenticated by Windows authentication. The scenario described in this How To article uses the **netTcpBinding** binding to expose a WCF service to WCF-enabled clients. The

netTcpBinding binding offers improved performance over an HTTP binding. Because Internet Information Services (IIS) 6.0 cannot host a TCP binding, in this scenario WCF is hosted in a Windows service. The WCF service with **netTcpBinding** can be consumed by a WCF-enabled .NET application through the use of a service reference. The Visual Studio service reference generates a proxy class to abstract the underlying message-based communication. WCF transport security is used to support a secure communication channel in a point-to-point scenario. In general, you should always use transport security unless you need the additional flexibility that message security affords you. For example, you would use message security for scenarios in which there are intermediaries who need to inspect and re-route the message.

In this How To article, you will create a Windows service to host your WCF service. You will then create sample WCF service in Visual Studio 2008 and configure the service to use **netTcpBinding** with transport security through the use of the WCF Configuration Editor. Next, you will configure **mexTcpBinding** so that the service can expose its metadata to clients from which they can generate a WCF proxy and call your service. Finally, you will create a test client to verify that the service is working properly.

Solution Summary

- **Binding** – By default, **netTcpBinding** offers improved performance over an HTTP binding and is the ideal choice for cross-machine communication between WCF clients and a WCF service, in an intranet environment.
- **Security mode** –Transport security is the default security mode for **netTcpBinding** and should be preferred over message security for better performance. If needed, message security can provide greater control over signing and encryption of the message.
- **Client authentication** – Because this binding is used inside an intranet, Windows is the default and recommended client authentication mechanism. The other options for this binding are None (for anonymous authentication) and Certificate.
- **Protection level** – It is recommended that you retain the default EncryptAndSign protection level for maximum transport security. This can be lowered to Sign for performance, but None is typically not recommended.
- **Hosting consideration** – For the purposes of this How To article, WCF is hosted in a Windows service. In general, **netTcpBinding** services can be hosted in a Windows service, in IIS 7.0 (not IIS 6.0 or lower), or in WAS, or it can be self-hosted. The choice should be based on the deployment requirements of the service.

Summary of Steps

- **Step 1 – Create a Windows Service**

- **Step 2 – Create a Sample WCF Service**
- **Step 3 – Modify the Windows Service to Host the WCF Service**
- **Step 4 – Configure the WCF Service to Use netTcpBinding with Transport Security**
- **Step 5 – Configure the WCF Service to Publish Metadata**
- **Step 6 – Install the Windows Service**
- **Step 7 – Create a Test Client Application**
- **Step 8 – Test the Client and WCF Service**

Step 1 – Create a Windows Service

In this step, you create a Windows service to host your WCF service.

1. In Visual Studio, on the menu, click **File** -> **New** -> **Project**.
2. In the **New Project** dialog box, in the **Project Types** section, select **Windows** under **Visual C#.**
3. In the **Templates** section, select **Windows Service**, specify the project location, and name it **"WCFServiceHost"**.
4. In the **Add a Project** dialog box, click **OK** to add a sample Windows service to the solution.
5. Right-click **Service1.cs** and then click **View Designer**.
6. Right-click the designer view and then click **Add Installer** to add the ProjectInstaller.cs file with two objects, **serviceProcessInstaller1** and **serviceInstaller1**.
7. In the designer view of ProjectInstaller.cs, right-click **serviceProcessInstaller1** and then click **Properties**.
8. In the properties section, set the **Account** attribute to **NetworkService**.
   This will run your Windows service under the Network Service account.

Step 2 – Create a Sample WCF Service

In this step, you add a WCF service to the Windows service that will host it.

1. Right-click the Widows Service project, click **Add**, and then click **New Item**.
2. In the **Add New Item** dialog box, select **WCF Service**.
3. Set the **Name** as MyService.cs and then click **Add**.
   Note that the configuration file, App.config, is automatically added.
4. In IMyService.cs, modify the **DoWork()** method signature to accept a string parameter and return a string data type as follows:

   ```
   string DoWork(string value);
   ```

5. In MyService.cs, modify the **DoWork()** method to accept a string parameter and return a string data type as follows:

```
public string DoWork(string value)
{
      return "Welcome " + value;
}
```

Step 3 – Modify the Windows Service to Host the WCF Service

In this step, you add code to the **OnStart()** and **OnStop()** methods to start and stop the WCF service inside the Windows service process.

1. In the Solution Explorer, right-click Service1.cs and then click **View Code**.
2. In the Service1.cs file, add a **using** statement as follows:

```
using System.ServiceModel;
```

3. Declare an internal static member of **ServiceHost** type as follows:

```
internal static ServiceHost myServiceHost = null;
```

4. To open the service host, add code to the **OnStart** method of a Windows service as follows:

```
protected override void OnStart(string[] args)
{
   if (myServiceHost != null)
   {
       myServiceHost.Close();
   }

   myServiceHost = new ServiceHost(typeof(MyService));
   myServiceHost.Open();
}
```

5. To close the service host, add code to the **OnStop** method of a Windows service as follows:

```
protected override void OnStop()
{
   if (myServiceHost != null)
   {
      myServiceHost.Close();
      myServiceHost = null;
   }
}
```

6. Build the solution and verify that your project produces “WCFServicecHost.exe” in your project \bin directory.

Step 4 – Configure the WCF Service to Use netTcpBinding with Transport Security

In this step, you configure your WCF service, “MyService”, to use **netTcpBinding**.

1. Right=click the App.config file and then click **Edit WCF Configuration**.
   If you do not see the Edit WCF Configuration option, on the **Tools** menu, click **WCF Service Configuration Editor**. Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu.
2. In the Configuration Editor, expand the **Services** node and then expand **WCFHostService.MyService**.
3. Select the **Host** node, select the default **BaseAddress** from the **Base addresses** section, and then click **Delete**.
4. In the **Base Address Editor** dialog box, click **New**, and then set the **Base address:** to "**net.tcp://localhost:8523/WCFTestService**".
   The port number 8523 is arbitrary and used for this example only. WCFTestService is also arbitrary and is used in this example to expose the endpoint.
5. Expand the **Endpoints** node, select the first **[Empty Name]** endpoint created, and then set the **Name** attribute to “**NetTcpBindingEndpoint**”.
6. Set the **Binding** attribute to **netTcpBinding** by choosing this option from the drop-down list.
7. In the Configuration Editor, on the **File** menu, click **Save**.
   Alternatively, press Ctrl + S.
8. In Visual Studio, verify your configuration in your App.config. The configuration should look as follows:

```
<services>
    <service behaviorConfiguration="WCFHostService.MyServiceBehavior"
        name="WCFHostService.MyService">
        <endpoint address=""
                  binding="netTcpBinding"
                  bindingConfiguration=""
                  name="NetTcpBindingEndpoint"
                  contract="WCFHostService.IMyService">
            <identity>
                <dns value="localhost" />
            </identity>
        </endpoint>

        <endpoint address="mex"
                  binding="mexHttpBinding"
                  contract="IMetadataExchange" />
        <host>
            <baseAddresses>
                <add
baseAddress="net.tcp://localhost:8523/WCFTestService" />
            </baseAddresses>
```

```
        </host>
    </service>
</services>
```

Note: Because netTcpBinding supports Windows authentication with transport security by default, you do not have to change any other configuration in the binding.

Step 5 – Configure the WCF Service to Publish Metadata

In this step, you configure your WCF service to publish metadata. Publishing the metadata will allow your client to add a reference to your WCF service.

1. In the Configuration Editor, expand the **Services** node, and then expand the **WCFHostService.MyService** node.
2. Expand the **Endpoints** node, select the second **[Empty Name]** endpoint created, and then set the **Name** attribute to "**MexTcpBindingEndpoint**".
3. Set the **Binding** attribute to mexTcpBinding by choosing this option from the drop-down list.
4. In the Configuration Editor, on the **File** menu, click **Save**.
   Alternatively, press Ctrl + S.
5. In Visual Studio, verify your configuration in App.config. The configuration should look as follows:

```
...
<services>
    <service
behaviorConfiguration="WCFHostService.MyServiceBehavior"
        name="WCFHostService.MyService">
        <endpoint address=""
                  binding="netTcpBinding"
                  bindingConfiguration=""
                  name="NetTcpBindingEndpoint"
                  contract="WCFHostService.IMyService">
            <identity>
                <dns value="localhost" />
            </identity>
        </endpoint>

        <endpoint address="mex"
                  binding="mexTcpBinding"
                  bindingConfiguration=""
                  name="MexTcpBidingEndpoint"
                  contract="IMetadataExchange" />
        <host>
            <baseAddresses>
                <add
baseAddress="net.tcp://localhost:8523/WCFTestService" />
            </baseAddresses>
        </host>
    </service>
</services>
```

```
...
```

6. In the Configuration Editor, expand the **Advanced** node, and then expand the **Service Behaviors** node.
7. Expand the **WCFHostService.MyServiceBehavior** node and then select the **serviceMetadata** node.
8. Set the **HttpGetEnabled** attribute to **False**.
9. In the Configuration Editor, on the **File** menu, click **Save**.
10. In Visual Studio, verify your configuration in App.config. The configuration should look as follows:

```
<behaviors>
    <serviceBehaviors>
        <behavior name="WCFHostService.MyServiceBehavior">
            <serviceMetadata httpGetEnabled="false" />
            <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
    </serviceBehaviors>
</behaviors>
```

Step 6 – Install the Windows Service

In this step, you install the Windows service and run it form the Services console.

1. Rebuild the solution and open a Visual Studio 2008 command prompt.
2. Navigate to the **\bin** directory of the project where WCFServiceHost.exe was created.
3. Execute the following command:

   `> Installutil WCFServiceHost.exe` to install the service.

4. If the service installs successfully, open the service console by typing **services.msc** in the Windows **Run** prompt.
5. Search for the name of the service, **Service1**, and then start it.

Note – If you modified the service that is already installed, you can uninstall it by using following command:

```
> Installutil /u WCFServiceHost.exe
```

Step 7 – Create a Test Client Application

In this step, you create a Windows Forms application named Test Client to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New ProjectB.**

2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK** to create a Windows Forms application.
4. Right-click your client project and then click **Add Service Reference**.
5. In the **Add Service Reference** dialog box, set the **Service URI:** to "net.tcp://localhost:8523/WCFTestService**"** and then click **Go**.
6. Change the **Service reference name:** to "**WCFTestService**" and then click **OK**.

Step 8 – Test the Client and WCF Service

In this step, you use the test client to ensure that the WCF service is running properly.

1. In your client project, drag a **Button** control onto your form.
2. Double-click the **Button** control to show the underlying code.
3. In the code behind the button click, create an instance of the proxy, and call the **DoWork()** method of your WCF service.
   When you call the service, your current user security context will automatically be passed to your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.MyServiceClient myService =
         new WCFTestService.MyServiceClient();
      MessageBox.Show(myService.DoWork("Hello World!"));
      myService.Close();
}
```

4. Right click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5.
   when you click the button on the form, the message "Welcome Hello World!" should appear.

Additional Resources

- For more information on security authentication best practices, see "Best Practices for Security in WCF" at http://msdn2.microsoft.com/en-us/library/ms731059.aspx
- For additional information on message security, see "Message Security in WCF" at http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- For more information on hosting in a Windows service, see the document "How To: Host WCF in a Windows Service."
- For more information on WCF hosting considerations, see "Hosting Services" at http://msdn2.microsoft.com/en-us/library/ms730158.aspx

- For more information on **netTcpBinding** configuration options see "<netTcpBinding>" at http://msdn2.microsoft.com/en-us/library/ms731343.aspx

How To: Use Protocol Transition for Impersonating and Delegating the Original Caller in WCF

Applies To

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008
- Microsoft Windows Server® 2003

Summary

This How To article walks you through the process of using protocol transition for impersonating and delegating the original caller. You will learn how to use the client certificate for authentication. You will then use the service for user (S4U) Kerberos extensions to create a Windows identity for the authenticated user, by using the user principal name (UPN) and impersonating and delegating the original caller.

Contents

Objectives

- Learn how to do protocol transition in WCF by using a client certificate.
- Learn how to configure the WCF process identity for protocol transition and constrained delegation.

Overview

In many situations—for example, if your users access a WCF service over the Internet—you cannot use Kerberos authentication because firewalls prevent the client computer from directly communicating with the domain controller. Instead, your application must authenticate the client by using another approach, such as username authentication, or client certificate authentication.

Windows Server 2003 includes a protocol transition feature that permits services to use a non-Windows authentication mechanism to authenticate users, while still using Kerberos authentication and delegation to access downstream network resources. This allows your application to access downstream servers that require Windows authentication, and allows you to use Windows auditing to track user access to back-end resources.

Note that impersonating a Windows identity to access downstream resources brings a number of advantages, but also some disadvantages. The advantages include the ability to use Windows auditing to track user access to back-end resources, and the ability to implement fine-grained access controls to resources (such as databases) on a per-user basis. The disadvantages include the additional administration required to administer fine-grained access controls, and reduced scalability. For many applications, the trusted subsystem model is appropriate; for example, where the WCF service authenticates the caller, but then uses a service identity to access downstream resources on behalf of the original caller. This results in reduced administration and improved scalability.

The use of protocol transition to access downstream resources relies on two extensions to the Kerberos protocol. Both extensions are implemented in Windows Server 2003. These extensions are:

- **Service-for-User-to-Self (S4U2Self)**, which allows you to obtain a Windows token for the client by supplying a UPN without a password.
- **Service-for-User-to-Proxy (S4U2Proxy)**, which allows an administrator to control exactly which downstream services can be accessed with the S4U2Self token.

Summary of Steps

- **Step 1 – Create a Sample WCF Service**
- **Step 2 – Configure wsHttpBinding with Certificate Authentication and Message Security**
- **Step 3 – Create and Install a Service Certificate**
- **Step 4 – Configure the Service Certificate for the WCF Service**
- **Step 5 – Impersonate the Original Caller in the WCF Service**
- **Step 6 – Configure the WCF Service Identity for Protocol Transition and Delegation**
- **Step 7 – Create a Test Client**
- **Step 8 – Add a WCF Service Reference to the Client**

- **Step 9 – Create and Install the Client Certificate for Authentication**
- **Step 10 – Configure the Client Certificate in the WCF Client Application**
- **Step 11 – Test the Client and WCF Service**

Step 1 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio.

1. In Visual Studio, on the menu, click **File** and then click **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http** and specify the virtual directory to be created in the **Path (**e.g., http://localhost/WCFTestService).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample WCF service.
4. Browse to your WCF service (i.e., http://localhost/WCFTestService/Service.svc). You should see details of your WCF service.

Step 2 – Configure wsHttpBinding with Certificate Authentication and Message Security

In this step, you configure the WCF service to use certificate authentication and message security.

1. Right-click the Web.config file of the WCF service, and then click **Edit WCF Configuration**.
2. In the Configuration Editor, in the **Configuration** section, expand **Service** and then expand **Endpoints**.
3. Select the first node **[Empty Name]** and set the **Name** attribute to **wsHttpEndpoint**. By default, the name will be empty because it is an optional attribute.
4. Click the **Identity** tab and then delete the **Dns** attribute value.
5. In the Configuration Editor, select the Bindings folder.
6. In the **Bindings** section, choose **New Binding Configuration**.
7. In the **Create a New Binding** dialog box, select **wsHttpBinding**.
8. Click **OK**.
9. Set the **Name** of the binding configuration to some logical and recognizable name; for example, **wsHttpEndpointBinding**.
10. Click the **Security** tab.
11. Make sure that the **Mode** attribute is set to **Message**, which is the default setting.
12. Set the **MessageClientCredentialType** to **Certificate** by selecting this option from the drop-down list.
13. In the **Configuration** section, select the **wsHttpEndpoint** node.
14. Set the **BindingConfiguration** attribute to **wsHttpEndpointBinding** by selecting this option from the drop-down list.
    This associates the binding configuration setting with the binding.
15. In the Configuration Editor, on the **File** menu, click **Save**.

16. In Visual Studio, open your configuration and comment out the identity element. It should look as follows:

```
<!--<identity>
  <dns value="" />
</identity>-->
```

17. In Visual Studio, verify your configuration. The configuration should look as follows:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security>
        <message clientCredentialType="Certificate" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
<services>
  <service behaviorConfiguration="ServiceBehavior" name="Service">
    <endpoint address="" binding="wsHttpBinding"
      bindingConfiguration="wsHttpEndpointBinding"
      name="wsHttpEndpoint" contract="IService">
      <!--<identity>
        <dns value="" />
      </identity>-->
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />
  </service>
</services>
…
```

Step 3 – Create and Install a Service Certificate

In this step, you create a temporary service certificate and install it in the local store. This certificate will be used for service authentication and to encrypt the message, thereby protecting any other sensitive data.

Creating and installing the certificate is outside the scope of this How To article. For detailed steps on how to do this, see “How To - Create and Install Temporary Certificates in WCF for Message Security During Development.”

Note:

- If you are running on Microsoft Windows® XP, give the certificate permissions for the ASPNET identity instead of the NT Authority\Network Service identity because the Internet Information Services (IIS) process runs under the ASPNET account in Windows XP.

- The temporary certificate should be used for development and testing purposes only. For actual production deployment, you will need to get a valid certificate from a certificate authority (CA).

Step 4 – Configure the Service Certificate for the WCF Service

In this step, you configure the WCF service to use the temporary certificate you created in the previous step.

1. In the Configuration Editor, expand the **Advanced** node, and then expand the **Service Behaviors** and **ServiceBehavior** nodes.
2. Click **Add**.
3. In the **Service Behavior Element Extensions** dialog box, select the **serviceCredentials** option and then click **Add**.
4. Expand the **serviceCredentials** node and then select the **serviceCertificate** node.
5. Set the **FindValue** attribute to the name of the service certificate that you created; for example, "CN=tempCertServer".
6. Leave the default settings for StoreLocation and StoreName.
7. In the Configuration Editor, on the **File** menu, click **Save**.
8. In Visual Studio, verify your configuration. The configuration should look as follows.

```
...
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceCredentials>
        <serviceCertificate findValue="CN=tempCertServer" />
      </serviceCredentials>
    </behavior>
  </serviceBehaviors>
</behaviors>
...
```

Step 5 – Impersonate the Original Caller in the WCF Service

Perform the following steps to retrieve the UPN from the client certificate, create the **WindowsIdentity** token, and impersonate the original caller.

1. Add **using** statements to add references to the relevant namespaces as follows:

```
using System.IdentityModel.Policy;
using System.IdentityModel.Claims;
using System.Security.Principal;
```

2. Extract the Subject name from the certificate, as shown in the following example. Note that the Subject name for the client certificate is the user's UPN – this is done consciously at the time of client certificate creation in order to simplify the process. Alternatively, you could have other extended attributes with the user UPN.

```
public string GetData(int value)
{

   AuthorizationContext authCon =
ServiceSecurityContext.Current.AuthorizationContext;
   X509CertificateClaimSet certClaims = null;
   foreach (ClaimSet cSet in authCon.ClaimSets)
   {
      certClaims = cSet as X509CertificateClaimSet;
      if (certClaims != null)
          break;
   }
   // As the subject name starts with "CN=" we are extracting the substring
   string userName = certClaims.X509Certificate.Subject.Substring(3);

   return string.Format("You entered: {0}", value);
}
```

3. Using the **WindowsIdentity** constructor, pass the UPN string as the parameter, get the **WindowsIdentity** token, and impersonate the original caller. If your WCF process identity is configured for protocol transition and trusted for delegation, you can access the remote resources as well.

```
public string GetData(int value)
{

   AuthorizationContext authCon =
ServiceSecurityContext.Current.AuthorizationContext;
   X509CertificateClaimSet certClaims = null;
   foreach (ClaimSet cSet in authCon.ClaimSets)
   {
      certClaims = cSet as X509CertificateClaimSet;
      if (certClaims != null)
          break;
   }
   // As the subject name starts with "CN=" we are extracting the substring
   string userName = certClaims.X509Certificate.Subject.Substring(3);

   WindowsIdentity winId = new WindowsIdentity(userName);
   using (winId.Impersonate())
   {
       // access the local resources on behalf of the original callers
       // Or access remote resources, like SQL database on remote machine
       // if configured for protocol transition and constrained delegation.
   }

   return string.Format("You entered: {0}", value);
}
```

Step 6 – Configure the WCF Service Identity for Protocol Transition and Constrained Delegation

In this step, you configure Active Directory to allow your WCF service to use protocol transition and constrained delegation to access a remote database server.

If your WCF service runs using the Network Service machine account, you must enable constrained delegation for your WCF server computer. However, if your WCF Service runs under a custom domain account, you must enable constrained delegation for the custom domain account.

Note: If you use a custom domain account for running your WCF service, create a service principal name (SPN) for your custom domain account. Kerberos requires an SPN in order to support mutual authentication.

To configure constrained delegation for the machine account

This procedure assumes that you are running your WCF service under the Network Service machine account.

1. On the domain controller, start the Microsoft Management Console (MMC) Active Directory Users and Computers snap-in.
2. In the left pane of the MMC snap-in, click the **Computers** node.
3. In the right pane, double-click your WCF server computer to display the **Properties** dialog box.
4. On the **Delegation** tab of the Properties window for the WCF server computer, **Do not trust the computer for delegation** is selected by default. To use constrained delegation, select **Trust this computer for delegation to specified services only**. You specify precisely which service or services can be accessed in the bottom pane.
5. Beneath **Trust this computer for delegation to specified services only**, select the option **Use any authentication protocol**.
6. Click the **Add** button to display the **Add Services** dialog box.
7. Click the **Users or computers** button.
8. In the **Select Users or Computers** dialog box, type the name of your database server computer if you are running SQL Server as System or Network Service.
   Alternatively, if you are running SQL Server by using a custom domain account, enter that account name instead and then click **OK**.
9. You will see all the SPNs configured for the selected user or computer account. To restrict access to SQL Server, select the **MSSQLSvc** service, and then click **OK**.

Note If you want to delegate to a file on a file share, you need to select the Common Internet File System (CIFS) service.

To configure constrained delegation for a custom domain account

This procedure assumes that you are running your Web application under a custom domain account.

1. Create an SPN for your custom domain account. Kerberos requires an SPN in order to support mutual authentication. To create an SPN for the domain account:
    a. Install the Windows Server 2003 Tools from the Windows Server 2003 CD.
    b. From a command prompt, run the **Setspn** tool twice from the **C:\Program Files\Support Tools** directory as shown below:

    setspn -A HTTP/*wcfservername domain\customAccountName*

    setspn -A HTTP/*wcfservername.fullyqualifieddomainname*

    domain\customAccountName

    Note You can only have a single SPN associated with any HTTP service (DNS) name, which means you cannot create SPNs for different service accounts mapped to the same HTTP server unless they are on different ports. The SPN can include a port number.

2. On the domain controller, start the Microsoft Management Console (MMC) Active Directory Users and Computers snap-in.
3. In the left pane of the MMC snap in, click the **Users** node.
4. In the right pane, double-click the user account you are using to run the WCF service. This displays the user account properties.
5. On the **Delegation** tab of the Properties window for the WCF server computer, **Do not trust the computer for delegation** is selected by default. To use constrained delegation, select **Trust this computer for delegation to specified services only**. You specify precisely which service or services can be accessed in the bottom pane.
6. Beneath **Trust this computer for delegation to specified services only**, select the option **Use any authentication protocol**.
7. Click the **Add** button to display the **Add Services** dialog box.
8. Click the **Users or computers** button.
9. In the **Select Users or Computers** dialog box, type the name of your database server computer if you are running SQL Server as System or Network Service. Alternatively, if you are running SQL Server by using a custom domain account, enter that account name instead and then click **OK**.
10. You will see all the SPNs configured for the selected user or computer account. To restrict access to SQL Server, select the **MSSQLSvc** service, and then click **OK**.

Step 7 – Create a Test Client

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.

Step 8 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service.

1. Right-click your client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF Service (e.g., http://localhost/WCFTestService/Service.svc) and then click **Go**.
3. In the **Web reference name** field, change ServiceReference1 to **WCFTestService**.
4. Click **Add Reference**.
   A reference to WCFTestService should appear beneath Web References in your client project.

Step 9 – Create and Install the Client Certificate for Authentication

In this step, you create a temporary client certificate by using the Root CA created as part of the Step 3, and install it in the local store. This certificate will be used for client authentication and to encrypt the message, thereby protecting any other sensitive data.

1. Copy the root CA certificate (RootCATest.cer) and privatekeyfile (RootCATest.pvk), created as part of Step 3, to the client machine.
2. Open a Visual Studio command prompt and browse to the location where you copied the root CA certificate and privatekeyfile.
3. Run following command for creating a certificate signed by the root CA certificate:

```
makecert -sk MyKeyName -iv RootCATest.pvk -n "CN=User@DomainName.com" -ic
RootCATest.cer -sr CurrentUser -ss my -sky signature -pe User1.cer
```

4. In the **Enter Private Key Password** dialog box, enter the password for the root CA privatekeyfile created as part of the Step 3 above, and then click **OK**.

For more information and detailed steps, see “How To - Create and Install Temporary Certificates in WCF for Message Security During Development.”

Step 10 – Configure the Client Certificate in the WCF Client Application

In this step, you configure the WCF client to use the temporary certificate you created in the previous step.

1. In your test client, right-click the App.config file and then click **Edit WCF Configuration**.

2. In the Configuration Editor, expand the **Advanced** node, select **Endpoint Behaviors**, and then select **New Endpoint Behavior Configuration**.
3. Click **Add**.
4. In the **Adding Behavior Element Extension Sections** dialog box, select **clientCredentials** and then click **Add**.
5. Expand the **clientCredentials** node, expand the **serviceCertificate** node, and then select **authentication** below this node.
6. Set the **CertificateValidationMode** to **PeerTrust** by choosing this option from the drop-down list.
7. Select the **clientCertificate** node, and then set the **FindValue** attribute to the subject name of the client certificate that you created and installed in Step 7; for example, "CN=User@DomainName.com".
8. Leave the default **StoreLocation** attribute set to **CurrentUser** as is.
9. In the Configuration Editor, expand the **Client** node, expand the **Endpoints** node, and then select the **WsHttpEndpoint** node.
10. Set the **BehaviorConfiguration** attribute to **NewBehavior** by choosing this option from the drop-down list.
    This is the endpoint behavior you just created.
11. In the Configuration Editor, on the **File** menu, click **Save**.
12. In Visual Studio, verify your configuration. The configuration should look as follows.

```
<system.serviceModel>
    <behaviors>
        <endpointBehaviors>
            <behavior name="NewBehavior">
                <clientCredentials>
                    <clientCertificate findValue="CN=tempCertClient"/>
                </clientCredentials>
            </behavior>
        </endpointBehaviors>
    </behaviors>
    ...
    <client>
        <endpoint address="http://<<service address>>"
            behaviorConfiguration="NewBehavior" binding="wsHttpBinding"
            bindingConfiguration="wsHttpEnpoint1"
contract="ServiceReference1.IService"
            name="wsHttpEnpoint">
            <identity>
                <certificate encodedValue="<<Encode Value>>" />
            </identity>
        </endpoint>
    </client>
</system.serviceModel>
```

Step 11 – Test the Client and WCF Service

In this step, you access the WCF service, pass the user credentials, and make sure that the username authentication works.

1. In your client project, drag a **Button** control onto your form.
2. Double-click the **Button** control to show the underlying code.
3. Create an instance of the proxy and call the **GetData** operation of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                  WCFTestService.ServiceClient();
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5.
   When you click the button on the form, the message "**You entered: 123**" should appear.

Additional Resources

- For more information on protocol transition and ASP.NET, see "How To: Use Protocol Transition and Constrained Delegation in ASP.NET 2.0" at http://msdn2.microsoft.com/en-us/library/ms998355.aspx
- For more information on S4U Kerberos extensions, see "Exploring S4U Kerberos Extensions in Windows Server 2003" at http://msdn2.microsoft.com/en-us/magazine/cc188757.aspx

How To – Use the SQL Server Role Provider with Username Authentication in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of using username authentication over **wsHttpBinding** binding to authenticate your users against a Microsoft SQL Server™ Role Provider. The article shows you how to configure the Role Provider, configure WCF, and test the service with a sample WCF client. Use of the SQL Server Role Provider requires that you first set up and use the SQL Server Membership Provider.

Contents

Objectives

- Configure the SQL Server Membership Provider.
- Configure the SQL Server Role Provider.
- Create a WCF service hosted in Microsoft Internet Information Services (IIS).
- Create and configure a certificate for the service.
- Expose the WCF service through **wsHttpBinding**.
- Call the service from a test client.

Overview

Username authentication is suited for scenarios in which your users do not have domain credentials. In the scenario described in this How To article, users are stored in SQL Server and are authenticated first against the SQL Server Membership Provider and then against the SQL Server Role Provider. The **wsHttpBinding** binding is used to provide support for message-based security, reliable messaging, and transactions, while also allowing the possibility that legacy clients can consume the service. WCF message security is used to support the scenario in which there may be intermediaries inspecting the message before final delivery. In general, you should always use transport security unless you need the additional flexibility that message security affords you.

In order to use the SQL Server Membership Provider, you will first create a user store and populate it with your users. You will then configure the membership store to allow the WCF service process identity to have access. You will set the **clientCredentialType** attribute to **UserName** on **wsHttpBinding** in order to configure the WCF service to use Username authentication. You will then install a certificate on the server and configure it for WCF so that messages sent between the client and server are encrypted. You will create a role store and populate it with your users and then configure the role store to grant access to the WCF process identity. You will use the **PrincipalPermissionAttribute** in your WCF service code to specify which roles are allowed to access specific operations in your WCF service. For test purposes, you will set the **revocationMode** attribute to **NoCheck** so that the temporary test certificate works properly.

Summary of Steps

- Step 1 – Create a WCF Service with Username Authentication Using the SQL Server Membership Provider
- Step 2 – Create a Role Store for SQL Server Role Provider
- Step 3 – Grant Access Permission to the WCF Service Process Identity
- Step 4 – Enable and Configure the Role Provider
- Step 5 – Create Roles and Assign Users
- Step 6 – Implement Declarative role-based security
- Step 7 – Create a Test Client
- Step 8 – Add WCF Service Reference to the Client
- Step 9 – Configure the Client to Set RevocationMode to NoCheck
- Step 10 – Test the Client and WCF Service

Step 1 – Create a WCF Service with Username Authentication Using the SQL Server Membership Provider

In this step, you create a WCF service with username authentication using the SQL Server Membership Provider and message security.

1. Create a user store for the SQL Server Membership Provider with the following command:

   ```
   aspnet_regsql -S .\SQLExpress -E -A m
   ```

2. Grant the WCF service process identity permission to the Aspnetdb database. You can accomplish this by creating a new SQL Server login for the WCF process identity (Network Service on Microsoft Windows Server® 2003 or ASPNET on Microsoft Windows® XP), create a new user in the Aspnetdb database, and then add the user to the aspnet_Membership_FullAccess database role.
3. Create a sample WCF service in Visual Studio 2008 by creating a new Web site project and selecting the **WCF Service** project template.
4. Configure the WCF service to use username authentication and message security by using the WCF Configuration Editor.
5. Configure the SQL Server Membership Provider to use username authentication by adding a connection string to the database in the service's web.config file and then adding a membership element to specify usage of the SQL Server Membership Provider.
6. Create and install a temporary certificate for the service.
7. Configure WCF to use the certificate by modifying the **Service Credentials** element in the WCF Configuration Editor.

For more information on these steps, see "How To – Use Username Authentication with the SQL Server Membership Provider and Message Security in WCF from Windows Forms" and follow steps 1 through 7.

Step 2 – Create a Role Store for the SQL Server Role Provider

The SQL Server Role Provider stores user information in a SQL Server database. You can create your SQL Server role store manually by using Aspnet_regsql.exe from the command line.

- From a Visual Studio 2008 command prompt, run the following command.

  ```
  aspnet_regsql -S .\SQLExpress -E -A r
  ```

  In this command:

  - **-S** – Specifies the server, which is (.\SQLExpress) in this example.
  - **-E** – Specifies to use Windows Authentication to connect to SQL Server.
  - **-A r** – Specifies to add only the Role Provider feature.
  - For a complete list of the commands, run Aspnet_regsql /?

Step 3 – Grant Access Permission to the WCF Service Process Identity

In Step 1, you granted the WCF service process identity access to the aspnetdb database. In this step, you add the Network Service database user to the aspnet_Roles_FullAccess role. You can do this either by using Enterprise Manager or by running the following script in SQL Query Analyzer:

```
-- Add user to database role
USE aspnetdb
GO
sp_addrolemember 'aspnet_Roles_FullAccess', 'Network Service'
```

Note:

- If you are running on Windows XP, add the ASPNET database user instead of Network Service because the IIS process runs under the ASPNET account in Windows XP.
- If you do not have Enterprise Manager or Query Analyzer, you can use Microsoft SQL Server Management Studio Express, available at http://www.microsoft.com/downloads/details.aspx?FamilyId=C243A5AE-4BD1-4E3D-94B8-5A0F62BF7796&displaylang=en

Step 4 – Enable and Configure the Role Provider

In this step, you configure the use of the SQL Server Role Provider in your WCF service.

1. In the web.config file, verify that you have a connection string similar to the following:

```
<connectionStrings>
  <add name="MyLocalSQLServer"
       connectionString="Initial Catalog=aspnetdb;
      data source=.\sqlexpress;Integrated Security=SSPI;" />
</connectionStrings>
```

2. Add a <roleManager> element inside the <system.web> element as shown in the following example. Note the use of the <clear/> element, which prevents the default provider from being loaded and then never used.

```
...
<system.web>
  <roleManager enabled="true" defaultProvider="MySqlRoleProvider"
>
    <providers>
      <clear/>
      <add name="MySqlRoleProvider"
           connectionStringName="MyLocalSQLServer"
           applicationName="MyAppName"
```

```
            type="System.Web.Security.SqlRoleProvider" />
    </providers>
  </roleManager>
</system.web>
...
```

3. Save the Web.Config file; otherwise the changes might get lost during execution of the following steps.
4. Right-click the Web.config file of the WCF service and then click **Edit WCF Configuration**.
   If you do not see the Edit WCF Configuration option, on the **Tools** menu, select **WCF Service Configuration Editor**. Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu.
5. In the Configuration Editor, expand the **Advanced** node, and then expand the Service Behaviors folder.
6. Select the default behavior, **ServiceBehavior**.
7. In the **Behavior: ServiceBehavior** section, click **Add**.
8. In the **Adding Behavior Element Extension Sections** dialog box, select **serviceAuthorization** and then click **Add**.
9. In the **Configuration** section, under **Service Behaviors**, select the **serviceAuthorization** option.
10. Set the **principalPermissionMode** attribute to **UseAspNetRoles** by choosing this option from the drop-down list.
11. Set the **roleProviderName** attribute to **MySqlRoleProvider**, which you created above.
12. In the **Configuration Editor** dialog box, on the **File** menu, select **Save**.
13. In Visual Studio, verify your configuration, which should look as follows:

```
….
<behavior name="ServiceBehavior">
  <serviceMetadata httpGetEnabled="true" />
    <serviceDebug includeExceptionDetailInFaults="false" />
       ….
       <serviceAuthorization
principalPermissionMode="UseAspNetRoles"
            roleProviderName="MySqlRoleProvider" />
       ….
 </behavior>
….
```

Step 5 – Create Roles and Assign Users

In this step, you create roles for your application and assign users to those roles by using the ASP.NET Web Site Configuration Tool.

1. In the Solution Explorer, select the WCF service project, and then on the **Website** menu, select **ASP.NET Configuration**.

2. On the ASP.NET Web Site Administration Tool page, click the **Security** tab, and then click the **Select authentication type** link.
3. On the page that appears, select the **From the internet** radio button and then click **Done**.
4. Click the **Create user** link.
5. On the Create User page, enter the details of the user you want to create in the SQL store, and then click **Create User**. If successful, a new user will be created.
6. Click the **Create or Manage roles** link.
7. Enner the New role name – for example, “Managers” – and then click **Add Role**. If successful, a new role will be created.
8. On the Roles creation page, click the **Manage** link, choose the user created in the previous steps, and assign this user to the role by selecting the **User Is In Role** check box.

Step 6 – Implement Declarative Role-based Security

In this step, you provide authorized access to the **GetData** method only for users in the Managers role.

1. Open the **Service.cs** file and add the following statement for using the **System.Security.Permissions** namespace:

```
using System.Security.Permissions;
```

2. Add the **PrincipalPermissionAttribute** to authorize users in the Managers role, with the **SecurityAction** as **Demand** to the **GetData** method.

```
[PrincipalPermission(SecurityAction.Demand, Role="Managers")]
public string GetData(int value)
{
      return string.Format("You entered: {0}", value);
}
```

Step 7 – Create a Test Client

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Application**.
3. In the **Name** field, type **Test Client** and then click **OK** to create a Windows Forms application.

Step 8 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF Service.

1. Right-click your Client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF service – for example, http://localhost/WCFTestService/Service.svc – and then click **Go**
3. In the **Namespace** field, change ServiceReference1 to **WCFTestService** and then click **OK**.
   A reference to WCFTestService should now appear beneath **Service References** in your Client project.

Step 9 – Configure the Client to Set RevocationMode to NoCheck

This step is required because you installed a temporary service certificate in Step 1.

1. Right-click the client configuration (App.config) file and then click **Edit WCF Configuration**.
2. In the Configuration Editor, expand the **Advanced** node and then select **New Endpoint Behavior Configuration**.
3. Click **Add**.
4. In the **Adding Behavior Element Extension Sections** dialog box, select **clientCredentials** and then click **Add**.
5. Expand the **clientCredentials** node, expand the **serviceCertificate** node, and then select **Authentication** below this node.
6. Set the **RevocationMode** attribute to **NoCheck** by choosing this option from the drop-down list.
7. In the Configuration Editor, expand the **Client** node, expand the **Endpoints** node, and then select the **WsHttpEndpoint** node.
8. Set the **BehaviorConfiguration** attribute to **NewBehavior** by choosing this option from the drop-down list.
   This is the endpoint behavior you just created.
9. In the **Configuration Editor** dialog box, on the **File** menu, select **Save**.
10. In Visual Studio, verify your configuration, which should look as follows:

```
…
<behaviors>
    <endpointBehaviors>
        <behavior name="NewBehavior">
            <clientCredentials>
                <serviceCertificate>
                    <authentication revocationMode="NoCheck" />
                </serviceCertificate>
            </clientCredentials>
        </behavior>
    </endpointBehaviors>
</behaviors>
<client>
    <endpoint address="http://<<fully qualified machine
name>>/WCFTestService/Service.svc"
        behaviorConfiguration="NewBehavior"
binding="wsHttpBinding"
```

```
        bindingConfiguration="wsHttpEndpoint"
contract="WCFTestService.IService"
        name="wsHttpEndpoint">
        <identity>
            <certificate encodedValue="SomeEncodeValue" />
        </identity>
    </endpoint>
</client>
…
```

Important: This should be done in development only, when using the makecert utility for creating the certificates. In a real-world production environment, you should not overwrite the **RevocationMode** settings.

Note: If your client application is on a separate machine, you will need to install the Root Authority certificate created in Step 6 on your client machine as well.

Step 10 – Test the Client and WCF Service

In this step, you access the WCF service as a legacy ASMX Web Service and make sure that it works.

1. In your Client project, drag a button control onto your Form.
2. Double-click the button control to show the underlying code.
3. In the code behind the button click, create an instance of the proxy; pass the credentials of a user with the Managers role created in previous steps, and then call the **GetData** operation of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                  WCFTestService.ServiceClient();
      //pass the credentials of a user in Manager's role
      myService.ClientCredentials.UserName.UserName = "username";
      myService.ClientCredentials.UserName.Password = "p@ssw0rd";
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the Client project and then click **Set as Startup Project**.
5. Run the Client application by pressing F5 or Ctrl+F5. When you click the button on the form, then a message "You entered: 123" should appear.
6. Test the application by passing the credentials of a user belonging to a different role (e.g., Employee) and you should receive the security exception Access Denied.
   This is because the **GetData** operation can be accessed only by the users who belong to the Managers role.

Additional Resources

- For more information on how to work with the ASP.NET Role Provider, see “How to: Use the ASP.NET Role Provider with a Service” at http://msdn2.microsoft.com/en-us/library/aa702542.aspx
- For more information on how to work with the ASP.NET Role Manager, see “How To: Use Role Manager in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998314.aspx
- For more information on how to work with the ASP.NET Membership Provider, see “How to: Use the ASP.NET Membership Provider” at http://msdn2.microsoft.com/en-us/library/ms731049.aspx
- For more information on how to work with temporary certificates, see “How to: Create Temporary Certificates for Use During Development” at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on how to view certificates with the Microsoft Management Console (MMC) snap in, see “How to: View Certificates with the MMC Snap-in” at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see “Differences Between Service Certificate Validation Done by Internet Explorer and WCF” at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see “Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security” at http://msdn2.microsoft.com/en-us/library/aa702579.aspx

How To – Use the SQL Server Role Provider with Windows Authentication in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008
- SQL Server

Summary

This How To article walks you through the process of using Windows Authentication over **wsHttpBinding** binding to authenticate your users against a Microsoft SQL Server™ Role Provider. The article shows you how to configure the Role Provider, configure WCF, and test the service with a sample WCF client. Use of the SQL Server Role Provider requires that you first set up and use the SQL Server Membership Provider.

Contents

Objectives

- Configure the SQL Server Role Provider to use Microsoft Windows® accounts for authorizing users of the service.
- Create a WCF service hosted in Microsoft Internet Information Services (IIS).
- Expose the WCF service through **netTcpBinding**.
- Call the service from a test client.

Overview

Windows authentication is suited for scenarios in which your users have domain credentials. In the scenario described in this How To article, users are authenticated

against their Windows domain account and authorized against roles in the SQL Server Role Provider. The **netTcpBinding** binding offers improved performance over an HTTP binding. Because IIS 6.0 cannot host a TCP binding, the scenario described in this How To article instead hosts WCF in a Windows service. The WCF service with **netTcpBinding** can be consumed by a WCF-enabled .NET application through the use of a service reference. WCF transport security is used to support a secure communication channel in a point-to-point scenario. In general, you should always use transport security unless you need the additional flexibility that message security affords you. For example, you would use message security for scenarios in which there are intermediaries who need to inspect and re-route the message.

You will first create a new WCF service and set the **clientCredentialType** attribute to **Windows** on the **netTcpBinding** in order to configure the WCF service to use Windows Authentication. You will then create a new Windows service and configure it to host your WCF service. Next, you will install a certificate on the server and configure it for WCF so that messages sent between the client and server are encrypted. You will create a role store, populate it with roles, and map Windows accounts to these roles. You will then configure the role store to grant access to the WCF process identity. Finally, you will use the **PrincipalPermissionAttribute** in your WCF service code to specify which roles are allowed to access specific operations in your WCF service.

Summary of Steps

- Step 1 – Create a WCF Service with Windows Authentication
- Step 2 – Create a Role Store for the SQL Server Role Provider
- Step 3 – Grant Access Permission to the WCF Service Process Identity
- Step 4 – Enable and Configure the Role Provider
- Step 5 – Create and Assign Roles to Windows Accounts
- Step 6 – Implement Declarative Role-based Security
- Step 7 – Create a Test Client
- Step 8 – Add a WCF Service Reference to the Client
- Step 9 – Test the Client and WCF Service

Step 1 – Create a WCF Service with Windows Authentication

In this step, you create a WCF service using **netTcpBinding** with Windows Authentication and WCF transport security.

1. In Visual Studio 2008, create a sample Windows service by creating a project and selecting the Windows Service project template. Add an installer to the Windows service project so that it can be installed on the host machine.
2. Create a sample WCF service in Visual Studio 2008 by creating a new Web site project and selecting the **WCF Service** project template.

3. Modify the Windows service to host the WCF service by overriding the **OnStart()** and **OnStop()** methods to start and stop the WCF service within the Windows service.
4. Configure the WCF service to use **netTcpBinding** with transport security by using the WCF Configuration Editor.
5. Add a **mexHttpBinding** binding to the WCF service so that it can publish metadata.
   This interface will allow client applications to generate a proxy from the service definition.
6. Install the Windows service by calling the installer from the command line using installutil.exe.

For more information on these steps, see "How To - Use netTcpBinding with Windows Authentication and Transport Security in WCF from Windows Forms" and follow steps 1 through 6.

Step 2 – Create a Role Store for the SQL Server Role Provider

The SQL Server Role Provider stores user information in a SQL Server database. You can create your SQL Server role store manually by using Aspnet_regsql.exe from the command line.

- From a Visual Studio 2008 command prompt, run the following command.

  ```
  aspnet_regsql -S .\SQLExpress -E -A r
  ```

  In this command:

  - **-S** – Specifies the server, which is (.\SQLExpress) in this example.
  - **-E** – Specifies to use Windows Authentication to connect to SQL Server.
  - -**A r** – Specifies to add only the Role Provider feature.
  - For a complete list of the commands, run **Aspnet_regsql /?**

Step 3 – Grant Access Permission to the WCF Service Process Identity

Your WCF service process identity requires access to the Aspnetdb database. If you host the WCF service in Microsoft Internet Information Services (IIS) 6.0 on Microsoft Windows Server® 2003, the NT AUTHORITY\Network Service account is used by default to run WCF Service.

1. Create a SQL Server login for NT AUTHORITY\Network Service.
2. Grant the login access to the Aspnetdb database by creating a database user.

3. Add the user to the **aspnet_Roles_FullAccess** database role.

You can perform these steps by using Enterprise Manager or by running the following script in SQL Query Analyzer:

```
-- Create a SQL Server login for the Network Service account
sp_grantlogin 'NT AUTHORITY\Network Service'

-- Grant the login access to the roles database
USE aspnetdb
GO
sp_grantdbaccess 'NT AUTHORITY\Network Service', 'Network Service'

-- Add user to database role
USE aspnetdb
GO
sp_addrolemember 'aspnet_Roles_FullAccess', 'Network Service'
```

Note:

- If you are running on Microsoft Windows® XP, add the ASPNET database user instead of Network Service because the IIS process runs under the ASPNET account in Windows XP.
- If you do not have Enterprise Manager or Query Analyzer, you can use Microsoft SQL Server Management Studio Express, available at http://www.microsoft.com/downloads/details.aspx?FamilyID=c243a5ae-4bd1-4e3d-94b8-5a0f62bf7796&displaylang=en

Step 4 – Enable and Configure the Role Provider

In this step, you configure the use of the SQL Server Role Provider in your WCF service.

1. In the web.config file, verify that you have a connection string similar to the following:

```
<connectionStrings>
  <add name="MyLocalSQLServer"
       connectionString="Initial Catalog=aspnetdb;
      data source=.\sqlexpress;Integrated Security=SSPI;" />
</connectionStrings>
```

2. Add a <**roleManager**> element inside the **<system.web>** element as shown in the following example. Note the use of the <**clear/**> element, which prevents the default provider from being loaded and then never used.

```
...
<system.web>
  <roleManager enabled="true" defaultProvider="MySqlRoleProvider"
```

```
        >
            <providers>
              <clear/>
              <add name="MySqlRoleProvider"
                   connectionStringName="MyLocalSQLServer"
                   applicationName="MyAppName"
                   type="System.Web.Security.SqlRoleProvider" />
            </providers>
          </roleManager>
        </system.web>
        ...
```

3. Save the Web.Config file; otherwise the changes might get lost during execution of the following steps.
4. Right-click the Web.config file of the WCF service and then click **Edit WCF Configuration**.
   If you do not see the **Edit WCF Configuration** option, click the **Tools** menu and select **WCF Service Configuration Editor**. Close the **WCF Service Configuration Editor** tool that appears. The option should now appear on the web.config context menu.
5. In the Configuration Editor, expand the **Advanced** node, and then expand the **Service Behaviors** folder.
6. Select the default behavior "**ServiceBehavior**".
7. In the **Behavior: ServiceBehavior** section, click the **Add**.
8. In the **Adding Behavior Element Extension Sections** dialog box select **serviceAuthorization** and then click **Add**.
9. In the **Configuration** section, under **Service Behaviors**, select **serviceAuthorization**.
10. Set the **principalPermissionMode** attribute to **UseAspNetRoles by** choosing this option from the drop-down list.
11. Set the **roleProviderName** attribute to "**MySqlRoleProvider**", which you created above.
12. In the **Configuration Editor** dialog box, on the **File** menu, click **Save**.
13. In Visual Studio, verify your configuration, which should look as follows.

```
….
<behavior name="ServiceBehavior">
  <serviceMetadata httpGetEnabled="true" />
    <serviceDebug includeExceptionDetailInFaults="false" />
       ….
       <serviceAuthorization principalPermissionMode="UseAspNetRoles"
            roleProviderName="MySqlRoleProvider" />
       ….
 </behavior>
….
```

Step 5 – Create and Assign Roles to Windows Accounts

In this step, you create roles for your application and assign users to those roles by executing SQL scripts to add them to the database directly.

1. Create a new role, Managers, for your application.
2. Add an existing Windows user to the Managers role.

You can perform these steps by using Enterprise Manager or by running the following script in SQL Query Analyzer.

```
USE aspnetdb
GO

-- Create a new role, called Managers
EXEC aspnet_Roles_CreateRole 'MyAppName', 'Managers'

-- Assign a windows user to the Managers role
-- parameters <<Application name>>, <<User Name>>, <<Role Name>>,
<<DateTime>>
EXEC aspnet_UsersInRoles_AddUsersToRoles 'MyAppName',
'Domain\userName', 'Managers', 8
```

Important: The application name should be the same name that is specified in the Role Provider configuration.

Note: If you do not have Enterprise Manager or Query Analyzer, you can use Microsoft SQL Server Management Studio Express, available at http://www.microsoft.com/downloads/details.aspx?FamilyID=c243a5ae-4bd1-4e3d-94b8-5a0f62bf7796&displaylang=en

Step 6 – Implement Declarative Role-based Security

In this step, you provide authorized access to the **GetData** method only for users in the Managers role.

1. Open the Service.cs file and add a statement for using the **System.Security.Permissions** namespace:

   ```
   using System.Security.Permissions;
   ```

2. Add the **PrincipalPermissionAttribute** attribute to authorize users in the **Managers** role with the **SecurityAction** as **Demand** to the **GetData** method:

   ```
   [PrincipalPermission(SecurityAction.Demand, Role="Managers")]
   public string GetData(int value)
   {
   ```

```
        return string.Format("You entered: {0}", value);
    }
```

Step 7 – Create a Test Client

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, the **Templates** section, select **Windows Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.
   A Windows Forms application is created.

Step 8 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service.

1. Right-click your Client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF service – for example, http://localhost/WCFTestService/Service.svc – and then click **Go**
3. In the **Namespace** field, change ServiceReference1 to **WCFTestService** and then click **OK**.
   A reference to WCFTestService should now appear beneath Service References in your Client project.

Step 9 – Test the Client and WCF Service

In this step, you access the WCF service and make sure that it authorizes the users correctly.

1. In your Client project, drag a button control onto your form.
2. Double-click the button control to show the underlying code.
3. In the code behind the button click, create an instance of the proxy, pass the credentials of a user with **Managers** role created in step 10, and call the **GetData** operation of your WCF Service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                  WCFTestService.ServiceClient();
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the Client project and then click **Set as Startup Project**.
5. Run the Client application by pressing F5 or Ctrl+F5. When you click the button on the form, the message "**You entered: 123**" should appear.
6. Test the application by passing the credentials of a user belonging to a different role (e.g., **Employee**) and you should receive the security exception **Access**

Denied. This is because the **GetData** operation can be accessed only by users who belong to **Managers** role.

Additional Resources

- For more information on how to work with the ASP.NET Role Provider, see "How to: Use the ASP.NET Role Provider with a Service" at http://msdn2.microsoft.com/en-us/library/aa702542.aspx
- For more information on how to work with the ASP.NET Role Manager, see "How To: Use Role Manager in ASP.NET 2.0" at http://msdn2.microsoft.com/en-us/library/ms998314.aspx
- For more information on how to work with the ASP.NET Membership Provider, see "How to: Use the ASP.NET Membership Provider" at http://msdn2.microsoft.com/en-us/library/ms731049.aspx
- For more information on how to work with temporary certificates, see "How to: Create Temporary Certificates for Use During Development" at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on how to view certificates with the Microsoft Management Console (MMC) snap in, see "How to: View Certificates with the MMC Snap-in" at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see "Differences Between Service Certificate Validation Done by Internet Explorer and WCF" at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see "Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security" at http://msdn2.microsoft.com/en-us/library/aa702579.aspx

How To – Use Username Authentication with the SQL Server Membership Provider and Message Security in WCF from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008
- Microsoft SQL Server ®

Summary

This How To article walks you through the process of using username authentication over **wsHttpBinding** to authenticate your users against a Microsoft SQL Server Membership Provider. The article shows you how to configure the Membership Provider, configure WCF, create and install the necessary certificate, and test the service with a sample WCF client.

Contents

Objectives

- Configure the SQL Server Membership Provider.
- Create a WCF service hosted in Microsoft Internet Information Services (IIS).
- Create and configure a certificate for the service.
- Call the service from a test client.

Overview

Username authentication is suited for scenarios in which your users do not have domain credentials. In the scenario described in this How To article, users are stored in SQL Server and are authenticated against the SQL Server Membership Provider, an identity management system that uses forms authentication. The **wsHttpBinding** binding is used in order to provide support for message-based security, reliable messaging, and transactions, while also allowing the possibility that legacy clients can consume the service. WCF message security is used to support the scenario in which there may be intermediaries inspecting the message before final delivery. In general, you should always use transport security unless you need the additional flexibility that message security affords you.

In order to use the SQL Server Membership Provider, you will first create a user store and populate it with your users. You will then configure the store to allow the WCF service access to authenticate users. You will set the **clientCredentialType** attribute to **UserName** on the **wsHttpBinding** binding in order to configure the WCF service to use UserName authentication. You will then install a certificate on the server and configure it for WCF so that messages sent between client and server are encrypted. For test purposes, you will set the **revocationMode** attribute to **NoCheck** so that the temporary test certificate works properly.

Summary of Steps

- Step 1 – Create a User Store for SQL Server Membership Provider
- Step 2 – Grant Access Permission to the WCF Service Process Identity
- Step 3 – Create a Sample WCF Service
- Step 4 – Configure wsHttpBinding with Username Authentication and Message Security
- Step 5 – Configure the Membership Provider for Username Authentication
- Step 6 – Create and Install a Service Certificate
- Step 7 – Configure the Service Certificate for WCF
- Step 8 – Create a User in the User Store
- Step 9 – Create a Test Client
- Step 10 – Add a WCF Service Reference to the Client
- Step 11 – Test the Client and WCF Service

Step 1 – Create a User Store for SQL Membership Provider

The SQL Server Membership Provider stores user information in a SQL Server database. You can create your SQL Server user store manually by using Aspnet_regsql.exe from the command line.

From a Visual Studio 2008 command prompt, run the following command:

```
aspnet_regsql -S .\SQLExpress -E -A m
```

In this command:

- **-S** specifies the server, which is **(.\SQLExpress)** in this example.
- **-E** specifies to use Windows Authentication to connect to SQL Server.
- -**A m** specifies to add only the membership feature. For simple authentication against a SQL Server user store, only the membership feature is required.
- For a complete list of the commands, run **Aspnet_regsql /?**

Step 2 – Grant Access Permission to the WCF Service Process Identity

Your WCF service process identity requires access to the Aspnetdb database. If you host the WCF Service in Internet Information Services (IIS) 6.0 on Microsoft Windows Server® 2003, the NT AUTHORITY\Network Service account is used by default to run the WCF service.

To grant database access

1. Create a SQL Server login for NT AUTHORITY\Network Service.
2. Grant the login access to the Aspnetdb database by creating a database user.
3. Add the user to the **aspnet_Membership_FullAccess** database role.

You can perform these steps by using the SQL Server Enterprise Manager, or you can run the following script in SQL Query Analyzer.

```
-- Create a SQL Server login for the Network Service account
sp_grantlogin 'NT AUTHORITY\Network Service'

-- Grant the login access to the membership database
USE aspnetdb
GO
sp_grantdbaccess 'NT AUTHORITY\Network Service', 'Network Service'

-- Add user to database role
USE aspnetdb
GO
sp_addrolemember 'aspnet_Membership_FullAccess', 'Network Service'
```

Note:

- If you are running on Microsoft Windows® XP, create a SQL Server login for the ASPNET identity instead of the NT Authority\Network Service identity, as IIS process runs under the ASPNET account in Windows XP.
- If you do not have Enterprise Manager or Query Analyzer, you can use Microsoft SQL Server Management Studio Express (SSMSE), available at http://www.microsoft.com/downloads/details.aspx?FamilyID=c243a5ae-4bd1-4e3d-94b8-5a0f62bf7796&displaylang=en

Step 3 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio.

1. In Visual Studio select **File** -> **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http** and specify the virtual directory to be created in the **Path (**e.g., http://localhost/WCFTestService).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample WCF service.
4. Browse to your WCF service (i.e., http://localhost/WCFTestService/Service.svc). You should see details of your WCF service.

Step 4 – Configure wsHttpBinding with Username Authentication and Message Security

In this step, you configure the WCF service to use Username authentication and message security.

1. In the Solution Explorer, right-click the Web.config file of the WCF service and choose the **Edit WCF Configuration** option.
2. If you do not see the **Edit WCF Configuration** option, click the **Tools** menu and select **WCF Service Configuration Editor**. Close the **WCF Service Configuration Editor** tool that appears. The option should now appear on the web.config context menu.
3. In the configuration editor, in the **Configuration** section, expand **Service** and then expand **Endpoints**.
4. Select the first node **[Empty Name].** Set the **name** attribute to **wsHttpEndpoint**. By default, the name field will be empty because it is an optional attribute.
5. Click the **Identity** tab and then delete the **Dns** attribute value.
6. In the configuration editor, select the **Bindings** folder.
7. In the **Bindings** section, choose **New Binding Configuration.**
8. In the **Create a New Binding** dialog box, select **wsHttpBinding**.
9. Click **OK**.
10. Set the **Name** of the binding configuration to some logical and recognizable name; for example, **wsHttpEndpointBinding**.
11. Click the **Security** tab.
12. Make sure that the **Mode** attribute is set to **Message**, which is the default setting.
13. Set the **MessageClientCredentialType** to the **Username** option by selecting this option from the drop-down list.
14. In the **Configuration** section, select the **wsHttpEndpoint** node.
15. Set the **BindingConfiguration** attribute to **wsHttpEndpointBinding** by selecting this option from the drop-down list.
    This associates the binding configuration setting with the binding.

16. In the configuration editor dialog box, on the **File** menu, select **Save**.
17. In Visual Studio, open your configuration and comment out the identity element. It should look as follows:

```
<!--<identity>
  <dns value="" />
</identity>-->
```

18. In Visual Studio, verify your configuration. The configuration should look as follows:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security>
        <message clientCredentialType="UserName" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
<services>
  <service behaviorConfiguration="ServiceBehavior"
name="Service">
    <endpoint address="" binding="wsHttpBinding"
      bindingConfiguration="wsHttpEndpointBinding"
      name="wsHttpEndpoint" contract="IService">
      <!--<identity>
        <dns value="" />
      </identity>-->
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />
  </service>
</services>
…
```

Step 5 – Configure Membership Provider for Username Authentication

In this step, you configure the SQL Server Membership Provider to use Username authentication.

1. In the web.config file, replace the existing single <connectionStrings/> element with the following to point to your membership database.

```
<connectionStrings>
  <add name="MyLocalSQLServer"
       connectionString="Initial Catalog=aspnetdb;
      data source=.\sqlexpress;Integrated Security=SSPI;" />
</connectionStrings>
```

2. Add a <**membership**> element inside the **<system.web>** element as shown in the following example. Note the use of the <**clear/**> element prevents the default provider from being loaded and then never used.

```
...
<system.web>
  ...
  <membership defaultProvider="MySqlMembershipProvider" >
    <providers>
      <clear/>
      <add name="MySqlMembershipProvider"
           connectionStringName="MyLocalSQLServer"
           applicationName="MyAppName"
           type="System.Web.Security.SqlMembershipProvider" />
    </providers>
  </membership>
</system.web>
...
```

3. Save the Web.Config file, to ensure that the changes do not get lost during the following steps.
4. In the configuration editor, expand the **Advanced** node, and then expand the Service Behaviors folder.
5. Select the default behavior that was created. Its name will be ServiceBehavior.
6. In the **Behavior: ServiceBehavior** section, click **Add**.
7. In the **Adding Behavior Element Extension Sections** dialog box, select **serviceCredentials** and then click **Add**.
8. In the **Configuration** section, and then under **Service Behaviors**, select the **serviceCredentials** option.
9. Set the **UsernamePasswordValidationMode** attribute to **MembershipProvider** by choosing this option from the drop-down list.
10. Set the **MembershipProviderName** attribute to **MySqlMembershipProvider**.
11. In the configuration editor dialog box, on the **File** menu, select **Save**.
12. In Visual Studio, verify your configuration. The configuration should look as follows:

```
…
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceCredentials>
        <userNameAuthentication
userNamePasswordValidationMode="MembershipProvider"
          membershipProviderName="MySqlMembershipProvider" />
      </serviceCredentials>
    </behavior>
  </serviceBehaviors>
</behaviors>
```

...

Step 6 – Create and Install a Service Certificate

In this step, you create a temporary Service Certificate and install it in the local store. This certificate will be used to encrypt the message, protecting the username and password as well as any other sensitive data.

Creating and installing the certificate is outside the scope of this How To article. For detailed steps on how to do this, see "How To - Create and Install Temporary Certificates in WCF for Message Security During Development."

Note:

- If you are running on Windows XP, give the certificate permissions for the ASPNET identity instead of the NT Authority\Network Service identity because the IIS process runs under the ASPNET account in Windows XP.
- Temp certificate should be used for development and testing purposes only. For actual production deployment, you will need to get a valid certificate from a certificate authority (CA).

Step 7 – Configure the Service Certificate for WCF

In this step, you configure WCF to use the temporary certificate you created in the previous step.

1. In the configuration editor, expand the **Advanced** node, expand the **ServiceBehaviors** and **ServiceBehavior** nodes, and then expand the **serviceCredentials** node.
2. Select the **serviceCertificate** node and set the **FindValue** attribute to the subject name of the certificate you are going to use; for example, "CN=tempCert".
3. In the configuration editor dialog box, on the **File** menu, select **Save**.
4. In Visual Studio, verify your configuration. The configuration should look as follows:

```
...
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
      <serviceMetadata httpGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceCredentials>
        <serviceCertificate findValue="CN=tempCert" />
        <userNameAuthentication
userNamePasswordValidationMode="MembershipProvider"
          membershipProviderName="MySqlMembershipProvider" />
      </serviceCredentials>
    </behavior>
  </serviceBehaviors>
</behaviors>
```

. . .

Step 8 – Create a User in the User Store

In this step, you will create a user that the test client will use to log into the service.

1. In the Solution Explorer, choose the WCF service project, and then on the **Website** menu, select **ASP.NET Configuration**.
2. On the ASP.NET Web Site Administration Tool page, click the **Security** tab, and then click the **Select authentication type** link.
3. On the page that appears, select the **From the internet** radio button and then click **Done**.
4. Click the **Create user** link.
5. On the Create User page, enter the details of the user you want to create in the SQL store and then click **Create User**.
   If successful, a new user will be created. By default, you will need to create a password of at least seven characters with one character that is not alphanumeric.

Step 9 – Create a Test Client

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.

Step 10 – Add a WCF Web Reference to the Client

In this step, you add a reference to your WCF service.

1. Right-click your Client project and select **Add Web Reference**.
2. In the **Add Web Reference** dialog box, set the URL to your WCF service (e.g., http://localhost/WCFTestService/Service.svc) and then click **Go**.
3. In the **Web reference name** field, change ServiceReference1 to **WCFTestService**.
4. Click **Add Reference**.
   In your Client project, a reference to WCFTestService should now appear beneath **Web References**.

Step 11 – Test the Client and WCF Service

In this step, you access the WCF service, pass the user credentials, and make sure that the username authentication works.

1. In your Client project, drag a **Button** control onto your form.

2. Double-click the **Button** control to show the underlying code.
3. Create an instance of the proxy, pass the credentials of the user created in step 8, and then call the **GetData** operation of your WCF Service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                  WCFTestService.ServiceClient();
      myService.ClientCredentials.UserName.UserName = "username";
      myService.ClientCredentials.UserName.Password = "p@ssw0rd";
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the Client project and select **Set as Startup Project**.
5. Run the Client application by pressing F5 or Ctrl+F5.
   When you click the button on the form, it should display the message "**You entered: 123**".

Additional Resources

- For more information on how to work with the SQL Server Membership Provider, see "How to: Use the ASP.NET Membership Provider" at http://msdn2.microsoft.com/en-us/library/ms731049.aspx
- For more information on how to work with temporary certificates, see "How to: Create Temporary Certificates for Use During Development" at http://msdn2.microsoft.com/en-us/library/ms733813.aspx and "Working with Certificates" at http://msdn2.microsoft.com/en-us/library/ms731899.aspx
- For more information on how to view certificates with the Microsoft Management Console (MMC) snap in, see "How to: View Certificates with the MMC Snap-in" at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see "Differences Between Service Certificate Validation Done by Internet Explorer and WCF" at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see "Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security" at http://msdn2.microsoft.com/en-us/library/aa702579.aspx

How To – Use Username Authentication with Transport Security in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of using username authentication with transport security to authenticate your users against a Microsoft SQL Server® membership provider and optionally authorize users with the SQL Server role provider. The article shows you how to configure the membership provider and the role provider, create a custom HTTP module for authenticating users against the membership provider, and create a class that derives from **IauthorizationPolicy** so that WCF can authorize users.

Contents

- Step 17 – Test the WCF/ASMX Client and WCF Service
- Additional Resources

Objectives

- Learn to configure the SQL Server Membership Provider.
- Learn to configure the SQL Server Role Provider.
- Learn to create a custom HTTP module to authenticate the user by using the SQL Server membership provider with transport security in Internet Information Services (IIS).
- Learn to configure the custom HTTP module.
- Learn to call the service from a WCF test client and from an ASMX test client.

Overview

Username authentication is suited for scenarios in which your users do not have Microsoft Windows® credentials. In the scenario described in this How To article, users and roles are stored in SQL Server. Users are authenticated in ASP.NET against the SQL Server membership provider and optionally are authorized in the WCF service against the SQL Server role provider. The the WCF service uses the **basicHttpBinding** binding type to provide compatibility with ASMX clients**.** Because transport security does not support username authentication, a custom HTTP module will be created for authentication, and the authentication type will be set to “None” in the WCF Service.

To use the SQL Server membership provider and role provider, you will first create a user and role store and then populate it with your users and roles. You will then configure the store to allow access to the WCF service process, in order to authenticate and authorize the users. Finally, you must configure IIS and WCF security settings to allow users to be authenticated and authorized correctly in ASP.NET and the WCF service.

Summary of Steps

- **Step 1 – Create a User Store for the SQL Server Membership Provider and a Role Store for the SQL Server Role Provider**
- **Step 2 – Grant Access Permission to the WCF Service Process**
- **Step 3 – Create a Sample WCF Service**
- **Step 4 – Configure basicHttpBinding with Transport Security and an Authentication Type of “None”**
- **Step 5 – Configure the WCF Service for ASP.NET Compatibility Mode**
- **Step 6 – Configure the SQL Server Membership Provider in the Web Configuration File**
- **Step 7 – Configure the SQL Server Role Provider and Enable It in WCF**
- **Step 8 – Create the User and Assign Roles**

- **Step 9 – Implement a Custom HTTP Module Class That Derives from IHttpModule to Authenticate Users with the SQL Server Membership Provider**
- **Step 10 – Configure the WCF Service to Use the HTTP Module for Authentication**
- **Step 11 – Implement a Class that Derives from IAuthorizationPolicy**
- **Step 12 – Configure the WCF Service to Use the Authorization Policy**
- **Step 13 – Configure Security Settings in IIS**
- **Step 14 – Implement Authorization Checks on Your Service**
- **Step 15 – Create a Test Client**
- **Step 16 – Add a WCF Service Reference and Web Service Reference to the Client**
- **Step 17 – Test the WCF/ASMX Client and WCF Service**

Step 1 – Create a User Store for the SQL Server Membership Provider and a Role Store for the SQL Server Role Provider

The SQL Server membership provider stores user information, and the SQL Server role provider stores role information, in a SQL Server database. The user and role information work independently and can be created in different SQL Server databases. For the purposes of this example, both will be created in the same database. You can create your SQL Server user and role store manually by using the Aspnet_regsql.exe command from the command line.

- From a Visual Studio 2008 command prompt, run the following command:

```
aspnet_regsql -S .\SQLExpress -E -A m -A r
```

In this command:

- **-S** specifies the database server, which is **(.\SQLExpress)** in this example.
- **-E** specifies to use Windows authentication to connect to SQL Server.
- -**A m –A r** specifies to add the membership and the role features.

For a complete list of the commands, run **Aspnet_regsql /?**

Note: By default, a database with the name Aspnetdb is created. Use the **-d** option to specify a different database name.

Step 2 – Grant Access Permission to the WCF Service Process

In this step, you grant the WCF service process identity access to the Aspnetdb database. If you host the WCF service in IIS 6.0 on Microsoft Windows Server® 2003, the NT AUTHORITY\Network Service account is used by default to run the WCF service.

1. Create a SQL Server login for NT AUTHORITY\Network Service.
2. Grant the login access to the Aspnetdb database by creating a database user.
3. Add the user to the **aspnet_Membership_FullAccess** database role.
4. Add the user to the **aspnet_Roles_FullAccess** database role.

You can perform these steps by using the SQL Server Management Studio, or you can run the following script in SQL Query Analyzer:

```
-- Create a SQL Server login for the Network Service account
sp_grantlogin 'NT AUTHORITY\Network Service'

-- Grant the login access to the membership database
USE aspnetdb
GO
sp_grantdbaccess 'NT AUTHORITY\Network Service', 'Network Service'

-- Add user to database role
USE aspnetdb
GO
sp_addrolemember 'aspnet_Membership_FullAccess', 'Network Service'

USE aspnetdb
GO
sp_addrolemember 'aspnet_Roles_FullAccess', 'Network Service'
```

Note:

- If you are running on Microsoft Windows® XP, create a SQL Server login for the ASPNET identity instead of the NT Authority\Network Service identity, as the IIS process runs under the ASPNET account in Windows XP.
- If you do not have Management Studio or Query Analyzer, you can use Microsoft SQL Server Management Studio Express (SSMSE), available at http://www.microsoft.com/downloads/details.aspx?FamilyID=c243a5ae-4bd1-4e3d-94b8-5a0f62bf7796&displaylang=en

Step 3 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio.

1. In Visual Studio, on the menu, click **File** and then click **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http** and specify the virtual directory to be created in the **Path (**e.g., https://localhost/WCFTestService).
3. In the **New Web Site** dialog box, click **OK** to create a virtual directory and a sample WCF service.
4. Change the service metadata to be published via https instead of http. Double-click the Web.config file, scroll down to **service metadata**, and change the entry <serviceMetadata httpGetEnabled="true"/> to <serviceMetadata httpsGetEnabled="true"/>

Step 4 – Configure basicHttpBinding with Transport Security and an Authentication Type of "None"

In this step, you configure the WCF service to use transport security and an authentication type of "none."

1. In the Solution Explorer, right-click the Web.config file of the WCF service and then click **Edit WCF Configuration**.
   If you do not see the Edit WCF Configuration option, click the **Tools** menu and then click **WCF Service Configuration Editor**. Close the **WCF Service Configuration Editor** tool that appears. The option should now appear on the web.config context menu.
2. In the Configuration Editor, in the **Configuration** section, expand **Service** and then expand **Endpoints**.
3. Select the first node **[Empty Name].** Set the **name** attribute to **basicEndpoint**.
   By default, the name field will be empty because it is an optional attribute.
4. Click the binding and change it from wsHttpBinding to **basicHttpBinding.**
5. Click the **Identity** tab and then delete the **Dns** attribute value.
6. Select the second node **[Empty Name].** Set the **name** attribute to **MexEndpoint**.
   By default, the name field will be empty because it is an optional attribute.
7. Click the binding and change it to **mexHttpsBinding**.
8. In the Configuration Editor, select the Bindings folder.
9. In the **Bindings** section, choose **New Binding Configuration**.
10. In the **Create a New Binding** dialog box, select **basicHttpBinding**.
11. Click **OK**.
12. Set the **Name** of the binding configuration to some logical and recognizable name; for example, **BasicBindingConfiguration**.
13. Click the **Security** tab and then set the **Mode** attribute to **Transport**.
14. Set the **TransportClientCredentialType** to **None** by selecting this option from the drop-down list.
    Authentication will be done by the HTTP module in ASP.NET.
15. In the **Configuration** section, select the **basicEndpoint** node.
16. Set the **BindingConfiguration** attribute to **BasicBindingConfiguration** by selecting this option from the drop-down list.
    This associates the binding configuration setting with the binding.
17. In the Configuration Editor, on the **File** menu, select **Save**.
18. In Visual Studio, open your configuration and delete the identity element under **basicEndpoint** node, if there is one.
19. In Visual Studio, verify your configuration. The configuration should look as follows:

```
<bindings>
      <basicHttpBinding>
        <binding name="BasicBindingConfiguration">
          <security mode="Transport" />
            <transport clientCredentialType="None" />
          </security>
```

```
        </binding>
</basicHttpBinding>
</bindings>
<services>
      <service behaviorConfiguration="ServiceBehavior" name="Service">
        <endpoint address="" binding="basicHttpBinding"
          bindingConfiguration="BasicBindingConfiguration"
          name="basicEndpoint" contract="IService" />
         <endpoint address="mex" binding="mexHttpsBinding"
          bindingConfiguration=""
          contract="IMetadataExchange" />
      </service>
 </services>
```

Step 5 – Configure the WCF Service for ASP.NET Compatibility Mode

In this step, you configure the WCF service for ASP.NET compatibility mode. This is required because authentication is done by the HTTP module and you must be able to get the principal and the identity from the HTTP context in order to authorize users either imperatively or declaratively in WCF.

1. Right-click the Service.svc file and then click **View Code**.
2. Add the following **using** statement to the list of other **using** statements:
   using System.ServiceModel.Activation;
3. Add the following attribute on top of the **Service** class:
   [AspNetCompatibilityRequirements(RequirementsMode =
   AspNetCompatibilityRequirementsMode.Required)]
   public class Service : IService

4. Open Web.config and add the following entry on the top, inside the <system.serviceModel> node:

```
<system.serviceModel>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
```

Step 6 – Configure the SQL Server Membership Provider in the Web Configuration File

In this step, you configure the SQL Server membership provider in the Web Configuration file.

1. In the Web.config file, replace the existing single <connectionStrings/> element with the following to point to your SQLServer membership provider database:

```
<connectionStrings>
  <add name="MyLocalSQLServer"
       connectionString="Initial Catalog=aspnetdb;
      data source=.\sqlexpress;Integrated Security=SSPI;" />
</connectionStrings>
```

2. Add a <membership> element inside the <system.web> element as shown in the following example.
   Note the use of the <clear/> element prevents the default provider from being loaded and then never used. Make sure that the connection string name points to your previous connection string setting.

```
...
<system.web>
  ...
  <membership defaultProvider="MySqlMembershipProvider" >
    <providers>
      <clear/>
      <add name="MySqlMembershipProvider"
           connectionStringName="MyLocalSQLServer"
           applicationName="MyAppName"
           type="System.Web.Security.SqlMembershipProvider" />
    </providers>
  </membership>
</system.web>
...
```

3. Save the Web.Config file, to ensure that the changes do not get lost during the following steps.

Step 7 – Configure the SQL Server Role Provider and Enable It in WCF

In this step, you configure the use of the SQL Server role provider in the Web.configfile and enable it in your WCF service.

1. Add a <roleManager> element inside the <system.web> element as shown in the following example.
   Note the use of the <clear/> element, which prevents the default provider from being loaded and then never used.

```
...
<system.web>

  ...
  <roleManager enabled="true" defaultProvider="MySqlRoleProvider"
>
    <providers>
      <clear/>
      <add name="MySqlRoleProvider"
           connectionStringName="MyLocalSQLServer"
           applicationName="MyAppName"
           type="System.Web.Security.SqlRoleProvider" />
    </providers>
  </roleManager>
</system.web>
...
```

2. Save the Web.Config file; otherwise the changes might get lost during execution of the following steps.
3. Right-click the Web.config file of the WCF service and then click **Edit WCF Configuration**.
   If you do not see the Edit WCF Configuration option, select **WCF Service Configuration Editor** on the **Tools** menu. Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu.
4. In the Configuration Editor, expand the **Advanced** node, and then expand the Service Behaviors folder.
5. Select the default behavior, **ServiceBehavior**.
6. In the **Behavior: ServiceBehavior** section, click **Add**.
7. In the **Adding Behavior Element Extension Sections** dialog box, select **serviceAuthorization**, and then click **Add**.
8. In the **Configuration** section, under **Service Behaviors**, select the **serviceAuthorization** option.
9. Set the **principalPermissionMode** attribute to **UseAspNetRoles** by choosing this option from the drop-down list.
10. Set the **roleProviderName** attribute to **MySqlRoleProvider**, which you created above.
11. In the Configuration Editor, on the **File** menu, click **Save**.
12. In Visual Studio, verify your configuration, which should look as follows:

```
….
<behavior name="ServiceBehavior">
  <serviceMetadata httpsGetEnabled="true" />
    <serviceDebug includeExceptionDetailInFaults="false" />
       ….
       <serviceAuthorization
       principalPermissionMode="UseAspNetRoles"
             roleProviderName="MySqlRoleProvider" />
       ….
 </behavior>
….
```

Step 8 – Create the User and Assign Roles

In this step, you create the user and assign a newly created role by using the ASP.NET Web Site Configuration Tool.

1. In the Solution Explorer, select the WCF service project, and then on the **Website** menu, select **ASP.NET Configuration**.
2. On the ASP.NET Web Site Administration Tool page, click the **Security** tab, and then click the **Select authentication type** link.
3. On the page that appears, select the **From the internet** radio button and then click **Done**.
4. Click the **Create user** link.

5. On the Create User page, enter the details of the user you want to create in the SQL Server store, and then click **Create User**.
   If successful, a new user will be created.
6. Click the **Create or Manage roles** link.
7. Enter the new role name – for example, "Managers" – and then click **Add Role**.
   If successful, a new role will be created.
8. On the Roles creation page, click the **Manage** link, choose the user created in the previous step, and assign this user to the role by selecting the **User Is In Role** check box.

Step 9 – Implement a Custom HTTP Module Class That Derives from IHttpModule to Authenticate Users with the SQL Server Membership Provider

In this step, you implement a custom HTTP module to authenticate users by using the SQL Server membership provider. You will create a Windows class library that will derive from **IHttpModule**.

1. Open a new instance of Visual Studio, leaving your WCF service solution open.
2. In the new instance of Visual Studio, click **File**, click **New**, and then click **Project**.
3. Expand **Visual C#,** click **Windows, and** then select **Class Library.**
4. In the **Name** field, enter **UserAuthenticator** and then click **OK**.
5. In the Solution Explorer, right-click **References**, click **Add Reference**, select the **.NET** tab, select System.Web, and then click **OK**.
6. Copy and paste the code below inside the Class1.cs file. This class has the following characteristics:
   a. It subscribes to event handlers Authenticate Request, which will execute when the request starts to authenticate the user, and End Request, which will execute when Authenticate Request has finished.
   b. It checks for the authorization header. If it is not present, it sends a response back to the client with the header "WWW-Authenticate" to challenge the client to send credentials. If it is present, it extracts the username and credentials from the header and authenticates them against the SQL Server membership provider.
   c. If authentication fails, it returns status 401 to flag authentication failure. If authentication succeeds, it builds the generic principal and assigns it to the HTTP application context current user, o be used for later WCF authorization.
7. Build the class solution.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;
using System.Web.Security;
```

```
using System.Security.Principal;

namespace Module
{
    public class UserNameAuthenticator: IHttpModule
    {


        public void Dispose()
        {
        }

        public void Init(HttpApplication application)
        {
            application.AuthenticateRequest += new
            EventHandler(this.OnAuthenticateRequest);
            application.EndRequest += new
            EventHandler(this.OnEndRequest);
        }

        public void OnAuthenticateRequest(object source, EventArgs
                                          eventArgs)
        {

            HttpApplication app = (HttpApplication)source;

            //the Authorization header is checked if present
            string authHeader = app.Request.Headers["Authorization"];
            if (!string.IsNullOrEmpty(authHeader))
            {
                string authStr = app.Request.Headers["Authorization"];

                if (authStr == null || authStr.Length == 0)
                {
                    // No credentials; anonymous request
                    return;
                }

                authStr = authStr.Trim();
                if (authStr.IndexOf("Basic", 0) != 0)
                {
                    // header is not correct...we'll pass it along and
                    // assume someone else will handle it
                    return;
                }


                authStr = authStr.Trim();


                string encodedCredentials = authStr.Substring(6);

                byte[] decodedBytes =
                Convert.FromBase64String(encodedCredentials);
                string s = new ASCIIEncoding().GetString(decodedBytes);

                string[] userPass = s.Split(new char[] { ':' });
```

```
                string username = userPass[0];
                string password = userPass[1];
                //the user is validated against the
                //SqlMemberShipProvider
                //If it is validated then the roles are retrieved from
                // the role provider and a generic principal is created
                //the generic principal is assigned to the user context
                // of the application


                if (Membership.ValidateUser(username, password))
                {
                    string[] roles = Roles.GetRolesForUser(username);
                    app.Context.User = new GenericPrincipal(new
                    GenericIdentity(username, "Membership Provider"),
                                    roles);

                }
                else
                {
                    DenyAccess(app);
                    return;

                }



            }
            else
            {
                app.Response.StatusCode = 401;
                app.Response.End();


            }
 }
        public void OnEndRequest(object source, EventArgs eventArgs)
        {
            //the authorization header is not present
            //the status of response is set to 401 and it ended
            //the end request will check if it is 401 and add
            //the authentication header so the client knows
      //it needs to send credentials to authenticate
            if (HttpContext.Current.Response.StatusCode == 401)
            {
                HttpContext context = HttpContext.Current;
                context.Response.StatusCode = 401;
                context.Response.AddHeader("WWW-Authenticate", "Basic
Realm");
            }
        }

        private void DenyAccess(HttpApplication app)
        {
            app.Response.StatusCode = 401;
            app.Response.StatusDescription = "Access Denied";
```

```
            // Write to response stream as well, to give user visual
            // indication of error during development
            app.Response.Write("401 Access Denied");

            app.CompleteRequest();
        }


    }
}
```

Step 10 – Configure the WCF Service to Use the HTTP Module for Authentication

In this step, you configure the WCF service to use the HTTP module for authentication.

1. Right-click **Web site** and then click **Add Reference**. Select the **Browse** tab, navigate to the directory containing the UserNameAuthenticator.dll file, select the file, and then click **OK**.
2. Configure the HTTP module in Web.Config. Open Web.Config in the Visual Studio editor and locate the **httpModules** node. If there is one, add the authentication module only as below. If it is not present create an **httpModules** entry with the authentication module as shown below:

```
<httpModules>
…
<add name="BasicAuthenticationModule"
type="Module.UserNameAuthenticator,UserAuthenticator" />
</httpModules>
```

Step 11 – Implement a Class that Derives from IAuthorizationPolicy

In this step, you implement a class that derives from **IAuthorizationPolicy**, which will assign the principal to the thread current principal and the identity to the WCF context. This will allow declarative authorization and retrieval of the caller's identity from the WCF security context.

1. Open a new instance of Visual Studio, leaving your WCF service solution open.
2. In the new instance of Visual Studio click **File**, click **New**, and then click **Project**.
3. Expand **Visual C#,** click **Windows**, **and** then select **Class Library**.
4. In the **Name** field, enter **AuthorizationPolicy** and then click **OK**.
5. In the Solution Explorer, right-click on **References** and then click **Add Reference**. Select the **.NET** tab, select **System.Web**, select **System.IdentityModel,System.IdentityModel.Selectors**, and then click **OK**.
6. Copy and paste the code below inside the Class1.cs file.
   This class has the following characteristics:

a. It implements Evaluate method, which retrieves the principal from **Httpcontext** and assigns it to the evaluation context of WCF, so that WCF can do declarative authorization checks.
b. It implements Evaluate, which retrieves the identity from **Httpcontext** and assigns it to the evaluation context of WCF, so that WCF can retrieve the identity of the caller from the WCF security context; otherwise It will not be available and you will need to get the identity from **HttpContext**.

7. Build the class solution.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IdentityModel.Claims;
using System.IdentityModel.Policy;
using System.Web;
using System.Security.Principal;


namespace AuthorizationPolicy
{
    // syncs Thread.CurrentPrincipal in WCF with whatever is set
    // by the HTTP pipeline on Context.User (optional)
    public class HttpContextPrincipalPolicy : IAuthorizationPolicy
    {
        public bool Evaluate(EvaluationContext evaluationContext, ref
                             object state)
        {
            HttpContext context = HttpContext.Current;

            if (context != null)
            {
                evaluationContext.Properties["Principal"] =
                context.User;
                evaluationContext.Properties["Identities"] =
                   new List<IIdentity>() { context.User.Identity };
            }

            return true;
        }

        public System.IdentityModel.Claims.ClaimSet Issuer
        {
            get { return ClaimSet.System; }
        }

        public string Id
        {
            get { return "HttpContextPrincipalPolicy"; }
        }
    }
}
```

Step 12 – Configure the WCF Service to Use the Authorization Policy

In this step, you configure the WCF service to use the authorization policy.

1. Right-click **Web site** and then click **Add Reference**. Select the **Browse** tab, navigate to the directory containing the AuthorizationPolicy.dll file, and then click **OK**.
2. Right-click the Web.config file of the WCF service and then click **Edit WCF Configuration**.
   If you do not see the Edit WCF Configuration option, select **WCF Service Configuration Editor** on the **Tools** menu. Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu.
3. In the Configuration Editor, expand the **Advanced** node, and then expand the Service Behaviors folder.
4. Select the default behavior, **ServiceBehavior**.
5. In the **Configuration** section, under **Service Behaviors**, select the **serviceAuthorization** option.
6. Click the **AuthorizationPolicies** tab and then click **new.** Navigate to the bin directory of the Web site and select AuthorizationPolicy.dll.
7. Select the **AuthorizationPolicy.HttpContextPrincipalPolicy** type and then click **OK**.
8. In the **Configuration Editor** dialog box, on the **File** menu, select **Save**.
   The configuration file should now include the following configuration:

```
  <serviceAuthorization principalPermissionMode="UseAspNetRoles"
            roleProviderName="MySqlRoleProvider">
            <authorizationPolicies>
              <add
           policyType="AuthorizationPolicy.HttpContextPrincipalPolicy,
AuthorizationPolicy, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null" />
            </authorizationPolicies>
  </serviceAuthorization>
```

Step 13 – Configure Security Settings in IIS

In this step, you configure the Web site in IIS to use Secure Sockets Layer (SSL) and anonymous security. You will need to have created a certificate for SSL security. To create a temporary certificate for SSL, refer to document “How To - Create and Install Temporary Certificates in WCF for Transport Security During Development.”

1. Click **Start** and then click **Run**.
2. In the **Run** dialog box, type **inetmgr** and then click **OK**.
3. In the **Internet Information Services (IIS) Manager** dialog box, expand the **(local computer)** node, and then expand the **Web Sites** node.
4. Right-click **Default Web Site** and then click **Properties**.

5. In the **Default Web Site Properties** dialog box, click the **Directory Security** tab, and then in the **Secure Communications** section, click **Server Certificate**.
6. On the Welcome screen of the Web Server Certificate Wizard, click **Next** to continue.
7. On the Server Certificate screen, select the **Assign an existing certificate** radio button option, and then click **Next**.
8. On the Available Certificates screen, select the certificate you created and installed, and then click **Next**.
9. Expand the **Default Web Site** node, right-click the virtual directory on your WCF service Web site, and then click **Properties**.
10. In the **Web site properties** dialog box, click the **Directory Security** tab, and then in the **Anonymous access and authentication control** section, click **Edit**. Select only the **Enable Anonymous Access** option. Clear the other options.
11. In the WCF service project, double-click the Web.config file and set **Authentication** to **None**.
    The following XML fragment contains the example:

```
<system.web>
…
<authentication mode="None" />
```

Step 14 – Implement Authorization Checks on Your Service

In this step, you authorize the user in WCF, either declaratively or imperatively.

Declarative check

1. Open the Service.cs file and add the following statement for using the **System.Security.Permissions** namespace:

```
using System.Security.Permissions;
```

2. Add the **PrincipalPermissionAttribute** to authorize users in the Managers role, with the **SecurityAction** as **Demand** to the **GetData** method:

```
[PrincipalPermission(SecurityAction.Demand, Role="Managers")]
public string GetData(int value)
{
      return string.Format("You entered: {0}", value);
}
```

Imperative check

In the same **GetData** method, you can perform the imperative check with the call to **Roles.IsUserInRole**. The below example demonstrates the imperative check:

```
using  System.Web.Security;
...
public string GetData(int value)
      {
        if(Roles.IsUserInRole("accounting"))
        {
                  return string.Format("You entered: {0}", value);
        }
        else return “not authorized”  ;
      }
```

You can also retrieve the identity of the user with the call to

```
ServiceSecurityContext.Current.PrimaryIdentity.Name
```

Step 15 – Create a Test Client

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.

Step 16 – Add a WCF Service Reference and Web Service Reference to the Client

In this step, you add both a WCF and Web reference to your WCF service to test consuming your service from WCF and ASMX clients.

Add a WCF Service reference

1. Right-click your client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the **URL** to your WCF service (e.g., https://localhost/WCFTestService/Service.svc) and then click **Go**.
3. You will be asked to enter a username and password. Provide a valid user name and password created in previous steps.
4. In the **Service reference name** field, change ServiceReference1 to **WCFTestService**. Click **OK**.
5. In your Client project, a reference to WCFTestService should now appear beneath **Service References.**
6. Double click the app.config file and change the authentication mode from none to basic

```
<basicHttpBinding>
<binding name="basicEndpoint" closeTimeout="00:01:00" />

    <security mode="Transport">
        <transport clientCredentialType="Basic"
        proxyCredentialType="None"
```

```
            realm="" />
        </binding>
```

Add a Web Service reference

1. Right-click your Client project and select **Add Service Reference**. Click on **Advanced** button then click **Add Web Reference** .
2. In the **Add Web Reference** dialog box, set the URL to your WCF service (e.g., http://localhost/WCFTestService/Service.svc) and then click **Go**. You will be asked to enter user name and password. Provide a valid user name and password created in previous steps.
3. In the **Web reference name** field, change ServiceReference1 to **ASMXTestService**.
4. Click **Add Reference**.
   In your client project, a reference to WCFTestService should now appear beneath **Web References.**

Step 17 – Test the WCF/ASMX Client and WCF Service

In this step, you access the WCF service, pass the user credentials, and make sure that the username authentication works.

1. In your client project, drag a **Button** control onto your form.
2. Double-click the **Button** control to show the underlying code.
3. Create an instance of the proxy, pass the credentials of the user created in step 8, and then call the **GetData** operation of your WCF Service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      //WCF client call
     WCFTestService.ServiceClient myService = new
     WCFTestService.ServiceClient();
     myService.ClientCredentials.UserName.UserName = "username";
     myService.ClientCredentials.UserName.Password = "p@ssw0rd";
     MessageBox.Show(myService.GetData(123));
     myService.Close();
      //ASMX client call
     NetworkCredential netCred = new
     NetworkCredential("username", " p@ssw0rd");
     ASMXTestService.Service proxy = new
     basicreference.Service();
     proxy.Credentials = netCred;
     proxy.GetData(21, true);
}
```

4. Right-click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5.
   When you click the button on the form, the message "**You entered: 123**" should appear.

Additional Resources

- For more information on how to work with the ASP.NET Role Provider, see “How to: Use the ASP.NET Role Provider with a Service” at http://msdn2.microsoft.com/en-us/library/aa702542.aspx
- For more information on how to work with the ASP.NET Role Manager, see “How To: Use Role Manager in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/ms998314.aspx
- For more information on how to work with the ASP.NET Membership Provider, see “How to: Use the ASP.NET Membership Provider” at http://msdn2.microsoft.com/en-us/library/ms731049.aspx
- For more information on how to work with temporary certificates, see “How to: Create Temporary Certificates for Use During Development” at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on how to view certificates with the Microsoft Management Console (MMC) snap in, see “How to: View Certificates with the MMC Snap-in” at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see “Differences Between Service Certificate Validation Done by Internet Explorer and WCF” at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see “Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security” at http://msdn2.microsoft.com/en-us/library/aa702579.aspx
- For more Information on IHTTP Module interface, see http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.aspx

How To – Use wsHttpBinding with UserName Authentication and TransportWithMessageCredentials in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of using username authentication over **wsHTTPBinding** to authenticate your users against a Microsoft SQL Server® Membership Provider. The WCF service in this article will use transport security, with credentials in the message protected using message security. The article shows you how to configure the membership provider, configure WCF, create and install the necessary certificate, and test the service with a sample WCF client.

Contents

Objectives

- Configure the SQL Server Membership Provider.
- Create a WCF service hosted in Internet Information Services (IIS).
- Configure the service to use the Secure Sockets Layer (SSL) protocol.
- Create and configure a certificate for the service.

- Expose the WCF service over **wsHttpBinding by** using TransportWithMessageSecurity.
- Call the service from a test client.

Overview

Username authentication is suited for scenarios in which your users do not have domain credentials. In the scenario described in this How To article, users are stored in SQL Server and are authenticated against the SQL Server Membership Provider, an identity management system that uses forms authentication. The **wsHttpBinding** binding is used in order to provide support for message-based security, reliable messaging, and transactions, while also allowing the possibility that legacy clients can consume the service. WCF TransportWithMessageCredential security is used to support a secure communication channel in a point-to-point scenario while allowing you to transmit user credentials that are encrypted and protected in the message.

In order to use the SQL Server Membership Provider, you will first create a user store and populate it with your users. You will then configure the store to allow the WCF service access to authenticate users. You will set the **clientCredentialType** attribute to **UserName** on the **wsHttpBinding** binding in order to configure the WCF service to use username authentication. You will then install a certificate on the server and configure it for WCF so that messages sent between client and server are encrypted.

Summary of Steps

- **Step 1 – Create a User Store for the SQL Server Membership Provider**
- **Step 2 – Grant Access Permission to the WCF Service Process Identity**
- **Step 3 – Create and Install a Service Certificate for Transport Security**
- **Step 4 – Create a Sample WCF Service Project with SSL**
- **Step 5 – Configure the Virtual Directory to Require SSL**
- **Step 6 – Configure wsHttpBinding for Username Authentication and TransportWithMessageCredential Security**
- **Step 7 – Configure the Service to Publish Metadata Securely**
- **Step 8 – Configure the Membership Provider for Username Authentication**
- **Step 9 – Create a User in the User Store**
- **Step 10 – Create a Test Client Application**
- **Step 11 – Add a WCF Service Reference to the Client**
- **Step 12 – Test the Client and WCF Service**

Step 1 – Create a User Store for the SQL Server Membership Provider

The SQL Server Membership Provider stores user information in a SQL Server database. You can create your SQL Server user store manually by using Aspnet_regsql.exe from the command line.

To do this, from a Visual Studio 2008 command prompt, run the following command:

```
aspnet_regsql -S .\SQLExpress -E -A m
```

In this command:

- **-S** specifies the server, which is **(.\SQLExpress)** in this example.
- **-E** specifies to use Windows Authentication to connect to SQL Server.
- -**A m** specifies to add only the membership provider feature. For simple authentication against a SQL Server user store, only the membership provider feature is required.
- For a complete list of the commands, run **Aspnet_regsql /?**

Step 2 – Grant Access Permission to the WCF Service Process Identity

Your WCF service process identity requires access to the Aspnetdb database. If you host the WCF service in Internet Information Services (IIS) 6.0 on Microsoft Windows Server® 2003, the NT AUTHORITY\Network Service account is used by default to run the WCF service.

To grant database access

1. Create a SQL Server login for NT AUTHORITY\Network Service.
2. Grant the login access to the Aspnetdb database by creating a database user.
3. Add the user to the **aspnet_Membership_FullAccess** database role.

You can perform these steps by using the SQL Server Enterprise Manager, or you can run the following script in SQL Query Analyzer:

```
-- Create a SQL Server login for the Network Service account
sp_grantlogin 'NT AUTHORITY\Network Service'

-- Grant the login access to the membership database
USE aspnetdb
GO
sp_grantdbaccess 'NT AUTHORITY\Network Service', 'Network Service'

-- Add user to database role
USE aspnetdb
GO
sp_addrolemember 'aspnet_Membership_FullAccess', 'Network Service'
```

Note:

- If you are running on Microsoft Windows® XP, create a SQL Server login for the ASPNET identity instead of the NT Authority\Network Service identity, as the IIS process runs under the ASPNET account in Windows XP.

- If you do not have Enterprise Manager or Query Analyzer installed, you can use Microsoft SQL Server Management Studio Express (SSMSE), available at http://www.microsoft.com/downloads/details.aspx?FamilyID=c243a5ae-4bd1-4e3d-94b8-5a0f62bf7796&displaylang=en

Step 3 – Create and Install a Service Certificate for Transport Security

In this step, you create a temporary service certificate and install it in the local store. This certificate will be used for establishing an SSL connection between the client and the WCF service.

Creating and installing the certificate is outside the scope of this How To article. For details on how to do this, see "How To – Create and Install Temporary Certificates in WCF for Transport Security during Development" and follow steps 1 through 4.

Note: Temporary certificates should be used for development and testing purposes only. For actual production deployment, get a valid certificate from a certificate authority (CA).

Step 4 – Create a Sample WCF Service Project with SSL

In this step, you create a WCF service in Visual Studio and enable SSL.

1. In Visual Studio, on the **File** menu, click **New Web Site**.
2. In the **Templates** section, select **WCF Service**. Make sure that the **Location** is set to **Http** and then click **Browse**.
3. In the **Choose Location** dialog box, click **Local IIS**.
4. Select the **Use Secure Sockets Layer** check box at the bottom of the dialog box, and then click **Open**.
5. In the **New Web Site dialog** box, set the new **Location** to **https://localhost/WCFTestService** and then click **OK**.

Note: The SSL port might not be configured by default on the IIS, so it might throw errors while creating the WCF service. To prevent this, open IIS Manager, right-click **Default Web Site**, and then click **Properties** option. In the **Default Web Site Properties** dialog box, click the **Web Site** tab and make sure that the **SSL port:** is set to 443.

Step 5 – Configure the Virtual Directory to Require SSL

In this step, you configure the virtual directory hosting the WCF service to use SSL.

1. Click **Start** and then click **Run**.
2. In the command line, type **inetmgr** and then click **OK** to open the IIS Manager.

3. In IIS Manager, expand the **(local computer)** node**,** expand the **Web Sites** node, and then expand the **Default Web Site** node.
4. Right-click your virtual directory (WCFTestService) and then click **Properties**.
5. In the Default Web Site Properties dialog box, on the **Directory Security** tab, click **Edit** in the **Secure Communications** section.
6. In the **Secure communications** dialog box, select the **Require secure channel (SSL)** check box.
7. In the **Secure communications** dialog box, click **OK**.
8. In the **Default Web Site Properties** dialog box, click **OK**.

Step 6 – Configure wsHttpBinding for Username Authentication and TransportWithMessageCredential Security

In this step, you configure the WCF service to use username authentication and TransportWithMessageCredentialSecurity.

1. In the Solution Explorer, right-click the Web.config file of the WCF service and then click **Edit WCF Configuration**.
   If you do not see the Edit WCF Configuration option, click the **Tools** menu and then click **WCF Service Configuration Editor.** Close the WCF Service Configuration Editor tool that appears. The option should now appear on the web.config context menu**.**
2. In the **C**onfiguration Editor, in the **Configuration** section, expand **Service** and then expand **Endpoints**.
3. Select the first node **[Empty Name]** and then set the **Name** attribute to **wsHttpEndpoint**.
4. Click the **Identity** tab and delete the **Dns** attribute value.
5. In the Configuration Editor, select the Bindings folder.
6. In the **Bindings** section, choose **New Binding Configuration**.
7. In the **Create a New Binding** dialog box, select **wsHttpBinding**.
8. Click **OK**.
9. Set the **Name** of the binding configuration to some logical and recognizable name; for example, **wsHttpEndpointBinding**.
10. Click the **Security** tab.
11. Set the **Mode** attribute to **TransportWithMessageCredential** by choosing this option from the drop-down list.
12. Set the **MessageClientCredentialType** to **UserName** by choosing this option from the drop-down list.
13. Set the **TransportClientCredentialType** to **None** by choosing this option from the drop-down list.
14. In the **Configuration** section, select the **wsHttpEndpoint** node.

15. Set the **BindingConfiguration** attribute to **wsHttpEndpointBinding** by choosing this option from the drop-down list.
    This associates the binding configuration setting with the binding.
16. In the Configuration Editor, on the **File** menu, click **Save**.
17. In Visual Studio, open your configuration and comment out the identity element. It should look as follows:

```
<!--<identity>
  <dns value="" />
</identity>-->
```

18. In Visual Studio, verify your configuration. The configuration should look as follows:

```
…
<bindings>
  <wsHttpBinding>
   <binding name="wsHttpEndpointBinding">
          <security mode="TransportWithMessageCredential">
            <transport clientCredentialType="None" />
            <message clientCredentialType="UserName" />
          </security>
  </binding>
</wsHttpBinding>
</bindings>
<services>
  <service behaviorConfiguration="ServiceBehavior"
name="Service">
    <endpoint address="" binding="wsHttpBinding"
      bindingConfiguration="wsHttpEndpointBinding"
      name="wsHttpEndpoint" contract="IService">
     <!--<identity>
        <dns value="" />
      </identity>-->
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />
  </service>
</services>
```

Step 7 – Configure the Service to Publish Metadata Securely

In this step, you configure your WCF service to publish and secure the metadata. By publishing the metadata, you will allow your client to add a reference to your WCF service.

1. In the Configuration Editor, expand the **Services** node and then expand **Endpoints**.
2. Select the second endpoint created **[Empty Name]** and then set the **Name** attribute to **MexHttpsBindingEndpoint**.
3. Set the **Binding** attribute to **mexHttpsBinding** by choosing this option from the drop-down list.

4. In the Configuration Editor, on the **File** menu, click **Save**.
5. In Visual Studio, verify your configuration in Web.config. The configuration should look as follows:

```
...
<services>
      <service behaviorConfiguration="ServiceBehavior"
name="Service">
            <endpoint address="" binding="wsHttpBinding"
                  bindingConfiguration="wsHttpEndpointBinding"
                  name="wsHttpEndpoint" contract="IService">
            </endpoint>

        <endpoint address="mex" binding="mexHttpsBinding"
bindingConfiguration=""
            name="MexHttpsBindingEndpoint"
contract="IMetadataExchange" />

    </service>
</services>
...
```

6. In the Configuration Editor, expand the **Advanced** node, and then expand the **Service Behaviors** node.
7. Select the **serviceMetadata** node.
8. Set the **HttpGetEnabled** attribute to **False** and the **HttpsGetEnabled** attribute to **True**.
9. In the Configuration Editor, on the **File** menu, click **Save**.
10. In Visual Studio, verify your configuration in App.config. The configuration should look as follows:

```
<behaviors>
  <serviceBehaviors>
      <behavior name="ServiceBehavior">
          <serviceMetadata httpGetEnabled="false"
httpsGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
  </serviceBehaviors>
</behaviors>
```

Step 8 – Configure the Membership Provider for Username Authentication

In this step, you configure the SQL Server Membership Provider to use username authentication.

1. In the web.config file, replace the existing single <connectionStrings/> element with the following to point to your membership database:

```
<connectionStrings>
  <add name="MyLocalSQLServer"
       connectionString="Initial Catalog=aspnetdb;
```

```
      data source=.\sqlexpress;Integrated Security=SSPI;" />
  </connectionStrings>
```

2. Add a <**membership**> element inside the **<system.web>** element as shown in the following example. Notice that the use of the <clear/> element prevents the default provider from being loaded and then never used.

```
...
<system.web>
  ...
  <membership defaultProvider="MySqlMembershipProvider" >
    <providers>
      <clear/>
      <add name="MySqlMembershipProvider"
           connectionStringName="MyLocalSQLServer"
           applicationName="MyAppName"
           type="System.Web.Security.SqlMembershipProvider" />
    </providers>
  </membership>
</system.web>
...
```

3. Save the Web.Config file, to ensure that the changes do not get lost during the following steps.
4. In the Configuration Editor, expand the **Advanced** node, and then expand the Service Behaviors folder.
5. Select the **ServiceBehavior** node.
6. In the **Behavior: ServiceBehavior** section, click **Add**.
7. In the **Adding Behavior Element Extension Sections** dialog box, select **serviceCredentials** and then click **Add**.
8. In the **Configuration** section, under **Service Behavior**, select **serviceCredentials**.
9. Set the **UsernamePasswordValidationMode** attribute to **MembershipProvider** by choosing this option from the drop-down list.
10. Set the **MembershipProviderName** attribute to **MySqlMembershipProvider** by choosing this option from the drop-down list.
11. In the Configuration Editor, on the **File** menu, click **Save**.
12. In Visual Studio, verify your configuration. The configuration should look as follows:

```
…
<behaviors>
  <serviceBehaviors>
    <behavior name="ServiceBehavior">
     <serviceMetadata httpGetEnabled="false"
httpsGetEnabled="true" />
      <serviceDebug includeExceptionDetailInFaults="false" />
      <serviceCredentials>
        <userNameAuthentication
userNamePasswordValidationMode="MembershipProvider"
          membershipProviderName="MySqlMembershipProvider" />
```

```
      </serviceCredentials>
    </behavior>
  </serviceBehaviors>
</behaviors>
…
```

Step 9 – Create a User in the User Store

In this step, you create a user that the test client will use to log into the WCF service.

1. In the Solution Explorer, choose the WCF service project, and then on the **Website** menu, click **ASP.NET Configuration**.
2. On the ASP.NET Web Site Administration Tool page, on the **Security** tab, click the **Select authentication type** link.
3. On the page that appears, select the **From the internet** radio button and then click **Done**.
4. Click the **Create user** link.
5. On the Create User page, enter the details of the user you want to create in the SQL store and then click **Create User**.
   If the procedure is successful, a new user will be created. By default, you will need to create a password of at least seven characters, with one character that is not alphanumeric.

Step 10 – Create a Test Client Application

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.

Step 11 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service to the client.

1. Right-click your client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF service (for example, https://localhost/WCFTestService/Service.svc) and then click **Go**.
3. In the **Namespace** field, change ServiceReference1 to **WCFService** and then click **OK**.
   ,A reference to WCFTestService should appear beneath Service References In your **client** project.

Step 12 – Test the Client and WCF Service

In this step, you access the WCF service, pass the user credentials, and make sure that the authentication works through a secure channel (HTTPS).

1. In your client project, drag a Button control onto your form.
2. Double-click the Button control to show the underlying code.
3. Create an instance of the proxy, pass the credentials of the user created in step 12, and then call the **GetData** operation of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                  WCFTestService.ServiceClient();
      myService.ClientCredentials.UserName.UserName = "username";
      myService.ClientCredentials.UserName.Password = "p@ssw0rd";
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5.
   When you click the button on the form, the message "**You entered: 123**" should appear.

Additional Resources

- For more information on how to work with temporary certificates, see "How to: Create Temporary Certificates for Use During Development" at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on how to view certificates by using the Microsoft Management Console (MMC) snap in, see "How to: View Certificates with the MMC Snap-in" at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see "Differences Between Service Certificate Validation Done by Internet Explorer and WCF" at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see "Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security" at http://msdn2.microsoft.com/en-us/library/aa702579.aspx

How To – Use WsHttpBinding with Windows Authentication and Message Security in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Windows Forms
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of using Windows Authentication over **wsHttpBinding** binding using transport security. The article shows you how to configure WCF and test the service with a sample WCF client. This configuration is suited for intranet scenarios where there is a domain controller that will issue Kerberos tickets to provide message protection. There is no need for certificate installation in these scenarios.

Contents

Objectives

- Create a WCF service hosted in Internet Information Services (IIS).
- Expose the WCF service with message security.
- Learn how to use Windows tokens to encrypt and sign your messages.
- Learn why you need service principle names (SPNs) and how to create them.
- Call the service from a test client.

Overview

In the scenario described in this How To article, users are authenticated by using Windows Authentication. This approach is suited for scenarios in which your users have

domain credentials. The **wsHttpBinding** binding is used in order to provide support for message-based security, reliable messaging, and transactions, while also allowing the possibility that legacy clients can consume the service. Message security is used to encrypt and sign your messages while allowing for intermediaries to re-route your message as needed. In general, you should always use WCF transport security unless you need the additional flexibility that message security affords you. The scenario described in this How To article uses a Kerberos ticket and a domain controller as the broker for authentication. This mechanism avoids the need for certificates that would otherwise be required for message protection.

Message security is used instead of Transport security in order to support:

- Partial encryption of the message.
- Message security that extends beyond a single point-to-point communication channel.
- Flexibility to use other transports such as Transmission Control Protocol (TCP) or named pipes.

Summary of Steps

- Step 1 – Create a Sample WCF Service
- Step 2 – Configure the WCF Service to Use wsHttpBinding with Windows Authentication and Message Security
- Step 3 – Create a Test Client
- Step 4 – Add a WCF Service Reference to the Client
- Step 5 – Test the Client and WCF Service

Step 1 – Create a Sample WCF Service

In this step, you create a WCF service in Visual Studio, hosted in an IIS virtual directory.

1. In Visual Studio, on the menu, select **File** > **New Web Site.**
2. In the **New Web Site** dialog box, under **Templates**, select **WCF Service**. Make sure that the **Location** is set to **Http**.
3. In the **New Web Site dialog** box, set the new Web site address as **https://localhost/ WCFTestService** and then click **OK**.

Step 2 – Configure the WCF Service to Use wsHttpBinding with Windows Authentication and Message Security

By default, **wsHttpBinding** is configured with message security and Windows Authentication, so you don't have to actually do anything in this step but verify that your configuration looks as follows:

```
…
<services>
  <service name="Service" behaviorConfiguration="ServiceBehavior">
    <!-- Service Endpoints -->
    <endpoint address="" binding="wsHttpBinding" contract="IService">
      <identity>
        <dns value="localhost"/>
      </identity>
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange"/>
  </service>
</services>
…
```

Note: Remove the <identity>, which gets added automatically, to prevent run-time errors ("The token provider cannot get tokens for target.") when testing with the client application.

Step 3 – Create a Test Client

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK** button.

Step 4 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service.

1. Right-click your client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF service; for example, http://localhost/WCFTestService/Service.svc
3. In the **Namespace** field, change ServiceReference1 to **WCFTestService** and then click **OK**.
4. Click **OK**.
   A reference to WCFTestService should now appear beneath Service References in your Client project.

Step 5 – Test the Client and WCF Service

In this step, you access the WCF service from the client and make sure that it works.

1. In your client project, drag a **Button** control onto your form.
2. Double-click the **Button** control to show the underlying code.

3. In the code behind the button click, create an instance of the proxy and call the **GetData** operation of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
        {
            WCFTestService.ServiceClient myService = new
WCFTestService.ServiceClient();
            MessageBox.Show(myService.GetData(123));
            myService.Close();
        }
```

4. Right-click the client project and then click **Set as Startup Project**.
5. Run the client application by pressing F5 or Ctrl+F5. When you click the button on the form, the message “You entered: 123” should appear.

Additional Considerations

By default, **negotiateServiceCredentials** is set to true, but this can be set to false if you do not support the **WS-Trust** or **WS-SecureConversation** specifications. Setting this value to false will also make your service interoperable with Simple Object Access Protocol (SOAP) stacks that implement the Kerberos token profile from OASIS. The following is a configuration sample for setting the **negotiateServiceCredentials** to false:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="WsHttpBindingConfig">
      <security>
        <message negotiateServiceCredential="false" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
<services>
  <service behaviorConfiguration="ServiceBehavior"
      name="Service">
    <endpoint address="" binding="wsHttpBinding"
      bindingConfiguration="WsHttpBindingConfig"
      contract="IService">
    </endpoint>
  </service>
</services>
…
```

If this property is set to false, the service account must be associated with a service principal name (SPN). To do this, run the service under the NETWORK SERVICE or LOCAL SYSTEM account. Alternatively, use the SetSpn.exe tool to create an SPN for the service account. In either case, the client must use the correct SPN; you can specify the SPN value in the configuration as follows:

```
…
<services>
  <service behaviorConfiguration="ServiceBehavior"
    name="Service">
```

```
    <endpoint address="" binding="wsHttpBinding"
bindingConfiguration="WsHttpBindingConfig"
      contract="IService">
      <identity>
        <servicePrincipalName value="Host/<MachineName>" />
      </identity>
    </endpoint>
  </service>
</services>
…
```

For more information, see "MessageSecurityOverHttp.NegotiateServiceCredential Property" at http://msdn2.microsoft.com/en-us/library/system.servicemodel.messagesecurityoverhttp.negotiateservicecredential.aspx.

Deployment Considerations

First, make sure that you have a domain controller in the network to authenticate the client and service.

If you are using a custom domain account in the identity pool for your WCF application in IIS, execute the following steps:

1. Create an SPN for Kerberos to be able to authenticate the client.
   By default, **"NT AUTHORITY\NETWORK SERVICE"** maps to the computer account, so Kerberos works with this account. Go to the domain controller of you domain and create an SPN mapping to the custom domain account. The SPN has the format HTTP/Machinename or HTTP/fullyQualifiedNameofMachine. The examples below show how to create an SPN and map it to the custom domain account myAccount:
   - Setspn –a HTTP/machinename myAccount
   - Setspn –a HTTP/machinename.code.com myAccount
2. Give permissions to the domain account to access C:\windows\temp. If this is not done, you will not be able to create a service reference or a proxy client with svcutil.exe. Perform the following steps:
   - Open Microsoft Windows® Explorer and navigate to the Windows folder.
   - Right-click the Temp directory and then click the **Security** tab.
   - In the user list, click **Add** and then enter the domain account name in the format domain\accountName.
   - Clear all permissions and then click **Advanced**.
   - Double-click the account.
   - In the list of permissions, select the **List Folder / Read Data** and **Delete** permissions.

Additional Resources:

- For more information on Windows Authentication, see “Explained: Windows Authentication in ASP.NET 2.0” at http://msdn2.microsoft.com/en-us/library/aa480475.aspx
- For more information on debugging authentication errors, see “Debugging Windows Authentication Errors” at http://msdn2.microsoft.com/en-us/library/bb463274.aspx
- For more information on security authentication best practices, see “Best Practices for Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms731059.aspx
- For additional information on message security, see “Message Security in WCF” at http://msdn2.microsoft.com/en-us/library/ms733137.aspx

How To – Use wsHttpBinding with Windows Authentication and Transport Security in WCF Calling from Windows Forms

Applies to

- Microsoft® Windows Communication Foundation (WCF) 3.5
- Microsoft Visual Studio® 2008

Summary

This How To article walks you through the process of using Windows Authentication over **wsHTTPBinding** using transport security. The article shows you how to configure WCF, create and install the necessary certificate, and test the service with a sample WCF client.

Contents

Objectives

- Create a WCF service hosted in Internet Information Services (IIS).
- Configure the service to use the Secure Sockets Layer (SSL) protocol.
- Create and configure a certificate for the service.
- Expose the WCF service over **wsHttpBinding**.
- Call the service from a test client.

Overview

Windows Authentication is suited for scenarios in which your users have domain credentials. In the scenario described in this How To article, users are authenticated by Windows Authentication. The **wsHttpBinding** binding is used in order to provide support for message-based security, reliable messaging, and transactions, while also allowing the possibility that legacy clients can consume the service. WCF transport security is used to support a secure communication channel in a point-to-point scenario. In general, you should always use transport security unless you need the additional flexibility 2that message security affords you. For

example, you would use message security for scenarios in which there are intermediaries who need to inspect and re-route the message.

In order to use SSL for transport security, you first need to install a service certificate. This certificate will be used to encrypt and protect the communication channel. You then configure the WCF service and IIS to use SSL. You will set the **clientCredentialType** attribute to Windows on the **wsHttpBinding** binding in order to configure the WCF service to use Windows Authentication. You will then configure and expose a **mexHttpsBinding** endpoint to expose the service metadata to the client securely. This metadata allows the client to generate a proxy and call the service.

Summary of Steps

- Step 1 – Create and Install a Service Certificate for Transport Security
- Step 2 – Create a Sample WCF Service Project with SSL
- Step 3 – Configure the Virtual Directory to Require SSL
- Step 4 – Configure wsHttpBinding for Windows Authentication and Transport Security
- Step 5 – Configure the Service to Publish Metadata Securely
- Step 6 – Create a Test Client Application
- Step 7 – Add a WCF Service Reference to the Client
- Step 8 – Test the Client and WCF Service

Step 1 – Create and Install a Service Certificate for Transport Security

In this step, you create a temporary service certificate and install it in the local store. This certificate will be used to establish an SSL connection between the client and the WCF service.

Creating and installing the certificate is outside the scope of this How To article. For instructions on how to do this, see "How To – Create and Install Temporary Certificates in WCF for Transport Security during Development" and follow steps 1 through 4.

Note: Temporary certificate should be used for development and testing purposes only. For actual production deployment, you will need to obtain a valid certificate from a certificate authority (CA).

Step 2 – Create a Sample WCF Service Project with SSL

In this step, you create a WCF service in Visual Studio and enable SSL.

1. In Visual Studio, on the menu, select **File** > **New Web Site**.
2. In the **New Web Site** dialog box, under **Templates**, select **WCF Service**. Make sure that the **Location** is set to **Http** and then click **Browse**.
3. In the **Choose Location dialog** box, click **Local IIS**.
4. At the bottom of the dialog box, select the **Use Secure Sockets Layer** check box, and then click **Open**.
5. In the **New Web Site** dialog box, set the new Web site address as **https://localhost/WCFTestService** and then click **OK**.

Note: Because the SSL port might not be configured by default on the IIS, it might throw errors while creating the WCF service. To prevent this, open the IIS Manager, right-click **Default Web Site**, and then click **Properties** option. In the **Default Web Site Properties** dialog box, click the **Web Site** tab and make sure that the **SSL port:** is set to 443.

Step 3 – Configure the Virtual Directory to Require SSL

In this step, you configure the virtual directory hosting the WCF service to use SSL.

1. Click Start then Run and then type inetmgr to open the Internet Information Services manager.
2. In the Internet Information Services Manager dialog box, expand the **(local computer),** expand the **Web Sites** node, and then expand the **Default Web Site** node.
3. Right-click your virtual directory (WCFTestService) and then click **Properties.**
4. In the **Properties** dialog box, click the **Directory Security** tab, and then in the **Secure Communication** section, click **Edit**.
5. In the **Secure communications** dialog box, select the **Require secure channel (SSL)** check box.
6. in the **Secure communications** dialog box, click **OK**.
7. In the **Properties** dialog box, click **O**K.

Step 4 – Configure wsHttpBinding for Windows Authentication and Transport Security

In this step, you configure the WCF service to use Windows Authentication and transport security.

1. Right-click the Web.config file of the WCF service and then click **Edit WCF Configuration**.
2. If you do not see the **Edit WCF Configuration** option, on the **Tools** menu, click **WCF Service Configuration Editor**. Close the **WCF Service Configuration Editor** tool that appears. The option should now appear on the web.config context menu.
3. In the configuration editor, in the **Configuration** section, expand **Service** and then expand **Endpoints**.
4. Select the first node **[Empty Name]** and set the **Name** attribute to **wsHttpEndpoint**.
5. Click the **Identity** tab and delete the **Dns** attribute value, which by default is set to "**localhost**".
6. In the configuration editor, select the Bindings folder.
7. In the **Bindings** section, select **New Binding Configuration.**
8. In the **Create a New Binding** dialog box, select **wsHttpBinding**.
9. Click **OK**.
10. Set the **Name** of the binding configuration to some logical and recognizable name; for example, **wsHttpEndpointBinding**.
11. Click the **Security** tab.
12. Set the **Mode** attribute to **Transport** by choosing this option from the drop-down list.
13. Make sure that the **TransportClientCredentialType** is set to **Windows**, which is the default setting.
14. In the **Configuration** section, select the **wsHttpEndpoint** node.

15. Set the **BindingConfiguration** attribute to **wsHttpEndpointBinding** by choosing this option from the drop-down list.
    This associates the binding configuration setting with the binding.
16. In the configuration editor dialog box, on the **File** menu, click **Save**.
17. In Visual Studio, open your configuration and comment out the identity element. It should look as follows:

```
<!--<identity>
  <dns value="" />
</identity>-->
```

18. In Visual Studio, verify your configuration. The configuration should look as follows:

```
…
<bindings>
  <wsHttpBinding>
    <binding name="wsHttpEndpointBinding">
      <security mode="Transport">
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
<services>
  <service behaviorConfiguration="ServiceBehavior"
name="Service">
    <endpoint address="" binding="wsHttpBinding"
      bindingConfiguration="wsHttpEndpointBinding"
      name="wsHttpEndpoint" contract="IService">
      <!--<identity>
        <dns value="" />
      </identity>-->
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />
  </service>
</services>
…
```

Step 5 – Configure the Service to Publish Metadata Securely

In this step, you configure your WCF service to publish and secure the metadata. By publishing the Metadata, you allow your client to add a reference to your WCF service.

1. In the configuration editor, expand the **Service** node, and then expand **Endpoints**.
2. Select the second endpoint created **[Empty Name]** and then set the **Name** attribute to "**MexHttpsBindingEndpoint**".
3. Set the **Binding** attribute to **mexHttpsBinding**.
4. In the configuration editor dialog box, on the **File** menu, click **Save**.
5. In Visual Studio, verify your configuration in App.config. The configuration should look as follows.

```
…
<services>
      <service behaviorConfiguration="ServiceBehavior"
```

6. In the configuration editor, expand the **Advanced** node, and then expand the **Service Behaviors** node.
7. Expand the **ServiceBehavior** node and then select the **serviceMetadata** node.
8. Set the **HttpGetEnabled** attribute to **False** and the **HttpsGetEnabled** attribute to **True**.
9. In the configuration editor dialog box, on the **File** menu, click **Save**.
10. In Visual Studio, verify your configuration in App.config. The configuration should look as follows.

```
…
<behaviors>
  <serviceBehaviors>
      <behavior name="ServiceBehavior">
          <serviceMetadata httpGetEnabled="false"
httpsGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
  </serviceBehaviors>
</behaviors>
…
```

Step 6 – Create a Test Client Application

In this step, you create a Windows Forms application to test the WCF service.

1. Right-click your solution, click **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Templates** section, select **Windows Forms Application**.
3. In the **Name** field, type **Test Client** and then click **OK**.

Step 7 – Add a WCF Service Reference to the Client

In this step, you add a reference to your WCF service so that your client can call the service.

1. Right-click your Client project and then click **Add Service Reference**.
2. In the **Add Service Reference** dialog box, set the URL to your WCF service – for example, https://<<YourMachineName>>/WCFTestService/Service.svc – and then click **Go** .
3. In the **Namespace** field, change ServiceReference1 to **WCFTestService** and then click **OK**.

A reference to WCFTestService should appear beneath Service References in your Client project.

Note: If the machine name used in the **Add Service Reference** dialog does not match the certificate name, you will get an error when trying to add the reference. You can resolve this by ensuring that the certificate name matches the machine name used in this URL.

Step 8 – Test the Client and WCF Service

In this step, you access the WCF service, pass the user credentials, and make sure that the authentication works through a secure channel (HTTPS).

1. In your Client project, drag a **Button** control onto your form.
2. Double-click the **Button** control to show the underlying code.
3. In the code behind the button click, create an instance of the proxy and call the **GetData** operation of your WCF service. The code should look as follows:

```
private void button1_Click(object sender, EventArgs e)
{
      WCFTestService.ServiceClient myService = new
                  WCFTestService.ServiceClient();
      MessageBox.Show(myService.GetData(123));
      myService.Close();
}
```

4. Right-click the Client project and then click **Set as Startup Project**.
5. Run the Client application by pressing F5 or Ctrl+F5. When you click the button on the form, the message "**You entered: 123**" should appear.

Additional Resources

- For more information on WCF Transport Layer Security using **wsHttpBinding** and SSL, see "WCF Transport Layer Security using wsHttpBinding and SSL" at http://www.codeproject.com/KB/WCF/WCF.aspx
- For more information on how to work with temporary certificates, see "How to: Create Temporary Certificates for Use During Development" at http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- For more information on how to view certificates by using the Microsoft Management Console (MMC) snap in, see "How to: View Certificates with the MMC Snap-in" at http://msdn2.microsoft.com/en-us/library/ms788967.aspx
- For more information on differences in certificate validation between Microsoft Internet Explorer and WCF, see "Differences Between Service Certificate Validation Done by Internet Explorer and WCF" at http://msdn2.microsoft.com/en-us/library/aa702599.aspx
- For more information on differences in certificate validation between protocols, see "Certificate Validation Differences Between HTTPS, SSL over TCP, and SOAP Security" at http://msdn2.microsoft.com/en-us/library/aa702579.aspx

WCF Security Resources

Getting Started

Microsoft

- Microsoft® MSDN® Library – "Fundamental Windows Communication Foundation Concepts" – http://msdn2.microsoft.com/en-us/library/ms731079.aspx
- MSDN Library – "Windows Communication Foundation Security" – http://msdn.microsoft.com/en-us/library/ms732362.aspx
- WCF Security Documentation – "Security Overview" – http://msdn.microsoft.com/en-us/library/ms735093.aspx

Community

- DevX.com – "Fundamentals of WCF Security," by Michèle Leroux Bustamante – http://www.devx.com/codemag/Article/33342
- TheServer Side.NET – "WCF Security Learning Guide," by Brent Sheets – http://www.theserverside.net/tt/articles/showarticle.tss?id=WCFSecurityLearningGuide

Articles

Microsoft

- MSDN Library – "The .NET Developer's Guide to Identity," by Keith Brown – http://msdn2.microsoft.com/en-us/library/aa480245.aspx
- *MSDN Magazine* – "Identity: Secure Your ASP.NET Apps And WCF Services With Windows CardSpace, "by Michèle Leroux Bustamante – http://msdn2.microsoft.com/en-us/magazine/cc163434.aspx
- *MSDN Magazine* – "IIS 7.0: Extend Your WCF Services Beyond HTTP With WAS," by Dominick Baier, Christian Weyer, and Steve Maine – http://msdn2.microsoft.com/en-us/magazine/cc163357.aspx
- *MSDN Magazine* – "Security Briefs: Exploring Claims-Based Identity," by Keith Brown – http://msdn2.microsoft.com/en-us/magazine/cc163366.aspx
- *MSDN Magazine* – "Security Briefs: Limited User Problems and Split Knowledge," By Keith Brown – http://msdn2.microsoft.com/en-us/magazine/cc163531.aspx
- *MSDN Magazine* – "Security Briefs: Security in Windows Communication Foundation," by Keith Brown – http://msdn2.microsoft.com/en-us/magazine/cc163570.aspx
- *MSDN Magazine* – "Service Station: WCF Messaging Fundamentals," by Aaron Skonnard – http://msdn2.microsoft.com/en-us/magazine/cc163447.aspx

Community

- DevX.com – "Fundamentals of WCF Security," by Michèle Leroux Bustamante – http://www.devx.com/codemag/Article/33342
- TheServerSide.NET – "Building a Claims-Based Security Model in WCF," by Michèle Leroux Bustamente – http://www.theserverside.net/tt/articles/showarticle.tss?id=ClaimsBasedSecurityModel
- TheServerSide.NET – "Building a Claims-Based Security Model in WCF – Part 2," by Michèle Leroux Bustamente – http://www.theserverside.net/news/thread.tss?thread_id=45499
- TheServerSide.NET – "Securing Your WCF service," by William Tay – http://www.theserverside.net/tt/articles/showarticle.tss?id=SecuringWCFService
- TopXML – "BizTalk and WCF: Part II, Security Patterns," by Richard Seroter – http://www.topxml.com/code/cod-72_10192_biztalk-and-wcf-part-ii-security-patterns.aspx

Blogs

Microsoft

- J.D. Meier – http://blogs.msdn.com/jmeier/archive/tags/WCF/default.aspx
- Kim Cameron – http://www.identityblog.com/
- Kenny Wolf – http://kennyw.com/category/indigo/
- Nicholas Allen – http://blogs.msdn.com/drnick/
- Ralph Squillace – http://blogs.msdn.com/ralph.squillace
- Steve Maine – http://hyperthink.net/blog/
- Tomasz Janczuk –http://www.pluralsight.com/blogs/tjanczuk/
- Vittorio Bertocci – http://blogs.msdn.com/vbertocci/
- Wenlong Dong – http://blogs.msdn.com/wenlong

Community

- Dominick Baier – http://www.leastprivilege.com/
- Keith Brown – http://www.pluralsight.com/blogs/keith
- Michèle Leroux Bustamante – http://www.thatindigogirl.com/
- Thomas Restrepo – http://www.winterdom.com/weblog

Channel9

Podcasts

- ARCast – "Secure, Reliable Transacted Messaging with WCF (Part 1)" – http://channel9.msdn.com/Showpost.aspx?postid=173405
- ARCast – "Secure, Reliable Transacted Messaging with WCF (Part 2)" – http://channel9.msdn.com/Showpost.aspx?postid=173830

ARCast.TV

- ARCast.TV – "WCF Session Behavior from Slovenia" – http://channel9.msdn.com/Showpost.aspx?postid=347826

Videos

- Vittorio Bertocci: "WS-Trust – Under the Hood" – http://channel9.msdn.com/tags/WS-Trust

Tags

- WCF tag – http://channel9.msdn.com/tags/WCF

Documentation

Overview

- Architecture – http://msdn2.microsoft.com/en-us/library/ms733128.aspx
- Concepts – http://msdn2.microsoft.com/en-us/library/ms731069.aspx
- Distributed Application Security – http://msdn2.microsoft.com/en-us/library/ms731204.aspx
- Security Architecture – http://msdn2.microsoft.com/en-us/library/ms788756.aspx
- Security Overview – http://msdn2.microsoft.com/en-us/library/ms735093.aspx
- WCF Security Terminology – http://msdn2.microsoft.com/en-us/library/ms731846.aspx

Guidance

- Best Practices for Queued Communication – http://msdn2.microsoft.com/en-us/library/ms731093.aspx
- Best Practices for Reliable Sessions – http://msdn2.microsoft.com/en-us/library/ms733795.aspx
- Security Guidance and Best Practices – http://msdn2.microsoft.com/en-us/library/ms731983.aspx

Scenarios

- Common Security Scenarios – http://msdn2.microsoft.com/en-us/library/ms730301.aspx
- Identity Model Scenarios – http://msdn2.microsoft.com/en-us/library/ms729851.aspx

Threats and Countermeasures

- Threats and Countermeasures – http://msdn2.microsoft.com/en-us/library/ms731086.aspx

Topics

- Auditing – http://msdn2.microsoft.com/en-us/library/ms731669.aspx
- Authentication – http://msdn2.microsoft.com/en-us/library/ms733082.aspx

- Authorization – http://msdn2.microsoft.com/en-us/library/ms733071.aspx
- Authorization Mechanisms – http://msdn2.microsoft.com/en-us/library/ms733106.aspx
- Bindings and Security – http://msdn2.microsoft.com/en-us/library/ms731172.aspx
- Claims-Based Authorization – http://msdn2.microsoft.com/en-us/library/ms729851.aspx
- Configuration Schema – http://msdn2.microsoft.com/en-us/library/ms731734.aspx
- Federation and Issued Tokens – http://msdn2.microsoft.com/en-us/library/ms730908.aspx
- Hosting – http://msdn2.microsoft.com/en-us/library/ms729846.aspx
- Impersonation and Delegation – http://msdn2.microsoft.com/en-us/library/ms730088.aspx
- Impersonation with Transport Security – http://msdn2.microsoft.com/en-us/library/ms788971.aspx
- Message Security in WCF – http://msdn2.microsoft.com/en-us/library/ms733137.aspx
- Partial Trust – http://msdn2.microsoft.com/en-us/library/bb412175.aspx
- Reliable Sessions Overview – http://msdn2.microsoft.com/en-us/library/ms733136.aspx
- SAML Tokens and Claims – http://msdn2.microsoft.com/en-us/library/ms733083.aspx
- Security Capabilities with Custom Bindings – http://msdn2.microsoft.com/en-us/library/ms733121.aspx
- Secure Conversations and Secure Sessions – http://msdn2.microsoft.com/en-us/library/ms731107.aspx
- Secure Sockets Layer (SSL) – http://msdn2.microsoft.com/en-us/library/ms734679.aspx
- Securing Services and Clients – http://msdn2.microsoft.com/en-us/library/ms734736.aspx
- Transport Security Overview – http://msdn2.microsoft.com/en-us/library/ms729700.aspx
- X.509 Certificates – http://msdn2.microsoft.com/en-us/library/ms731899.aspx

How To Articles

- How to: Audit Windows Communication Foundation Security Events – http://msdn2.microsoft.com/en-us/library/ms734737.aspx
- How to: Configure Credentials on a Federation Service – http://msdn2.microsoft.com/en-us/library/ms730131.aspx
- How to: Configure a Local Issuer – http://msdn2.microsoft.com/en-us/library/aa347715.aspx
- How to: Configure a Port with an SSL Certificate – http://msdn2.microsoft.com/en-us/library/ms733791.aspx
- How to: Consistently Reference X.509 Certificates – http://msdn2.microsoft.com/en-us/library/aa702627.aspx
- How to: Create a Custom Binding Using the SecurityBindingElement – http://msdn2.microsoft.com/en-us/library/ms730305.aspx
- How to: Create a Federated Client – http://msdn2.microsoft.com/en-us/library/ms731690.aspx
- How to: Create a Secure Session – http://msdn2.microsoft.com/en-us/library/ms733783.aspx
- How to: Create a Security Token Service – http://msdn2.microsoft.com/en-us/library/ms733095.aspx

- How to: Create a Stateful Security Context Token for a Secure Session – http://msdn2.microsoft.com/en-us/library/ms731814.aspx
- How to: Create a Supporting Credential – http://msdn2.microsoft.com/en-us/library/ms734664.aspx
- How to: Create Temporary Certificates for Use During Development – http://msdn2.microsoft.com/en-us/library/ms733813.aspx
- How to: Create a WSFederationHttpBinding – http://msdn2.microsoft.com/en-us/library/aa347982.aspx
- How to: Create a Custom Reliable Session Binding with HTTPS – http://msdn2.microsoft.com/en-us/library/ms735116.aspx
- How to: Disable Encryption of Digital Signatures – http://msdn2.microsoft.com/en-us/library/aa738768.aspx
- How to: Disable Secure Sessions on a WSFederationHttpBinding – http://msdn2.microsoft.com/en-us/library/ms731827.aspx
- How to: Enable Message Replay Detection – http://msdn2.microsoft.com/en-us/library/ms733063.aspx
- How to: Exchange Messages Within a Reliable Session – http://msdn2.microsoft.com/en-us/library/ms733049.aspx
- How to: Impersonate a Client on a Service – http://msdn2.microsoft.com/en-us/library/ms731090.aspx
- How to: Make X.509 Certificates Accessible to WCF – http://msdn2.microsoft.com/en-us/library/aa702621.aspx
- How to: Obtain a Certificate (WCF) – http://msdn2.microsoft.com/en-us/library/aa702761.aspx
- How to: Restrict Access with the PrincipalPermissionAttribute Class – http://msdn2.microsoft.com/en-us/library/ms731200.aspx
- How to: Retrieve the Thumbprint of a Certificate – http://msdn2.microsoft.com/en-us/library/ms734695.aspx
- How to: Secure Messages within Reliable Sessions – http://msdn2.microsoft.com/en-us/library/aa702650.aspx
- How to: Secure a Service with Windows Credentials – http://msdn2.microsoft.com/en-us/library/ms734673.aspx
- How to: Secure a Service with an X.509 Certificate – http://msdn2.microsoft.com/en-us/library/ms788968.aspx
- How to: Set Up a Signature Confirmation – http://msdn2.microsoft.com/en-us/library/ms730328.aspx
- How to: Set a Max Clock Skew – http://msdn2.microsoft.com/en-us/library/aa738468.aspx
- How to: Specify the Certificate Authority Certificate Chain Used to Verify Signatures (WCF) – http://msdn2.microsoft.com/en-us/library/aa738659.aspx
- How to: Use the ASP.NET Authorization Manager Role Provider with a Service – http://msdn2.microsoft.com/en-us/library/ms734774.aspx
- How to: Use the ASP.NET Membership Provider – http://msdn2.microsoft.com/en-us/library/ms731049.aspx

- How to: Use the ASP.NET Role Provider with a Service – http://msdn2.microsoft.com/en-us/library/aa702542.aspx
- How to: Use a Custom User Name and Password Validator – http://msdn2.microsoft.com/en-us/library/aa702565.aspx
- How to: Use Multiple Security Tokens of the Same Type – http://msdn2.microsoft.com/en-us/library/bb885138.aspx
- How to: Use Transport Security and Message Credentials – http://msdn2.microsoft.com/en-us/library/ms789011.aspx
- How to: View Certificates with the MMC Snap-in – http://msdn2.microsoft.com/en-us/library/ms788967.aspx

Guides

Community

- dasblonde.net – "WCF Security Fundamentals," by Michèle Leroux Bustamante – http://www.dasblonde.net/downloads/sessions/WCFSecurityFundamentals.pdf
- TheServer Side.NET – "WCF Security Learning Guide," by Brent Sheets – http://www.theserverside.net/tt/articles/showarticle.tss?id=WCFSecurityLearningGuide

Posts

Microsoft

- Alexander Strauss – "WCF – Let's Start The Dialogue" – http://blogs.msdn.com/astrauss/archive/2006/10/27/wcf-let-s-start-the-dialogue.aspx
- Alik Levine – "How To Consume WCF Using AJAX Without ASP.NET" – http://blogs.msdn.com/alikl/archive/2008/02/18/how-to-consume-wcf-using-ajax-without-asp-net.aspx

Community

- Dominick Baier – "Using IdentityModel: Authorization Policies, Context and Claims Transformation" – http://www.leastprivilege.com/UsingIdentityModelAuthorizationPoliciesContextAndClaimsTransformation.aspx
- Dominick Baier – "Using IdentityModel: Creating Custom Claim Sets" – http://www.leastprivilege.com/UsingIdentityModelCreatingCustomClaimSets.aspx
- Dominick Baier – "Using IdentityModel: Typical Operations on Claim Sets" – http://www.leastprivilege.com/UsingIdentityModelTypicalOperationsOnClaimSets.aspx

- Dominick Baier – "Using IdentityModel: Windows and X509Certificate Claim Sets" – http://www.leastprivilege.com/UsingIdentityModelWindowsAndX509CertificateClaimSets.aspx
- Dominick Baier – "Using IdentityModel: Inspecting Claim Sets" – http://www.leastprivilege.com/UsingIdentityModelInspectingClaimSets.aspx
- Dominick Baier – "Using IdentityModel: Claim Sets" – http://www.leastprivilege.com/UsingIdentityModelClaimSets.aspx
- Dominick Baier – "Using IdentityModel: Claims" – http://www.leastprivilege.com/UsingIdentityModelClaims.aspx
- Dominick Baier – "Be careful with ServiceAuthorizationManager.CheckAccess()" – http://www.leastprivilege.com/BeCarefulWithServiceAuthorizationManagerCheckAccess.aspx
- Dominick Baier – "UserName SupportingToken in WCF" – http://www.leastprivilege.com/UserNameSupportingTokenInWCF.aspx
- Paolo Pialorsi – "WCF Custom Authentication and Impersonation" – http://weblogs.asp.net/paolopia/archive/2005/12/08/432658.aspx
- Tomas Restrepo – "WCF Configuration Complexity" – http://www.winterdom.com/weblog/CommentView,guid,d8954fbc-3c04-441c-8d81-9e98e70a8580.aspx

patterns & practices

- WCF Security Guidance Package – http://www.codeplex.com/servicefactory/Release/ProjectReleases.aspx?ReleaseId=8814

Product Support Services (PSS)

- WCF Troubleshooting Quickstart – http://msdn2.microsoft.com/en-us/library/aa702636.aspx

Samples

Microsoft

- Basic Windows Communication Foundation Technology Samples – http://msdn2.microsoft.com/en-us/library/ms752239.aspx

- Windows Communication Foundation Samples – http://msdn2.microsoft.com/en-us/library/ms751514.aspx

Community

- Paolo Pialorsi – "WCF Security Full Demo" – http://weblogs.asp.net/paolopia/archive/2007/12/16/wcf-security-full-demo.aspx

Videos

- MSDN TV – Windows Communication Foundation Bindings and Channels by Clemens Vasters – http://www.microsoft.com/downloads/details.aspx?FamilyID=16FF9371-82EA-4229-8868-EBC87F0F5E77&displaylang=en
- MSDN Webcast: Windows Communication Foundation Top to Bottom (Part 10 of 15): Security Fundamentals (Level 200) – http://msevents.microsoft.com/CUI/WebCastEventDetails.aspx?culture=en-US&EventID=1032344349&CountryCode=US

Web Casts

MSDN Support WebCasts

- MSDN Support WebCast: Building distributed services on the Windows Communication Foundation – http://support.microsoft.com/kb/907388/en-us