Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/secnetlpMSDN.asp

Roadmap

J.D. Meier, Alex Mackman, Michael Dunner, and Srinath Vasireddy
Microsoft Corporation

November 2002

Applies to:
Microsoft® .NET Framework version 1.0
ASP.NET
Enterprise Services
Web services
.NET Remoting
ADO.NET
Visual Studio® .NET
SQL™ Server
Windows® 2000

Summary: This guide presents a practical, scenario driven approach to designing and building secure ASP.NET applications for Windows 2000 and version 1.0 of the .NET Framework. It focuses on the key elements of authentication, authorization, and secure communication within and across the tiers of distributed .NET Web applications. (This roadmap: 6 printed pages; the entire guide: 608 printed pages)

Download

Download Building Secure ASP.NET Applications in .pdf format. (1.67 MB, 608 printed pages)

Contents

What This Guide Is About
- Part I, Security Models
- Part II, Application Scenarios
- Part III, Securing the Tiers
- Part IV, Reference

Who Should Read This Guide?
What You Must Know
Feedback and Support
Collaborators

Recommendations and sample code in the guide were built and tested using Visual Studio .NET Version 1.0 and validated on servers running Windows 2000 Advanced Server SP 3, .NET Framework SP 2, and SQL Server 2000 SP 2.

What This Guide Is About

This guide focuses on:

- Authentication (to identify the clients of your application)
- Authorization (to provide access controls for those clients)

- Secure communication (to ensure that messages remain private and are not altered by unauthorized parties)

Why authentication, authorization, and secure communication?

Security is a broad topic. Research has shown that early design of authentication and authorization eliminates a high percentage of application vulnerabilities. Secure communication is an integral part of securing your distributed application to protect sensitive data, including credentials, passed to and from your application, and between application tiers.

There are many technologies used to build .NET Web applications. To build effective application-level authentication and authorization strategies, you need to understand how to fine-tune the various security features within each product and technology area, and how to make them work together to provide an effective, defense-in-depth security strategy. This guide will help you do just that.

Figure 1 summarizes the various technologies discussed throughout the guide.

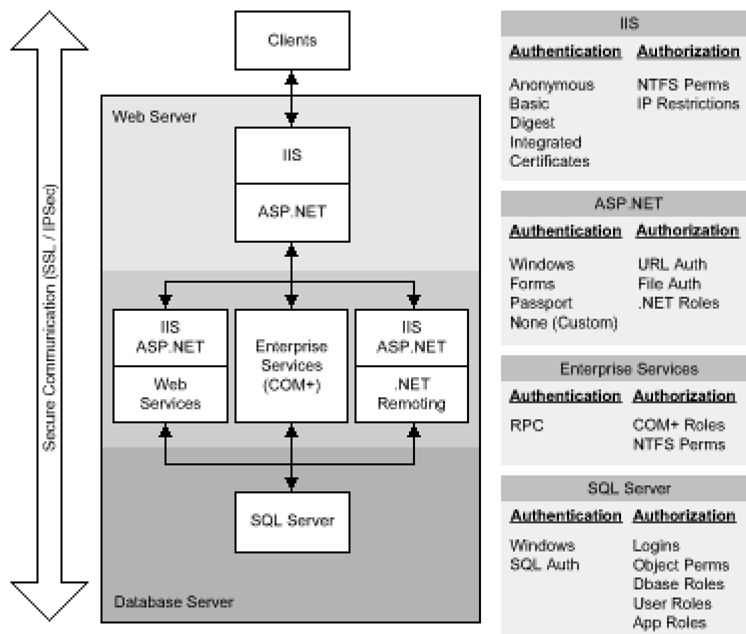

Figure 1. .NET Web application security

The guide is divided into four parts. The aim is to provide a logical partitioning, which will help you to more easily digest the content.

Part I, Security Models

Part I of the guide provides a foundation for the rest of the guide. Familiarity with the concepts, principles, and technologies introduced in Part I will allow you to extract maximum value from the remainder of the guide. Part I contains the following chapters.

- Chapter 1: Introduction

  This chapter highlights the goals of the guide, introduces key terminology, and presents a set of core principles that apply to the guidance presented in later chapters.

- Chapter 2: Security Model for ASP.NET Applications

  This chapter describes the common characteristics of .NET Web applications from a security perspective and introduces the .NET Web application security model. It also introduces the set of core implementation technologies that you will use to build secure .NET Web applications.

- Chapter 3: Authentication and Authorization

  Designing a coherent authentication and authorization strategy across your application's multiple tiers is a critical task. This chapter provides guidance to help you develop an appropriate strategy for your particular application scenario. It will help you choose the most appropriate authentication and authorization technique and apply them at the correct places in your application.

- Chapter 4: Secure Communication

  This chapter introduces the two core technologies that can be used to provide message confidentiality and message integrity for data that flows across the network between clients and servers on the Internet and corporate intranet. These are SSL and IPSec. This chapter also discusses RPC encryption, which can be used to secure the communication with remote serviced components.

Part II, Application Scenarios

Most applications can be categorized as intranet, extranet, or Internet applications. This part of the guide presents a set of common application scenarios, each of which falls into one of those categories. The key characteristics of each scenario are described and the potential security threats analyzed.

You are then shown how to configure and implement the most appropriate authentication, authorization, and secure communication strategy for each application scenario.

- Chapter 5: Intranet Security

  This chapter describes how to secure common intranet application scenarios.

- Chapter 6: Extranet Security

  This chapter describes how to secure common extranet application scenarios.

- Chapter 7: Internet Security

  This chapter describes how to secure common Internet application scenarios.

Part III, Securing the Tiers

This part of the guide contains detailed drill-down information that relates to the individual tiers and technologies associated with secure .NET Web applications. Part III contains the following chapters.

- Chapter 8: ASP.NET Security

  This chapter provides in-depth security recommendations for ASP.NET Web applications. It describes how to implement Forms and Windows authentication and how to perform authorization using the various gatekeepers supported by ASP.NET. Among many other topics, it also discusses how to store secrets, how to use the correct process identity, and how to access network resources such as remote databases by using Windows authentication.

- Chapter 9: Enterprise Services Security

  This chapter explains how to secure business functionality in serviced components contained within Enterprise Services applications. It shows you how and when to use Enterprise Services (COM+) roles for authorization, and how to configure RPC authentication and impersonation. It also shows you how to securely call serviced components from an ASP.NET Web application and how to identify and flow the original caller's security context through a middle tier serviced component.

- Chapter 10: Web Services Security

  This chapter focuses on platform-level security for Web services using the underlying features of Internet Information Services (IIS) and ASP.NET. For message-level security, Microsoft is developing the Web Services Development Kit, which allows you to build security solutions that conform to the WS-Security specification, part of the Global XML Architecture (GXA) initiative.

- Chapter 11: Remoting Security

  The .NET Framework provides a remoting infrastructure that allows clients to communicate with objects, hosted in remote application domains and processes or on remote computers. This chapter shows you how to implement secure .NET Remoting solutions.

- Chapter 12: Data Access Security

  This chapter presents recommendations and guidance that will help you develop a secure data access strategy. Topics covered include using Windows authentication from ASP.NET to the database, securing connection strings, storing credentials securely in a database, protecting against SQL injection attacks, and using database roles.

Part IV, Reference

This reference part of the guide contains supplementary information to help further your understanding of the techniques, strategies, and security solutions presented in earlier chapters.

- Chapter 13: Troubleshooting Security

  This chapter presents a set of troubleshooting tips, techniques, and tools to help diagnose security related issues.

- How Tos

  This section contains a series of step-by-step How-to articles that walk you through many of the solution techniques discussed in earlier chapters.

- Base Configuration

  This section lists the hardware and software used during the development and testing of the guide.

- Configuration Stores and Tools

  This section summarizes the configuration stores used by the various authentication, authorization, and secure communication services and lists the associated maintenance tools.

- Reference Hub

  This section provides a set of links to useful articles and Web sites that provide additional background information about the core topics discussed throughout the guide.

- How Does It Work?

  This section provides supplementary information that details how particular technologies work.

- ASP.NET Identity Matrix

  This section summarizes (with examples) the variables available to ASP.NET Web applications, Web services, and remote components hosted within ASP.NET that provide caller, thread, and process-level identity information.

- Cryptography and Certificates

  This section includes supplementary background information about cryptography and certificates.

- .NET Web Application Security

  This section provides a diagram that shows the authentication, authorization, and secure communication services available across the tiers of an ASP.NET application.

- Glossary

A glossary of security terminology used throughout the guide.

Who Should Read This Guide?

If you are a middleware developer or architect, who plans to build, or is currently building .NET Web applications using one or more of the following technologies, you should read this guide.

- ASP.NET
- Web services
- Enterprise Services
- Remoting
- ADO.NET

What You Must Know

To most effectively use this guide to design and build secure .NET Web applications, you should already have some familiarity and experience with .NET development techniques and technologies. You should be familiar with distributed application architecture and if you have already implemented .NET Web application solutions, you should know your own application architecture and deployment pattern.

Feedback and Support

Questions? Comments? Suggestions? For feedback on this security guide, please send e-mail to secguide@microsoft.com.

The security guide is designed to help you build secure .NET distributed applications. The sample code and guidance is provided as-is. Support is available through Microsoft Product Support for a fee.

Collaborators

Many thanks to the following contributors and reviewers:

Manish Prabhu, Jesus Ruiz-Scougall, Jonathan Hawkins and Doug Purdy, Keith Ballinger, Yann Christensen and Alexei Vopilov, Laura Barsan, Greg Fee, Greg Singleton, Sebastian Lange, Tarik Soulami, Erik Olson, Caesar Samsi, Riyaz Pishori, Shannon Pahl, Ron Jacobs, Dave McPherson, Christopher Brown, John Banes, Joel Scambray, Girish Chander, William Zentmayer, Shantanu Sarkar, Carl Nolan, Samuel Melendez, Jacquelyn Schmidt, Steve Busby, Len Cardinal, Monica DeZulueta, Paula Paul, Ed Draper, Sean Finnegan, David Alberto, Kenny Jones, Doug Orange, Alexey Yeltsov, Martin Kohlleppel, Joel Yoker, Jay Nanduri, Ilia Fortunov, Aaron Margosis (MCS), Venkat Chilakala, John Allen, Jeremy Bostron, Martin Petersen-Frey, Karl Westerholm, Jayaprakasam Siddian Thirunavukkarasu, Wade Mascia, Ryan Kivett, Sarath Mallavarapu, Jerry Bryant, Peter Kyte, Philip Teale, Ram Sunkara, Shaun Hayes, Eric Schmidt, Michael Howard, Rich Benack, Carlos Lyons, Ted Kehl, Peter Dampier, Mike Sherrill, Devendra Tiwari, Tavi Siochi, Per Vonge Nielsen, Andrew Mason, Edward Jezierski, Sandy Khaund, Edward Lafferty, Peter M. Clift, John Munyon, Chris Sfanos, Mohammad Al-Sabt, Anandha Murukan (Satyam), Keith Brown (DevelopMentor), Andy Eunson, John Langley (KANA Software), Kurt Dillard, Christof Sprenger, J.K.Meadows, David Alberto, Bernard Chen (Sapient)

At a Glance

J.D. Meier, Alex Mackman, Michael Dunner, and Srinath Vasireddy
Microsoft Corporation

November 2002

Applies to:
Microsoft® ASP.NET

Summary: This section allows you to quickly see the scope and coverage of the individual chapters in the guide. (12 printed pages)

Contents

Chapter 1: Introduction
Chapter 2: Security Model for ASP.NET Applications
Chapter 3: Authentication and Authorization
Chapter 4: Secure Communication
Chapter 5: Intranet Security
Chapter 6: Extranet Security
Chapter 7: Internet Security
Chapter 8: ASP.NET Security
Chapter 9: Enterprise Services Security
Chapter 10: Web Services Security
Chapter 11: .NET Remoting Security
Chapter 12: Data Access Security
Chapter 13: Troubleshooting Security Issues
Reference

Chapter 1: Introduction

This chapter highlights the goals of the guide, introduces key terminology and presents a set of core principles that apply to the guidance presented in later chapters.

Chapter 2: Security Model for ASP.NET Applications

This chapter describes the common characteristics of.NET Web applications from a security perspective and introduces the .NET Web application security model. It also introduces the set of core implementation technologies that you will use to build secure .NET Web applications.

The full range of gatekeepers that allow you to develop defense-in-depth security strategies are also introduced and the concept of principal-based authorization, using principal and identity objects is explained.

This chapter will help you answer the following questions:

- What are the typical deployment patterns adopted by .NET Web applications?
- What security features are provided by the various technologies that I use to build .NET Web applications?
- What gatekeepers should I be aware of and how do I use them to provide a defense-in-depth security strategy?
- What are principal and identity objects and why are they so significant?
- How does .NET security relate to Windows security?

Chapter 3: Authentication and Authorization

Designing a coherent authentication and authorization strategy across your application's multiple tiers is a critical task. This chapter provides guidance to help you develop an appropriate strategy for your particular application scenario. It will help you choose the most appropriate authentication and authorization technique and apply them at the correct places in your application.

Read this chapter to learn how to:

- Choose an appropriate authentication mechanism to identify users.
- Develop an effective authorization strategy.
- Choose an appropriate type of role-based security.
- Compare and contrast .NET roles with Enterprise Services (COM+) roles.
- Use database roles.
- Choose between the trusted subsystem resource access model and the impersonation/delegation model, which is used to flow the original caller's security context at the operating system level throughout an application's multiple tiers.

  These two core resource access models are shown below in Figure 1 and Figure 2.

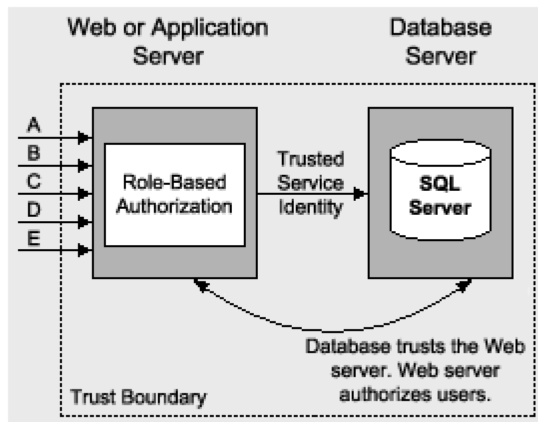

Figure 1. The Trusted Subsystem model

With the trusted subsystem model:

- Downstream resource access is performed using a fixed *trusted* identity and security context.
- The downstream resource manager (for example, database) *trusts* the upstream application to properly authenticate and authorize callers.
- The resource manager authorizes the application to access resources. Original callers are not authorized to directly access the resource manager.
- A trust boundary exists between the downstream and upstream components.
- Original caller identity (for auditing) flows at the application (not operating system) level.

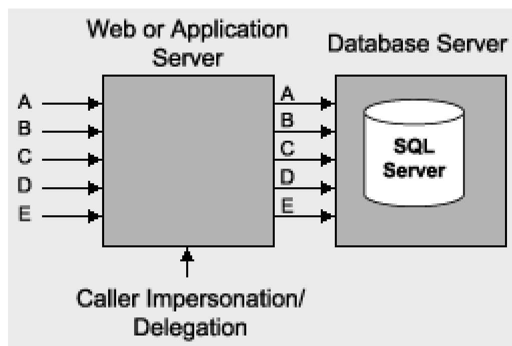

Figure 2. The impersonation/delegation model

With the impersonation/delegation model:

- Downstream resource access is performed using the original caller's security context.
- The downstream resource manager (for example, database) authorizes individual callers.
- The original caller identity flows at the operating system and is available for platform level auditing and per caller authorization.

Chapter 4: Secure Communication

This chapter introduces the two core technologies that can be used to provide message confidentiality and message integrity for data that flows across the network between clients and servers on the Internet and corporate intranet. These are SSL and IPSec. It also discusses RPC encryption that can be used to secure the communication with remote serviced components.

Read this chapter to learn how to:

- Apply secure communication techniques throughout the various tiers of your application.
- Choose between SSL and IPSec.
- Configure secure communication.
- Use RPC encryption.

The chapter addresses the need to provide secure communication channels between your application's various physical tiers as shown in Figure 3.

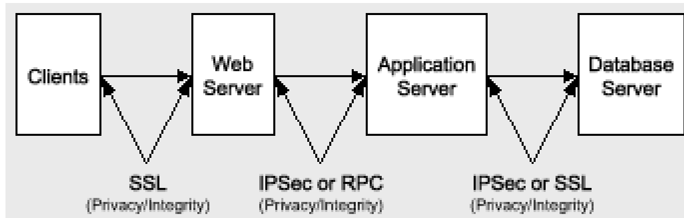

Figure 3. A typical Web deployment model, with secure communications

Chapter 5: Intranet Security

This chapter presents a set of common intranet application scenarios and for each one presents recommended security configurations. In each case, the configuration steps necessary to build the secure solution are presented, together with analysis and related scenario variations.

The application scenarios covered in this chapter are:

- ASP.NET to SQL Server

  This scenario is shown in Figure 4.

- ASP.NET to Enterprise Services to SQL Server
- ASP.NET to Web services to SQL Server
- ASP.NET to Remoting to SQL Server
- Flowing the original caller to the database

  This includes multi-tier Kerberos delegation scenarios, as shown in Figure 5.

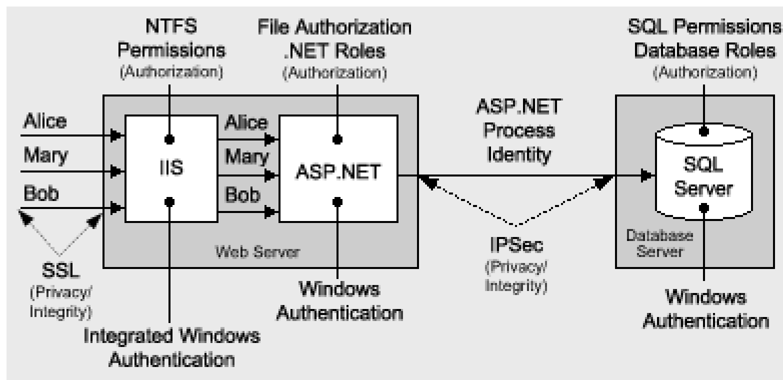

Figure 4. Security configuration for ASP.NET to remote SQL Server scenarios

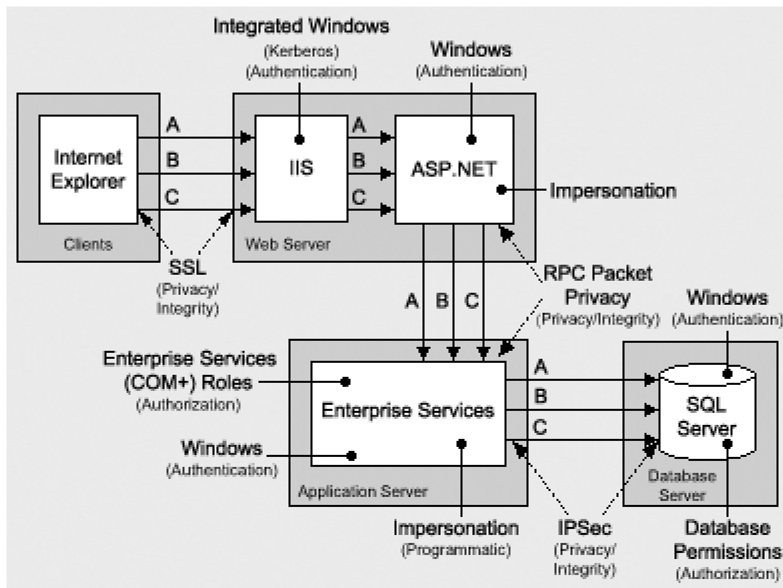

Figure 5. Security configuration for ASP.NET to remote Enterprise Services to remote SQL Server Kerberos delegation scenario

Read this chapter to lean how to:

- Use the local ASPNET account to make calls from an ASP.NET Web application to a remote SQL Server database.
- Establish trusted database connections to SQL Server using Windows authentication.
- Authorize database access with SQL Server user-defined database roles.
- Avoid storing credentials within your application.
- Secure sensitive data with a combination of SSL and IPSec.
- Implement Kerberos delegation to flow the original caller's security context across multiple application tiers to a back-end database.
- Flow the original caller's security context by using Basic authentication.
- Authorize users with a combination of ASP.NET file authorization, URL authorization, .NET roles and Enterprise Services (COM+) roles.
- Effectively use impersonation within an ASP.NET Web application.

Chapter 6: Extranet Security

This chapter presents a set of common extranet application scenarios and for each one presents recommended security configurations, configuration steps and analysis.

This chapter covers the following extranet scenarios.

- Exposing a Web Service (B2B partner exchange)

  This scenario is shown in Figure 6.

- Exposing a Web Application (partner application portal)

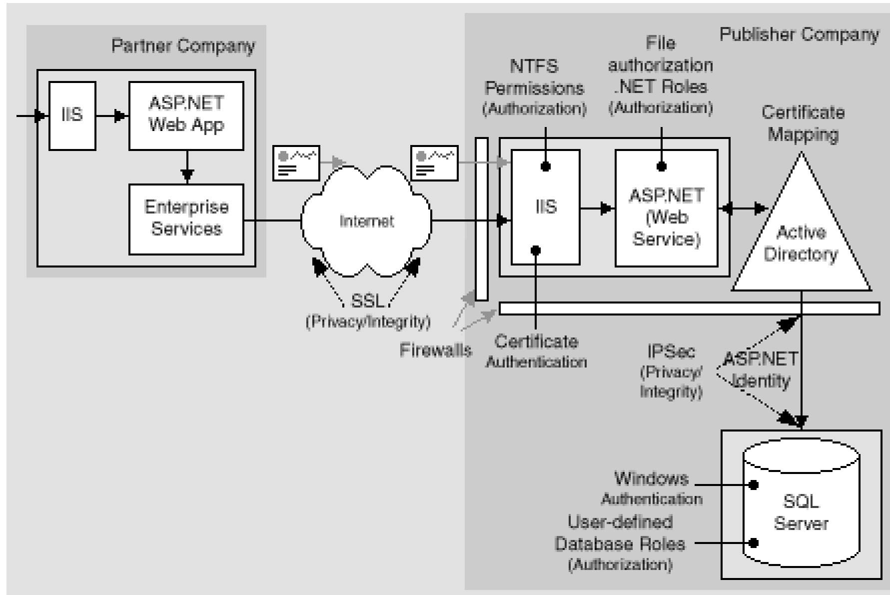

Figure 6. Security configuration for Web Service B2B partner exchange scenario

Read this chapter to lean how to:

- Authenticate partner companies by using client certificate authentication against a dedicated extranet Active Directory.
- Map certificates to Windows accounts.
- Authorize partner companies by using ASP.NET file authorization and .NET roles.
- Use the ASPNET identity to access a remote SQL Server database located on the corporate intranet.

Chapter 7: Internet Security

This chapter presents a set of common Internet application scenarios, and for each one presents recommended security configurations, configuration steps, and analysis.

This chapter covers the following Internet application scenarios:

- ASP.NET to SQL Server
- ASP.NET to Remote Enterprise Services to SQL Server

  This scenario is shown in Figure 7.

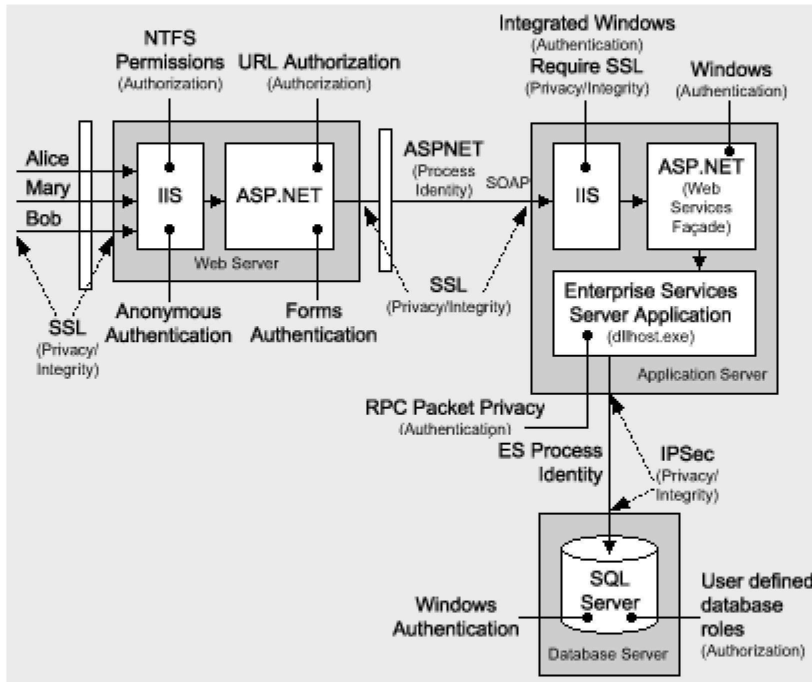

Figure 7. Security configuration for ASP.NET to remote Enterprise Services to SQL Server

Read this chapter to learn how to:

- Use Forms authentication with a SQL Server credential database.
- Avoid storing passwords in the credential database.
- Authorize Internet users with URL Authorization and .NET roles.
- Use Windows authentication from an ASP.NET Web application to SQL Server through a firewall.
- Secure sensitive data with a combination of SSL and IPSec.
- Communicate from an ASP.NET Web application to a remote Enterprise Services application through a firewall by using SOAP.
- Secure calls to serviced component in the application's middle tier.

Chapter 8: ASP.NET Security

This chapter provides in-depth security recommendations for ASP.NET Web applications. This chapter covers the range of authentication, authorization and secure communication services provided by IIS and ASP.NET. These are illustrated in Figure 8.

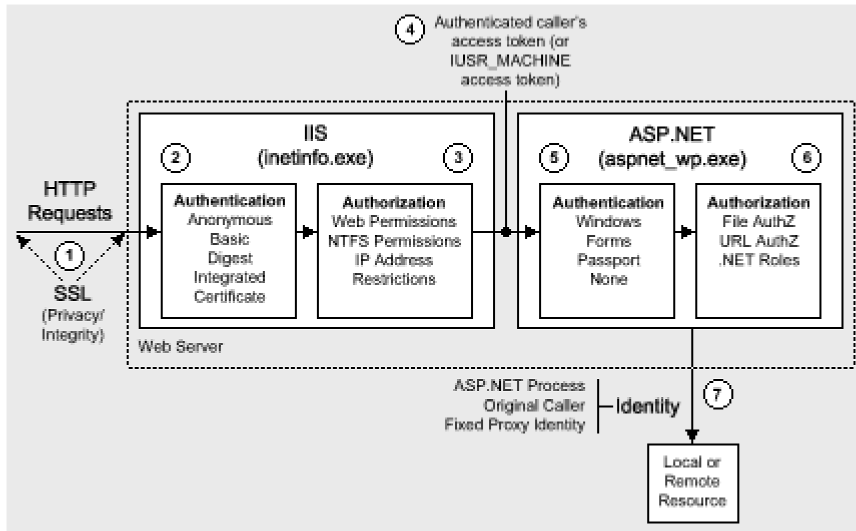

Figure 8. ASP.NET security services

Read this chapter to learn how to:

- Configure the various ASP.NET authentication modes.
- Implement Forms authentication.
- Implement Windows authentication.
- Work with **IPrincipal** and **IIdentity** objects.
- Effectively use the IIS and ASP.NET gatekeepers.
- Configure and use ASP.NET File authorization.
- Configure and use ASP.NET URL authorization.
- Implement declarative, imperative and programmatic role-based security, using principal permission demands and **IPrincipal.IsInRole**.
- Know when and when not to use impersonation within an ASP.NET Web application.
- Choose an appropriate account to run ASP.NET.
- Access local and network resources using the ASP.NET process identity.
- Access remote SQL Server databases using the local ASPNET account.
- Call COM objects from ASP.NET.
- Effectively use the anonymous Internet user account in Web hosting environments.
- Store secrets in an ASP.NET Web application.
- Secure session and view state.
- Configure ASP.NET security in Web Farm scenarios.

Chapter 9: Enterprise Services Security

This chapter explains how to secure business functionality in serviced components contained within Enterprise Services applications. It shows you how and when to use Enterprise Services (COM+) roles for authorization, and how to configure RPC authentication and impersonation. It also shows you how to securely call serviced components from an ASP.NET Web application and how to identify and flow the original caller's security context through a middle tier serviced component.

Figure 9 shows the Enterprise Services security features covered by this chapter.

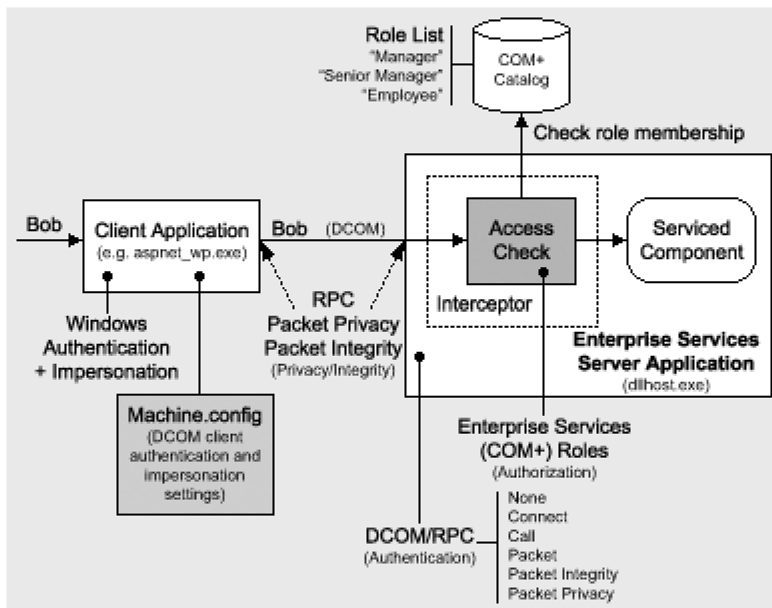

Figure 9. Enterprise Services security overview

Read this chapter to learn how to:

- Configure an Enterprise Services application using .NET attributes.
- Secure server and library applications.
- Choose an appropriate account to run an Enterprise Services server application.
- Implement method level Enterprise Services (COM+) role based security both programmatically and declaratively.
- Configure ASP.NET as a DCOM client.
- Securely call serviced components from ASP.NET.
- Compare Enterprise Services (COM+) roles with .NET roles.
- Identify callers within a serviced component.
- Flow the original caller's security context through an Enterprise Services application by using programmatic impersonation within a serviced component.
- Access local and network resources from a serviced component.
- Use RPC encryption to secure sensitive data passed to and from serviced components.
- Understand the process of RPC authentication level negotiation.
- Use DCOM through firewalls.

Chapter 10: Web Services Security

This chapter focuses on platform level security for Web services using the underlying features of IIS and ASP.NET. For message level security, Microsoft is developing the Web Services Development Kit, which allows you to build security solutions that conform to the WS-Security specification, part of the Global XML Architecture (GXA) initiative.

The ASP.NET Web services platform security architecture is shown in Figure 10.

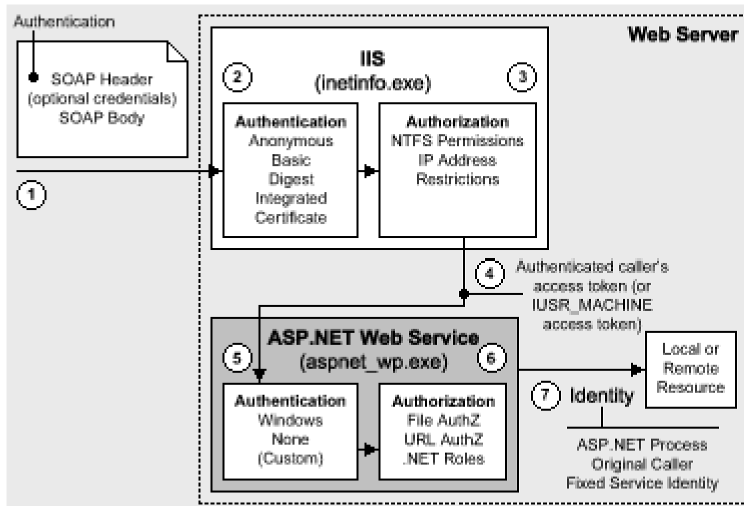

Figure 10. Web services security architecture

Read this chapter to learn how to:

- Implement platform-based Web service security solutions.
- Develop an authentication and authorization strategy for a Web service.
- Use client certificate authentication with Web services.
- Use ASP.NET file authorization, URL authorization, and .NET roles to provide authorization in Web services.
- Flow the original caller's security context through a Web service.
- Call Web services using SSL.
- Access local and network resources from Web services.
- Pass credentials for authentication to a Web service through a Web service proxy.
- Implement the trusted subsystem model for Web services.
- Call COM objects from Web services.

Chapter 11: .NET Remoting Security

The .NET Framework provides a remoting infrastructure that allows clients to communicate with objects, hosted in remote application domains and processes, or on remote computers. This chapter shows you how to implement secure .NET Remoting solutions.

Read this chapter to learn how to:

- Choose an appropriate host for remote components.
- Use all of the available gatekeepers to provide defense-in-depth security.
- Use URL authentication and .NET roles to authorize access to remote components.
- Use File authentication with remoting. This requires you to create a physical .rem or .soap file that corresponds to the remote component's object URI.
- Access local and network resources from a remote component.
- Pass credentials for authentication to a remote component through the remote component proxy object.

- Flow the original caller's security context through a remote component.
- Secure communication to and from remote components using a combination of SSL and IPSec.
- Know when to use remoting and when to use Web services.

Chapter 12: Data Access Security

This chapter presents recommendations and guidance that will help you develop a secure data access strategy. The key issues covered by this chapter are shown in Figure 11. These include storing connection strings securely, using an appropriate identity for database access, securing data passed to and from the database, using an appropriate authentication mechanism and implementing authorization in the database.

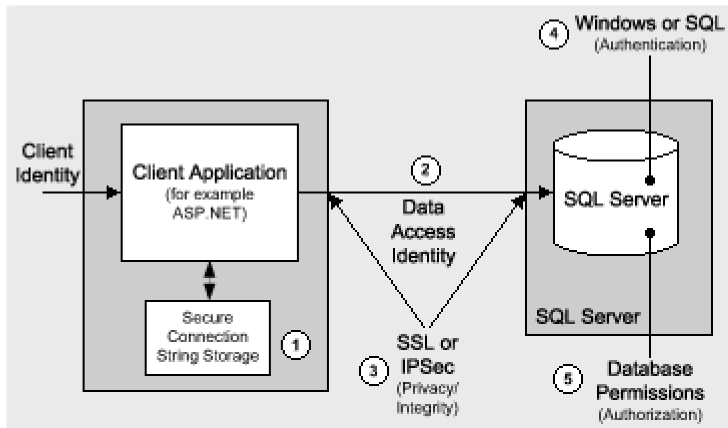

Figure 11. Data Access security overview

Read this chapter to learn how to:

- Use Windows authentication from ASP.NET to your database.
- Secure connection strings.
- Use DPAPI from ASP.NET Web applications to store secrets such as connection strings and credentials.
- Store credentials for authentication securely in a database.
- Validate user input to protect against SQL injection attacks.
- Mitigate the security threats associated with the use of SQL authentication.
- Know which type of database roles to use.
- Compare and contrast database user roles with SQL Server application roles.
- Secure communication to SQL Server using IPSec and also SSL.
- Create a least privilege database account.
- Enable auditing in SQL Server.

Chapter 13: Troubleshooting Security Issues

This chapter provides troubleshooting tips, techniques and tools to help diagnose security related issues. Read this chapter to learn a proven process for effectively troubleshooting security issues you may encounter while building your ASP.NET applications. For example, you'll learn techniques for determining identity in your ASP.NET pages, which can be used to diagnose authentication and access control issues. You'll also learn how to troubleshoot Kerberos authentication. The chapter concludes with a concise list of some of the more useful troubleshooting tools, used by Microsoft support to troubleshoot customer issues.

Reference

Use the supplementary information in this section of the guide to help further your understanding of the techniques, strategies and security solutions presented in earlier chapters. Detailed How To articles provide step-by-step procedures that enable you to implement specific security solutions. It contains the following information:

- Reference Hub
- How To articles
- How Does it Work?
- ASP.NET Identity Matrix
- Base Configuration
- Configuring Security
- Cryptography and Certificates
- .NET Web Application Security Figure
- Glossary

Introduction

J.D. Meier, Alex Mackman, Michael Dunner and Srinath Vasireddy
Microsoft Corporation

November 2002

Applies to:
Microsoft® ASP.NET

See the Landing Page for the starting point and a complete overview of *Building Secure ASP.NET Applications*.

Summary: This chapter defines the scope and organization of the guide and highlights its goals. It also introduces key terminology and presents a set of core principles that apply to the guidance presented in later chapters. (7 printed pages)

Contents

The Connected Landscape
Scope
What Are the Goals of this Guide?
How You Should Read This Guide
Organization of the Guide
Key Terminology
Principles
Summary

Building secure distributed Web applications is challenging. Your application is only as secure as its weakest link. With distributed applications, you have a lot of moving parts and making those parts work together in a secure fashion requires a working knowledge that spans products and technologies.

You already have a lot to consider; integrating various technologies, staying current with technology and keeping a step ahead of the competition. If you don't already know how to build secure applications, can you afford the time and effort to learn? More to the point, can you afford not to?

The Connected Landscape

If you already know how to build secure applications, are you able to apply what you know when you build .NET Web applications? Are you able to apply your knowledge in today's landscape of Web-based distributed applications, where Web services connect businesses to other business and business to customers and where applications offer various degrees of exposure; for example, to users on intranets, extranets and the Internet?

Consider some of the fundamental characteristics of this connected landscape:

- Web services use standards such as SOAP, Extensible Markup Language (XML) and Hypertext Transport Protocol (HTTP), but fundamentally they pass potentially sensitive information using plain text.
- Internet business-to-consumer applications pass sensitive data over the Web.
- Extranet business-to-business applications blur the lines of trust and allow applications to be called by other applications in partner companies.
- Intranet applications are not without their risks considering the sensitive nature of payroll and Human Resource (HR) applications. Such applications are particularly vulnerable to rogue administrators and disgruntled employees.

Scope

This guide focuses on:

- Authentication (to identify the clients of your application)

- Authorization (to provide access controls for those clients)
- Secure communication (to ensure that messages remain private and are not altered by unauthorized parties)

Why authentication, authorization and secure communication?

Security is a broad topic. Research has shown that early design of authentication and authorization eliminates a high percentage of application vulnerabilities. Secure communication is an integral part of securing your distributed application to protect sensitive data, including credentials, passed to and from your application and between application tiers.

What Are the Goals of This Guide?

This guide is not an introduction to security. It is not a security reference for the Microsoft .NET Framework —for that you have the .NET Framework Software Development Kit (SDK) available from MSDN; see the "References" section of this guide for details. This guide picks up where the documentation leaves off and presents a scenario-based approach to sharing recommendations and proven techniques, as gleaned from the field, customer experience and insight from the product teams at Microsoft.

The information in this guide is designed to show you how to:

- Raise the security bar for your application.
- Identify where and how you need to perform authentication.
- Identify where and how you need to perform authorization.
- Identify where and how you need to secure communication both to your application (from your end users) and between application tiers.
- Identify common pitfalls and how to avoid them.
- Identify top risks and their mitigation related to authentication and authorization.
- Avoid opening up security just to make things work.
- Identify not only *how*, but also *when* to use various security features.
- Eliminate FUD (fear, uncertainty and doubt).
- Promote best practices and predictable results.

How You Should Read This Guide

The guide has been developed to be modular. This allows you to pick and choose which chapters to read. For example, if you are interested in learning about the in-depth security features provided by a specific technology, you can jump straight to Part III of the guide (Chapters 8 through 12), which contains in-depth material covering ASP.NET, Enterprise Services, Web Services, .NET Remoting, and data access.

However, you are encouraged to read the early chapters (Chapters 1 through 4) in Part I of the guide first, because these will help you understand the security model and identify the core technologies and security services at your disposal. Application architects should make sure they read Chapter 3, which provides some key insights into designing an authentication and authorization strategy that spans the tiers of your Web application. Part I will provide you with the foundation materials that will allow you to extract maximum benefit from the remainder of the guide.

The intranet, extranet and Internet chapters (Chapters 5 through 7) in Part II of the guide will show you how to secure specific application scenarios. If you know the architecture and deployment pattern that is or will be adopted by your application, use this part of the guide to understand the security issues involved and the basic configuration steps required to secure specific scenarios.

Finally, additional information and reference material in Part IV of the guide will help further your understanding of specific technology areas. It also contains a library of How Tos that enable you to develop working security solutions in the shortest possible time.

Organization of the Guide

The guide is divided into four parts. The aim is to provide a logical partitioning, which will help you to more easily digest the content.

Part I, Security Models

Part 1 of the guide provides a foundation for the rest of the guide. Familiarity with the concepts, principles and technologies introduced in Part 1 will enable you to extract maximum value from the remainder of the guide. Part 1 contains the following chapters.

- Chapter 1, "Introduction"
- Chapter 2, "Security Model for ASP.NET Applications "
- Chapter 3, "Authentication and Authorization"
- Chapter 4, "Secure Communication"

Part II, Application Scenarios

Most applications can be categorized as intranet, extranet or Internet applications. This part of the guide presents a set of common application scenarios, each of which falls into one of the aforementioned categories. The key characteristics of each scenario are described and the potential security threats analyzed.

You are then shown how to configure and implement the most appropriate authentication, authorization and secure communication strategy for each application scenario. Each scenario also contains sections that include a detailed analysis, common pitfalls to watch out for and frequently asked questions (FAQ). Part II contains the following chapters:

- Chapter 5, "Intranet Security"
- Chapter 6, "Extranet Security"
- Chapter 7, "Internet Security"

Part III, Securing the Tiers

This part of the guide contains detailed information that relates to the individual tiers and technologies associated with secure .NET-connected Web applications. Part III contains the following chapters:

- Chapter 8, "ASP.NET Security"
- Chapter 9, "Enterprise Services Security"
- Chapter 10, "Web Services Security"
- Chapter 11, ".NET Remoting Security"
- Chapter 12, "Data Access Security"

Within each chapter, a brief overview of the security architecture as it applies to the particular technology in question is presented. Authentication and authorization strategies are discussed for each technology along with configurable security options, programmatic security options and actionable recommendations of when to use the particular strategy.

Each chapter offers guidance and insight that will allow you to choose and implement the most appropriate authentication, authorization and secure communication option for each technology. In addition, each chapter presents additional information specific to the particular technology. Finally, each chapter concludes with a concise recommendation summary.

Part IV, Reference

This reference part of the guide contains supplementary information to help further your understanding of the techniques, strategies, and security solutions presented in earlier chapters. Detailed How Tos provide step-by-step procedures that enable you to implement specific security solutions. It contains the following information:

- Chapter 13, "Troubleshooting Security"
- How Tos
- "Base Configuration"
- "Configuration Stores and Tools"
- "How Does It Work?"
- "ASP.NET Identity Matrix"
- "Cryptography and Certificates"
- "ASP.NET Security Model"
- "Reference Hub"
- "Glossary"

Key Terminology

This section introduces some key security terminology used throughout the guide. Although a full glossary of terminology is provided within the "Reference" section of this guide, make sure you are very familiar with the following terms:

- **Authentication**. Positively identifying the clients of your application; clients might include end-users, services, processes or computers.
- **Authorization**. Defining what authenticated clients are allowed to see and do within the application.
- **Secure Communications**. Ensuring that messages remain private and unaltered as they cross networks.
- **Impersonation**. This is the technique used by a server application to access resources on behalf of a client. The client's security context is used for access checks performed by the server.
- **Delegation**. An extended form of impersonation that allows a server process that is performing work on behalf of a client, to access resources on a remote computer. This capability is natively provided by Kerberos on Microsoft® Windows® 2000 and later operating systems. Conventional impersonation (for example, that provided by NTLM) allows only a single network hop. When NTLM impersonation is used, the one hop is used between the client and server computers, restricting the server to local resource access while impersonating.
- **Security Context**. Security context is a generic term used to refer to the collection of security settings that affect the security-related behavior of a process or thread. The attributes from a process' logon session and access token combine to form the security context of the process.
- **Identity**. Identity refers to a characteristic of a user or service that can uniquely identify it. For example, this is often a display name, which often takes the form authority/user name.

Principles

There are a number of overarching principles that apply to the guidance presented in later chapters. The following summarizes these principles:

- **Adopt the principle of least privilege**. Processes that run script or execute code should run under a least privileged account to limit the potential damage that can be done if the process is compromised. If a malicious user manages to inject code into a server process, the privileges granted to that process determine to a large degree the types of operations the user is able to perform. Code that requires additional trust (and raised privileges) should be isolated within separate processes.

  The ASP.NET team made a conscious decision to run the ASP.NET account with least privileges (using the ASPNET account). During the beta release of the .NET Framework, ASP.NET ran as SYSTEM, an inherently less secure setting.

- **Use defense in depth**. Place check points within each of the layers and subsystems within your application. The check points are the gatekeepers that ensure that only authenticated and authorized users are able to access the next downstream layer.
- **Don't trust user input**. Applications should thoroughly validate all user input before performing operations with that input. The validation may include filtering out special characters. This preventive measure protects the application against accidental misuse or deliberate attacks by people who are attempting to inject malicious commands into the system. Common examples include SQL injection attacks, script injection and buffer overflow.
- **Use secure defaults**. A common practice among developers is to use reduced security settings, simply to make an application work. If your application demands features that force you to reduce or change default security settings, test the effects and understand the implications before making the change.
- **Don't rely on security by obscurity**. Trying to hide secrets by using misleading variable names or storing them in odd file locations does not provide security. In a game of hide-and-seek, it's better to use platform features or proven techniques for securing your data.
- **Check at the gate**. You don't always need to flow a user's security context to the back end for authorization checks. Often, in a distributed system, this is not the best choice. Checking the client at the gate refers to authorizing the user at the first point of authentication (for example, within the Web application on the Web server), and determining which resources and operations (potentially provided by downstream services) the user should be allowed to access.

  If you design solid authentication and authorization strategies at the gate, you can circumvent the need to delegate the original caller's security context all the way through to your application's data tier.

- **Assume external systems are insecure**. If you don't own it, don't assume security is taken care of for you.
- **Reduce surface area**. Avoid exposing information that is not required. By doing so, you are potentially opening doors that can lead to additional vulnerabilities. Also, handle errors gracefully; don't expose any more information than is required when returning an error message to the end user.
- **Fail to a secure mode**. If your application fails, make sure it does not leave sensitive data unprotected. Also, do not provide too much detail in error messages; meaning don't include details that could help an attacker exploit a vulnerability in your application. Write detailed error information to the Windows event log.
- **Remember you are only as secure as your weakest link**. Security is a concern across all of your application tiers.
- **If you don't use it, disable it**. You can remove potential points of attack by disabling modules and components that your application does not require. For example, if your application doesn't use output caching, then you should disable the ASP.NET output cache module. If a future security vulnerability is found in the module, your application is not threatened.

Summary

This chapter has provided some foundation material to prepare you for the rest of the guide. It has described the goals of the guide and presented its overall structure. Make sure you are familiar with the key terminology and principles introduced in this chapter, because these are used and referenced extensively throughout the forthcoming chapters.

Security Model for ASP.NET Applications

J.D. Meier, Alex Mackman, Michael Dunner, and Srinath Vasireddy
Microsoft Corporation

November 2002

Applies to:
Microsoft® ASP.NET

See the Landing Page for the starting point and a complete overview of *Building Secure ASP.NET Applications*.

Summary: This chapter describes the common characteristics of .NET Web applications from a security perspective and introduces the .NET Web application security model. It also introduces the set of core implementation technologies that you will use to build secure .NET Web applications. (16 printed pages)

Contents

.NET Web Applications
Implementation Technologies
Security Architecture
Identities and Principals
Summary

This chapter introduces .NET Web application security. It provides an overview of the security features and services that span the tiers of a typical .NET Web application.

The goal of the chapter is to:

- Provide a frame of reference for typical .NET Web applications
- Identify the authentication, authorization and secure communication security features provided by the various implementation technologies used to build .NET Web applications
- Identify gatekeepers and gates that can be used in your application to enforce trust boundaries

.NET Web Applications

This section provides a brief introduction to .NET Web applications and describes their characteristics both from a logical and physical viewpoint. It also provides an introduction to the various implementation technologies used to build .NET Web applications.

Logical Tiers

Logical application architecture views any system as a set of cooperating services grouped in the following layers:

- User Services
- Business Services
- Data Services

The value of this logical architecture view is to identify the generic types of services invariably present in any system, to ensure proper segmentation and to drive the definition of interfaces between tiers. This segmentation allows you to make more discreet architecture and design choices when implementing each layer, and to build a more maintainable application.

The layers can be described as follows:

- **User Services** are responsible for the client interaction with the system and provide a common bridge into the core business logic encapsulated by components within the Business Services layer. Traditionally, User Services are associated most often with interactive users. However, they also perform the initial processing of programmatic requests from other systems, where no visible user interface is involved. Authentication and

authorization, the precise nature of which varies depending upon the client type, are typically performed within the User Services layer.

- **Business Services** provide the core functionality of the system and encapsulate business logic. They are independent from the delivery channel and back-end systems or data sources. This provides the stability and flexibility necessary to evolve the system to support new and different channels and back-end systems. Typically, to service a particular business request involves a number of cooperating components within the Business Services layer.
- **Data Services** provide access to data (hosted within the boundaries of the system), and to other (back-end) systems through generic interfaces, which are convenient to use from components within the Business Services layer. Data Services abstract the multitude of back-end systems and data sources, and encapsulate specific access rules and data formats.

The logical classification of service types within a system may correlate with, but is relatively independent from, the possible physical distribution of the components implementing the services.

It is also important to remember that the logical tiers can be identified at any level of aggregation; that is, the tiers can be identified for the system as a whole (in the context of its environment and external interactions) and for any contained subsystem. For example, each remote node that hosts a Web service consists of User Services (handling incoming requests and messages), Business Services and Data Services.

Physical Deployment Models

The three logical service layers described earlier, in no way imply specific numbers of physical tiers. All three logical services may be physically located on the same computer, or they may be spread across multiple computers.

The Web server as an application server

A common deployment pattern for .NET Web applications is to locate business and data access components on the Web server. This minimizes the network hops, which can help performance. This model is shown in Figure 2.1.

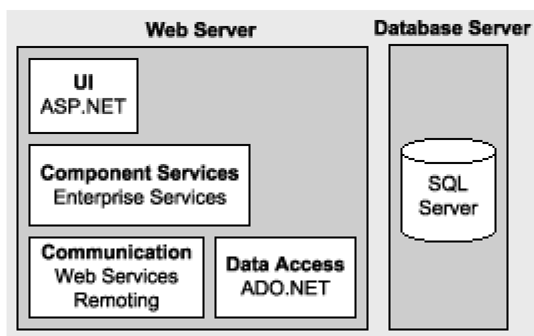

Figure 2.1. The Web server as an application server

Remote application tier

The remote application tier is a common deployment pattern, particularly for Internet scenarios where the Web tier is self-contained within a perimeter network (also known as DMZ, demilitarized zone, and screened subnet) and is separated from end users and the remote application tier with packet filtering firewalls. The remote application tier is shown in Figure 2.2.

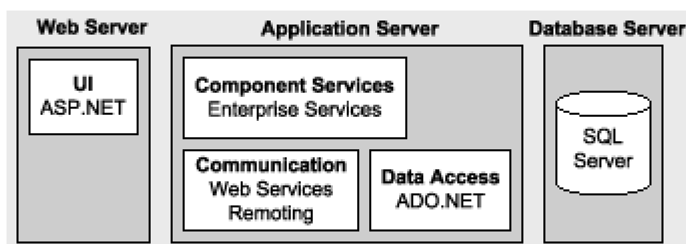

Figure 2.2. The introduction of a remote application tier

Implementation Technologies

.NET Web applications typically implement one or more of the logical services by using the following technologies:

- ASP.NET
- Enterprise Services
- Web services
- .NET Remoting
- ADO.NET and Microsoft® SQL Server™ 2000
- Internet Protocol Security (IPSec)
- Secure Sockets Layer (SSL)

ASP.NET

ASP.NET is typically used to implement User Services. ASP.NET provides a pluggable architecture that can be used to build Web pages. For more information about ASP.NET, see the following resources:

- Chapter 8, ASP.NET Security
- ASP.NET in the "Reference Hub" section of this guide

Enterprise Services

Enterprise Services provide infrastructure-level services to applications. These include distributed transactions and resource management services such as object pooling for .NET components. For more information about Enterprise Services, see the following resources:

- Chapter 9, Enterprise Services Security
- Understanding Enterprise Services (COM+) in .NET on MSDN®
- Enterprise Services in the "Reference Hub" section of this guide

Web Services

Web services enable the exchange of data and the remote invocation of application logic using SOAP-based message exchanges to move data through firewalls and between heterogeneous systems. For more information about Web services, see the following resources:

- Chapter 10, Web Services Security
- XML Web Services Development Center on MSDN
- Web Services in the "Reference Hub" section of this guide

.NET Remoting

.NET Remoting provides a framework for accessing distributed objects across process and machine boundaries. For more information about .NET Remoting, see the following resources:

- Chapter 11, .NET Remoting Security
- Remoting in the "Reference Hub" section of this guide

ADO.NET and SQL Server 2000

ADO.NET provides data access services. It is designed from the ground up for distributed Web applications, and it has rich support for the disconnected scenarios inherently associated with Web applications. For more information about ADO.NET, see the following resources:

- Chapter 12, Data Access Security
- ADO.NET in the "Reference Hub" section of this guide

SQL Server provides integrated security that uses the operating system authentication mechanisms (Kerberos or NTLM). Authorization is provided by logons and granular permissions that can be applied to individual database objects. For more information about SQL Server 2000, see the following resources:

- Chapter 12, Data Access Security

Internet Protocol Security (IPSec)

IPSec provides point-to-point, transport level encryption and authentication services. For more information about IPSec, see the following resources:

- Chapter 4, Secure Communication.
- *IPSec—The New Security Standard for the Internet, Intranets and Virtual Private Networks* by Naganand Doraswamy and Dan Harkins (Prentice Hall PTR, ISBN; ISBN: 0-13-011898-2); Chapter 4 is available on TechNet.

Secure Sockets Layer (SSL)

SSL provides a point-to-point secure communication channel. Data sent over the channel is encrypted. For more information about SSL, see the following resources:

- Chapter 4, Secure Communication
- *Microsoft® Windows® 2000 and IIS 5.0 Administrator's Pocket Consultant* (Microsoft Press, ISBN: 0-7356-1024-X); Chapter 6 is available on TechNet

Security Architecture

Figure 2.3 shows the remote application tier model together with the set of security services provided by the various technologies introduced earlier. Authentication and authorization occurs at many individual points throughout the tiers. These services are provided primarily by Internet Information Services (IIS), ASP.NET, Enterprise Services and SQL Server. Secure communication channels are also applied throughout the tiers and stretch from the client browser or device, right through to the database. Channels are secured with a combination of Secure Sockets Layer (SSL) or IPSec.

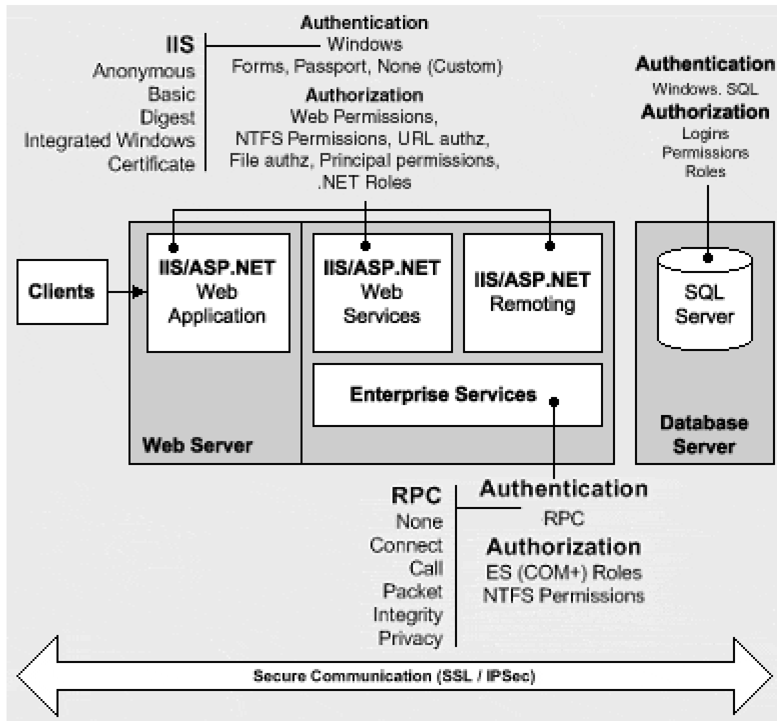

Figure 2.3. Security architecture

Security Across the Tiers

The authentication, authorization, and secure communication features provided by the technologies discussed earlier are summarized in Table 2.1.

Table 2.1. Security features

Technology	Authentication	Authorization	Secure Communication
IIS	Anonymous Basic Digest Windows Integrated (Kerberos/NTLM) Certificate	IP/DNS Address Restrictions Web Permissions NTFS Permissions; Windows Access Control Lists (ACLs) on requested files	SSL
ASP.NET	None (Custom) Windows Forms Passport	File Authorization URL Authorization Principal Permissions .NET Roles	
Web services	Windows None (Custom) Message level authentication	File Authorization URL Authorization Principal Permissions .NET Roles	SSL and Message level encryption
Remoting	Windows	File Authorization URL Authorization Principal Permissions .NET Roles	SSL and message level encryption
Enterprise Services	Windows	Enterprise Services (COM+) Roles NTFS Permissions	Remote Procedure Call (RPC) Encryption

SQL Server 2000	Windows (Kerberos/NTLM) SQL authentication	Server logins Database logins Fixed database roles User defined roles Application roles Object permissions	SSL
Windows 2000	Kerberos NTLM	Windows ACLs	IPSec

Authentication

The .NET Framework on Windows 2000 provides the following authentication options:

- ASP.NET Authentication Modes
- Enterprise Services Authentication
- SQL Server Authentication

ASP.NET authentication modes

ASP.NET authentication modes include Windows, Forms, Passport and None.

- **Windows authentication**. With this authentication mode, ASP.NET relies on IIS to authenticate users and create a Windows access token to represent the authenticated identity. IIS provides the following authentication mechanisms:
  - **Basic authentication**. Basic authentication requires the user to supply credentials in the form of a user name and password to prove their identity. It is a proposed Internet standard based on RFC 2617. Both Netscape Navigator and Microsoft Internet Explorer support Basic authentication. The user's credentials are transmitted from the browser to the Web server in an unencrypted Base64 encoded format. Because the Web server obtains the user's credentials unencrypted, the Web server can issue remote calls (for example, to access remote computers and resources) using the user's credentials.

    Note Basic authentication should only be used in conjunction with a secure channel (typically established by using SSL). Otherwise, user names and passwords can be easily stolen with network monitoring software. If you use Basic authentication you should use SSL on all pages (not just a logon page), because credentials are passed on all subsequent requests. For more information about using Basic authentication with SSL, see Chapter 8, "ASP.NET Security."

  - **Digest authentication**. Digest authentication, introduced with IIS 5.0, is similar to Basic authentication except that instead of transmitting the user's credentials unencrypted from the browser to the Web server, it transmits a hash of the credentials. As a result it is more secure, although it requires an Internet Explorer 5.0 or later client and specific server configuration.
  - **Integrated Windows authentication**. Integrated Windows Authentication (Kerberos or NTLM depending upon the client and server configuration) uses a cryptographic exchange with the user's Internet Explorer Web browser to confirm the identity of the user. It is supported only by Internet Explorer (and not by Netscape Navigator), and as a result tends to be used only in intranet scenarios, where the client software can be controlled. It is used only by the Web server if either anonymous access is disabled or if anonymous access is denied through Windows file system permissions.
  - **Certificate authentication**. Certificate authentication uses client certificates to positively identify users. The client certificate is passed by the user's browser (or client application) to the Web server. (In the case of Web services, the Web services client passes the certificate by means of the ClientCertificates property of the HttpWebRequest object). The Web server then extracts the user's identity from the certificate. This approach relies on a client certificate being installed on the user's computer and as a result tends to be used mostly in intranet or extranet scenarios where the user population is well known and controlled. IIS, upon receipt of a client certificate, can map the certificate to a Windows account.
  - **Anonymous authentication**. If you do not need to authenticate your clients (or you implement a custom authentication scheme), IIS can be configured for Anonymous authentication. In this event, the

Web server creates a Windows access token to represent all anonymous users with the same anonymous (or guest) account. The default anonymous account is IUSR_MACHINENAME, where MACHINENAME is the NetBIOS name of your computer specified at install time.

- **Passport authentication**. With this authentication mode, ASP.NET uses the centralized authentication services of Microsoft Passport. ASP.NET provides a convenient wrapper around functionality exposed by the Microsoft Passport Software Development Kit (SDK), which must be installed on the Web server.
- **Forms authentication**. This approach uses client-side redirection to forward unauthenticated users to a specified HTML form that allows them to enter their credentials (typically user name and password). These credentials are then validated and an authentication ticket is generated and returned to the client. The authentication ticket maintains the user identity and optionally a list of roles that the user is a member of for the duration of the user's session.

  Forms authentication is sometimes used solely for Web site personalization. In this case, you need write little custom code because ASP.NET handles much of the process automatically with simple configuration. For personalization scenarios, the cookie needs to hold only the user name.

  > **Note** Forms authentication sends the user name and password to the Web server in plain text. As a result, you should use Forms authentication in conjunction with a channel secured by SSL. For continued protection of the authentication cookie transmitted on subsequent requests, you should consider using SSL for all pages within your application and not just the logon page.

- **None**. None indicates that you either don't want to authenticate users or that you are using a custom authentication protocol.

More information

For more details about ASP.NET authentication, see Chapter 8, ASP.NET Security.

Enterprise Services authentication

Enterprise Services authentication is performed by using the underlying Remote Procedure Call (RPC) transport infrastructure, which in turn uses the operating system Security Service Provider Interface (SSPI). Clients of Enterprise Services applications may be authenticated using Kerberos or NTLM authentication.

A serviced component can be hosted in a Library application or Server application. Library applications are hosted within client processes and as a result assume the client's identity. Server applications run in separate server processes under their own identity. For more information about identity, see the "Identities and Principals" section later in this chapter.

The incoming calls to a serviced component can be authenticated at the following levels:

- **Default**: The default authentication level for the security package is used.
- **None**: No authentication occurs.
- **Connect**: Authentication occurs only when the connection is made.
- **Call**: Authenticates at the start of each remote procedure call.
- **Packet**: Authenticates and verifies that all call data is received.
- **Packet Integrity**: Authenticates and verifies that none of the data has been modified in transit.
- **Packet Privacy**: Authenticates and encrypts the packet, including the data and the sender's identity and signature.

More information

For more information about Enterprise Services authentication, see Chapter 9, Enterprise Services Security.

SQL Server authentication

SQL Server can authenticate users by using Windows authentication (NTLM or Kerberos) or can use its own built-in authentication scheme referred to as SQL authentication. The following two options are available:

- **SQL Server and Windows**. Clients can connect to an instance of Microsoft SQL Server by using either SQL Server authentication or Windows authentication. This is sometimes referred to as mixed mode authentication.
- **Windows Only**. The user must connect to the instance of Microsoft SQL Server by using Windows authentication.

More information

The relative merits of each approach are discussed in Chapter 12, "Data Access Security."

Authorization

The .NET Framework on Windows 2000 provides of the following authorization options:

- ASP.NET Authorization Options
- Enterprise Services Authorization
- SQL Server Authorization

ASP.NET authorization options

ASP.NET authorization options can be used by ASP.NET Web applications, Web services and remote components. ASP.NET provides the following authorization options:

- **URL Authorization**. This is an authorization mechanism, configured by settings within machine and application configuration files. URL Authorization allows you to restrict access to specific files and folders within your application's Uniform Resource Identifier (URI) namespace. For example, you can selectively deny or allow access to specific files or folders (addressed by means of a URL) to nominated users. You can also restrict access based on the user's role membership and the type of HTTP verb used to issue a request (GET, POST, and so on).

  URL Authorization requires an authenticated identity. This can be obtained by a Windows or ticket-based authentication scheme.

- **File Authorization**. File authorization applies only if you use one of the IIS-supplied Windows authentication mechanisms to authenticate callers and ASP.NET is configured for Windows authentication.

  You can use it to restrict access to specified files on a Web server. Access permissions are determined by Windows ACLs attached to the files.

- **Principal Permission Demands**. Principal permission demands can be used (declaratively or programmatically) as an additional fine-grained access control mechanism. They allow you to control access to classes, methods or individual code blocks based on the identity and group membership of individual users.
- **NET Roles**. .NET roles are used to group together users who have the same permissions within your application. They are conceptually similar to previous role-based implementations, for example Windows groups and COM+ roles. However, unlike these earlier approaches, .NET roles do not require authenticated Windows identities and can be used with ticket-based authentication schemes such as Forms authentication.

  .NET roles can be used to control access to resources and operations and they can be configured both declaratively and programmatically.

More information

For more information about ASP.NET authorization, see Chapter 8, ASP.NET Security.

Enterprise Services authorization

Access to functionality contained in serviced components within Enterprise Services applications is governed by Enterprise Services role membership. These are different from .NET roles and can contain Windows group or user accounts. Role membership is defined within the COM+ catalog and is administered by using the Component Services tool.

More information

For more information about Enterprise Services authorization, see Chapter 9, Enterprise Services Security.

SQL Server authorization

SQL Server allows fine-grained permissions that can be applied to individual database objects. Permissions may be based on role membership (SQL Server provides fixed database roles, user defined roles and application roles), or permission may be granted to individual Windows user or group accounts.

More information

For more information about SQL Server authorization, see Chapter 12, Data Access Security.

Gatekeepers and Gates

Throughout the remainder of this document, the term *gatekeeper* is used to identify the technology that is responsible for a *gate*. A gate represents an access control point (guarding a resource) within an application. For example, a resource might be an operation (represented by a method on an object) or a database or file system resource.

Each of the core technologies listed earlier provide gatekeepers for access authorization. Requests must pass through a series of gates before being allowed to access the requested resource or operation. The following describes the gates the requests must pass through:

- IIS provides a gate when you authenticate users (that is, you disable Anonymous authentication). IIS Web permissions can be used as an access control mechanism to restrict the capabilities of Web users to access specific files and folders. Unlike NTFS file permissions, Web permissions apply to all Web users, as opposed to individual users or groups. NTFS file permissions provide further restrictions on Web resources such as Web pages, images files, and so on. These restrictions apply to individual users or groups.

  IIS checks Web permissions, followed by NTFS file permissions. A user must be authorized by both mechanisms for them to be able to access the file or folder. A failed Web permission check results in IIS returning an HTTP 403–Access Forbidden response, whereas a failed NTFS permission check results in IIS returning an HTTP 401–Access Denied.

- ASP.NET provides various configurable and programmatic gates. These include URL Authorization, File Authorization, Principal Permission demands, and .NET Roles.
- The Enterprise Services gatekeeper uses Enterprise Services roles to authorize access to business functionality.
- SQL Server 2000 includes a series of gates that include server logins, database logins, and database object permissions.
- Windows 2000 provides gates using ACLs attached to secure resources.

The bottom line is that gatekeepers perform authorization based on the identity of the user or service calling into the gate and attempting to access a specific resource. The value of multiple gates is in-depth security with multiple lines of defense. Table 2.2 summaries the set of gatekeepers and identifies for each one the gates that they are responsible for.

Table 2.2. Gatekeepers responsibilities and the gates they provide

Gatekeeper	Gates
Windows Operating System	Logon rights (positive and negative, for example "Deny logon locally") Other privileges (for example "Act as part of the operating system") Access checks against secured resources such as the registry and file system.

	Access checks use ACLs attached to the secure resources, which specify who is and who is not allowed to access the resource and also the types of operation that may be permitted. TCP/IP filtering IP Security
IIS	Authentication (Anonymous, Basic, Digest, Integrated, Certificate) IP address and domain name restrictions (these can be used as an additional line of defense, but should not be relied upon due to the relative ease of spoofing IP addresses). Web permissions NTFS permissions
ASP.NET	URL Authorization File Authorization Principal Permission Demands .NET Roles
Enterprise Services	Windows (NTLM / Kerberos) authentication Enterprise Services (COM+) roles Impersonation levels
Web services	Uses gates provided by IIS and ASP.NET
Remoting	Uses gates provided by the host. If hosted in ASP.NET it uses the gates provided by IIS and ASP.NET. If hosted in a Windows service, then you must develop a custom solution.
ADO.NET	Connection strings. Credentials may be explicit or you may use Windows authentication (for example, if you connect to SQL Server)
SQL Server	Server logins Database logins Database object permissions

By using the various gates throughout the tiers of your application, you can filter out users that should be allowed access to your back-end resources. The scope of access is narrowed by successive gates that become more and more granular as the request proceeds through the application to the back-end resources.

Consider the Internet-based application example using IIS that is shown in Figure 2.4.

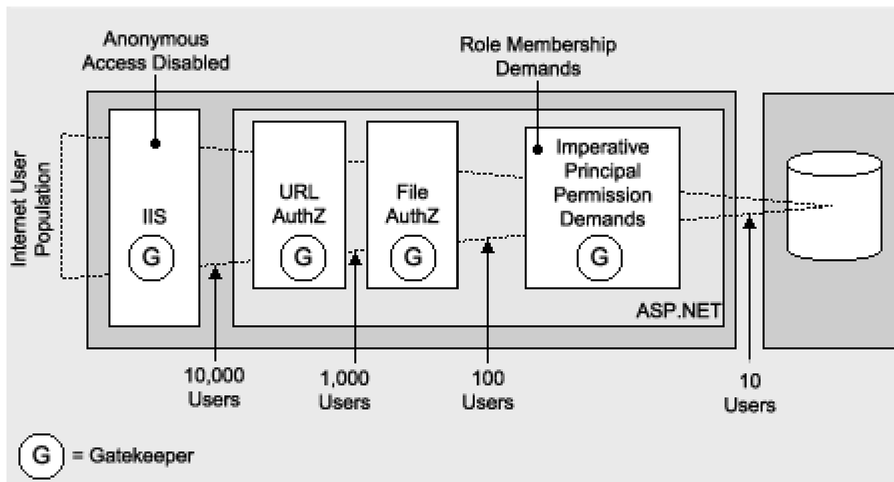

Figure 2.4. Filtering users with gatekeepers

Figure 2.4 illustrates the following:

- You can disable Anonymous authentication in IIS. As a result, only accounts that IIS is able to authenticate are allowed access. This might reduce the potential number of users to 10,000.
- Next, in ASP.NET you use URL Authorization, which might reduce the user count to 1,000 users.
- File authorization might further narrow access down to 100 users.
- Finally, your Web application code might allow only 10 users to access your restricted resource, based on specific role membership.

Identities and Principals

.NET security is layered on top of Windows security. The user centric concept of Windows security is based on security context provided by a logon session while .NET security is based on **IPrincipal** and **IIdentity** objects.

In Windows programming when you want to know the security context code is running under, the identity of the process owner or currently executing thread is consulted. With .NET programming, if you want to query the security context of the current user, you retrieve the current **IPrincipal** object from **Thread.CurrentPrincipal**.

The .NET Framework uses identity and principal objects to represent users when .NET code is running and together they provide the backbone of .NET role-based authorization.

Identity and principal objects must implement the **IIdentity** and **IPrincipal** interfaces respectively. These interfaces are defined within the **System.Security.Principal** namespace. Common interfaces allow the .NET Framework to treat identity and principal objects in a polymorphic fashion, regardless of the underlying implementation details.

The **IPrincipal** interface allows you to test role membership through an **IsInRole** method and also provides access to an associated **IIdentity** object.

```
public interface IPrincipal

{

  bool IsInRole( string role );

  IIdentity Identity {get;}

}
```

The **IIdentity** interface provides additional authentication details such as the name and authentication type.

```
public interface IIdentity

{

  string authenticationType {get;}

  bool IsAuthenticated {get;}

  string Name {get;}

}
```

The .NET Framework supplies a number of concrete implementations of **IPrincipal** and **IIdentity** as shown in Figure 2.5 and described in the following sections.

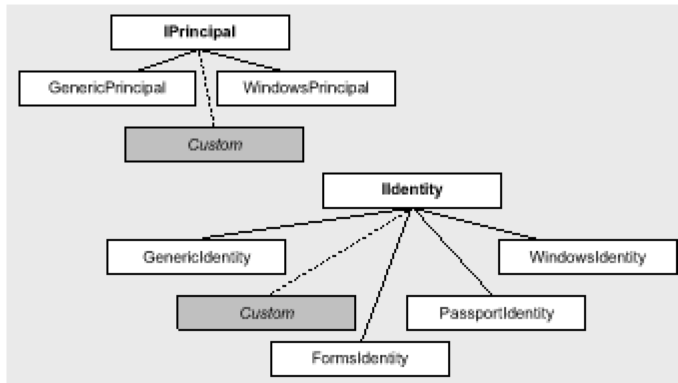

Figure 2.5. IPrincipal and IIdentity implementation classes

WindowsPrincipal and WindowsIdentity

The .NET version of a Windows security context is divided between two classes:

- **WindowsPrincipal**. This class stores the roles associated with the current Windows user. The **WindowsPrincipal** implementation treats Windows groups as roles. The **IPrncipal.IsInRole** method returns true or false based on the user's Windows group membership.
- **WindowsIdentity**. This class holds the identity part of the current user's security context and can be obtained from the static **WindowsIdentity.GetCurrent()**method. This returns a **WindowsIdentity** object that has a **Token** property that returns an **IntPtr** that represents a Windows handle to the access token associated with the current execution thread. This token can then be passed to native Win32® application programming interface (API) functions such as **GetTokenInformation**, **SetTokenInformation**, **CheckTokenMembership** and so on, to retrieve security information about the token.

  Note The static **WindowsIdentity.GetCurrent()**method returns the identity of the currently executing thread, which may or may not be impersonating. This is similar to the Win32 **GetUserName** API.

GenericPrincipal and Associated Identity Objects

These implementations are very simple and are used by applications that do not use Windows authentication and where the application does not need complex representations of a principal. They can be created in code very easily and as a result a certain degree of trust must exist when an application deals with a **GenericPrincipal**.

If you are relying upon using the **IsInRole** method on the **GenericPrincipal** in order to make authorization decisions, you must trust the application that sends you the **GenericPrincipal**. This is in contrast to using **WindowsPrincipal** objects, where you must trust the operating system to provide a valid **WindowsPrincipal** object with an authenticated identity and valid group/role names.

The following types of identity object can be associated with the **GenericPrincipal** class:

- **FormsIdentity**. This class represents an identity that has been authenticated with Forms authentication. It contains a **FormsAuthenticationTicket**, which contains information about the user's authentication session.
- **PassportIdentity**. This class represents an identity that has been authenticated with Passport authentication and contains Passport profile information.
- **GenericIdentity**. This class represents a logical user that is not tied to any particular operating system technology and is typically used in association with custom authentication and authorization mechanisms.

ASP.NET and HttpContext.User

Typically, **Thread.CurrentPrincipal** is checked in .NET code before any authorization decisions are made. ASP.NET, however, provides the authenticated user's security context using **HttpContext.User**.

This property accepts and returns an **IPrincipal** interface. The property contains an authenticated user for the current request. ASP.NET retrieves **HttpContext.User** when it makes authorization decisions.

When you use Windows authentication, the Windows authentication module automatically constructs a **WindowsPrincipal** object and stores it in **HttpContext.User**. If you use other authentication mechanisms such as Forms or Passport, you must construct a **GenericPrincipal** object and store it in **HttpContext.User**.

ASP.NET identities

At any given time during the execution of an ASP.NET Web application, there may be multiple identities present during a single request. These identities include:

- **HttpContext.User** returns an **IPrincipal** object that contains security information for the current Web request. This is the authenticated Web client.
- **WindowsIdentity.GetCurrent()**returns the identity of the security context of the currently executing Win32 thread. By default, this identity is ASPNET; the default account used to run ASP.NET Web applications. However, if the Web application has been configured for impersonation, the identity represents the authenticated user (which if IIS Anonymous authentication is in effect, is IUSR_MACHINE).
- **Thread.CurrentPrincipal** returns the principal of the currently executing .NET thread, which rides on top of the Win32 thread.

More information

- For a detailed analysis of ASP.NET identity for a combination of Web application configurations (both with and without impersonation), see ASP.NET Identity Matrix within the "Reference" section of this guide.
- For more information about creating your own **IPrincipal** implementation, see Chapter 8, ASP.NET Security, and How to Implement IPrincipal in the "Reference" section of this guide.

Remoting and Web Services

In the current version of the .NET Framework, Remoting and Web services do not have their own security model. They both inherit the security feature of IIS and ASP.NET.

Although there is no security built into the Remoting architecture, it was designed with security in mind. It is left up to the developer and/or administrator to incorporate certain levels of security in Remoting applications. Whether or not principal objects are passed across Remoting boundaries depends on the location of the client and remote object, for example:

- **Remoting within the same process**. When remoting is used between objects in the same or separate application domain(s), the remoting infrastructure copies a reference to the **IPrincipal** object associated with the caller's context to the receiver's context.
- **Remoting across processes**. In this case, **IPrincipal** objects are not transmitted between processes. The credentials used to construct the original **IPrincipal** must be transmitted to the remote process, which may be located on a separate computer. This allows the remote computer to construct an appropriate **IPrincipal** object based on the supplied credentials.

Summary

This chapter has introduced the full set of authentication and authorization options provided by the various .NET related technologies. By using multiple gatekeepers throughout your .NET Web application, you will be able to implement a defense-in-depth security strategy. To summarize:

- ASP.NET applications can use the existing security features provided by Windows and IIS.
- A combination of SSL and IPSec can be used to provide secure communications across the layers of a .NET Web application; for example, from browser to database.
- Use SSL to protect the clear text credentials passed across the network when you use Basic or Forms authentication.

- .NET represents users who have been identified with Windows authentication using a combination of the **WindowsPrincipal** and **WindowsIdentity** classes.
- The **GenericPrincipal** and **GenericIdentity** or **FormsIdentity** classes are used to represent users who have been identified with non-Windows authentication schemes, such as Forms authentication.
- You can create your own principal and identity implementations by creating classes that implement **IPrincipal** and **IIdentity**.
- Within ASP.NET Web applications, the **IPrincipal** object that represents the authenticated user is associated with the current HTTP Web request using the **HttpContext.User** property.
- Gates are access control points within your application through which authorized users can access resources or services. Gatekeepers are responsible for controlling access to gates.
- Use multiple gatekeepers to provide a defense-in-depth strategy.

The next chapter, Chapter 3, Authentication and Authorization, provides additional information to help you choose the most appropriate authentication and authorization strategy for your particular application scenario.

Authentication and Authorization

J.D. Meier, Alex Mackman, Michael Dunner, and Srinath Vasireddy
Microsoft Corporation

November 2002

Applies to:
Microsoft® ASP.NET

See the Landing Page for the starting point and a complete overview of *Building Secure ASP.NET Applications*.

Summary: This chapter provides guidance to help you develop an appropriate authorization strategy for your particular application scenario. It will help you choose the most appropriate authentication and authorization technique and apply them at the correct places in your application. (22 printed pages)

Contents

Designing an Authentication and Authorization Strategy
Authorization Approaches
Flowing Identity
Role-Based Authorization
Choosing an Authentication Mechanism
Summary

Designing an authentication and authorization strategy for distributed Web applications is a challenging task. The good news is that proper authentication and authorization design during the early phases of your application development helps to mitigate many top security risks.

This chapter will help you design an appropriate authorization strategy for your application and will also help answer the following key questions:

- Where should I perform authorization and what mechanisms should I use?
- What authentication mechanism should I use?
- Should I use Active Directory® directory service for authentication or should I validate credentials against a custom data store?
- What are the implications and design considerations for heterogeneous and homogenous platforms?
- How should I represent users who do not use the Microsoft® Windows® operating system within my application?
- How should I flow user identity throughout the tiers of my application? When should I use operating system level impersonation/delegation?

When you consider authorization, you must also consider authentication. The two processes go hand in hand for two reasons:

- First, any meaningful authorization policy requires authenticated users.
- Second, the way in which you authenticate users (and specifically the way in which the authenticated user identity is represented within your application) determines the available gatekeepers at your disposal.

  Some gatekeepers such as ASP.NET file authorization, Enterprise Services (COM+) roles and Windows ACLs, require an authenticated Windows identity (in the form of a **WindowsIdentity** object that encapsulates a Windows access token, which defines the caller's security context). Other gatekeepers, such as ASP.NET URL authorization and .NET roles, do not. They simply require an authenticated identity; one that is not necessarily represented by a Windows access token.

Designing an Authentication and Authorization Strategy

The following steps identify a process that will help you develop an authentication and authorization strategy for your application:

1. Identify resources
2. Choose an authorization strategy
3. Choose the identities used for resource access
4. Consider identity flow
5. Choose an authentication approach
6. Decide how to flow identity

Identify Resources

Identify resources that your application needs to expose to clients. Typical resources include:

- Web Server resources such as Web pages, Web services, static resources (HTML pages and images).
- 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 that 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.

Choose an Authorization Strategy

The two basic authorization strategies are:

- **Role based**. Access to operations (typically methods) 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).
- **Resource based**. Individual resources are secured using Windows ACLs. The application 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 impersonation approach severely impacts application scalability, because it means that connection pooling cannot be used effectively within the application's middle tier.

In the vast majority of .NET Web applications where scalability is essential, a role-based approach to authorization represents the best choice. For certain smaller scale intranet applications that serve per-user content from resources (such as files) that can be secured with Windows ACLs against individual users, a resource-based approach may be appropriate.

The recommended and common pattern for role-based authorization is:

- Authenticate users within your front-end Web application.
- Map users to roles.
- Authorize access to operations (not directly to resources) based on role membership.
- Access the necessary back-end resources (required to support the requested and authorized operations) by using fixed service identities. The back-end resource managers (for example, databases) *trust* the application to authorize callers and are willing to grant permissions to the trusted service identity or identities.

  For example, a database administrator may grant access permissions exclusively to a specific HR application (but not to individual users).

More information

- For more information about the two contrasting authorization approaches, see Authorization Approaches later in this chapter.
- For more information about role-based authorization and the various types of roles that can be used, see Role-Based Authorization later in this chapter.

Choose the Identities Used for Resource Access

Answer the question, "who will access resources?"

Choose the identity or identities that should be used to access resources across the layers of your application. This includes resources accessed from Web-based applications, and optionally Web services, Enterprise Services and .NET Remoting components. In all cases, the identity used for resource access can be:

- **Original caller's identity**. This assumes an impersonation/delegation model in which the original caller identity can be obtained and then flowed through each layer of your system. The delegation factor is a key criteria 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.
  - When a fixed Windows identity is required, use an Enterprise Services server application.
- **Custom identity**. When you don't have Windows accounts to work with, you can construct your own identities (using **IPrincipal** and **IIdentity** implementations) that can contain details that relate to your own specific security context. For example, these could include role lists, unique identifiers, or any other type of custom information.

  By implementing your custom identity with **IPrincipal** and **IIdentity** types and placing them in the current Web context (using **HttpContext.User**), you immediately benefit from built-in gatekeepers such as .NET roles and **PrincipalPermission** demands.

Consider Identity Flow

To support per-user authorization, auditing, and per-user data retrieval you may need to flow the original caller's identity through various application tiers and across multiple computer boundaries. For example, if a back-end resource manager needs to perform per-caller authorization, the caller's identity must be passed to that resource manager.

Based on resource manager authorization requirements and the auditing requirements of your system, identify which identities need to be passed through your application.

Choose an Authentication Approach

Two key factors that influence the choice of authentication approach are first and foremost the nature of your application's user base (what types of browsers are they using and do they have Windows accounts), and secondly your application's impersonation/delegation and auditing requirements.

More information

For more detailed considerations that help you to choose an authentication mechanism for your application, see Choosing an Authentication Mechanism later in this chapter.

Decide How to Flow Identity

You can flow identity (to provide security context) at the application level or you can flow identity and security context at the operating system level.

To flow identity at the application level, use method and stored procedure parameters. Application identity flow supports:

- Per-user data retrieval using trusted query parameters
-
```
SELECT x,y FROM SomeTable WHERE username="bob"
```
- Custom auditing within any application tier

Operating system identity flow supports:

- Platform level auditing (for example, Windows auditing and SQL Server auditing)
- Per-user authorization based on Windows identities

To flow identity at the operating system level, you can use the impersonation/delegation model. In some circumstances you can use Kerberos delegation, while in others (where perhaps the environment does not support Kerberos) you may need to use other approaches such as, using Basic authentication. With Basic authentication, the user's credentials are available to the server application and can be used to access downstream network resources.

More information

For more information about flowing identity and how to obtain an impersonation token with network credentials (that is, supports delegation), see Flowing Identity later in this chapter.

Authorization Approaches

There are two basic approaches to authorization:

- **Role based**. Users are partitioned into application-defined, logical roles. Members of a particular role share the same privileges within the application. Access to operations (typically expressed by method calls) is authorized based on the role-membership of the caller.

  Resources are accessed using fixed identities (such as a Web application's or Web service's process identity). The resource managers trust the application to correctly authorize users and they authorize the *trusted* identity.

- **Resource based**. Individual resources are secured using Windows ACLs. The ACL determines which users are allowed to access the resource and also the types of operation that each user is allowed to perform (read, write, delete and so on).

  Resources are accessed using the original caller's identity (using impersonation).

Role Based

With a role-based (or operations-based) approach to security, access to operations (not back-end resources) is authorized based on the role membership of the caller. Roles (analyzed and defined at application design time) are used as logical containers that group together users who share the same security privileges (or capabilities) within the application. Users are mapped to roles within the application and role membership is used to control access to specific operations (methods) exposed by the application.

Where within your application this role mapping occurs is a key design criterion; for example:

- On one extreme, role mapping might be performed within a back-end resource manager such as a database. This requires the original caller's security context to flow through your application's tiers to the back-end database.
- On the other extreme, role mapping might be performed within your front-end Web application. With this approach, downstream resource managers are accessed using fixed identities that each resource manager authorizes and is willing to trust.
- A third option is to perform role mapping somewhere in between the front-end and back-end tiers; for example, within a middle tier Enterprise Services application.

In multi-tiered Web applications, the use of trusted identities to access back-end resource managers provides greater opportunities for application scalability (thanks to connection pooling). Also, the use of trusted identities alleviates the need to flow the original caller's security context at the operating system level, something that can be difficult (if not impossible in certain scenarios) to achieve.

Resource Based

The resource-based approach to authorization relies on Windows ACLs and the underlying access control mechanics of the operating system. The application impersonates the caller and leaves it to the operating system in conjunction with specific resource managers (the file system, databases, and so on) to perform access checks.

This approach tends to work best for applications that provide access to resources that can be individually secured with Windows ACLs, such as files. An example would be an FTP application or a simple data driven Web application. The approach starts to break down 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.

The resource-based approach also 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.

Resource Access Models

The two contrasting approaches to authorization can be seen within the two most commonly used resource-access security models used by .NET Web applications (and distributed multi-tier applications in general). These are:

- The trusted subsystem model
- The impersonation/delegation model

Each model offers advantages and disadvantages both from a security and scalability perspective. The next sections describe these models.

The Trusted Subsystem Model

With this model, the middle tier service uses a fixed identity to access downstream services and resources. The security context of the original caller does not flow through the service at the operating system level, although the application may choose to flow the original caller's identity at the application level. It may need to do so to support back-end auditing requirements, or to support per-user data access and authorization.

The model name stems from the fact that the downstream service (perhaps a database) trusts the upstream service to authorize callers. Figure 3.1 shows this model. Pay particular attention to the trust boundary. In this example, the database *trusts* the middle tier to authorize callers and allow only authorized callers to access the database using the trusted identity.

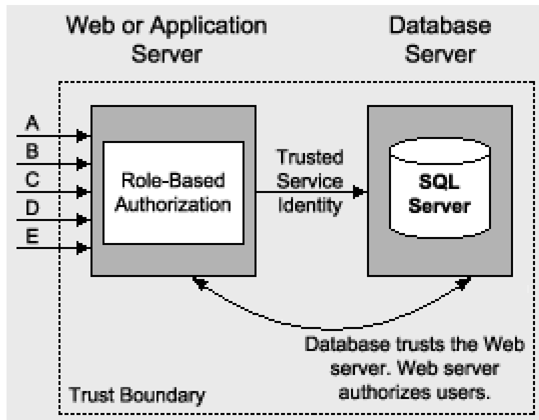

Figure 3.1. The Trusted Subsystem model

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

- Authenticate users
- Map users to roles
- Authorize based on role membership
- Access downstream resource manager using a fixed trusted identity

Fixed identities

The fixed identity used to access downstream systems and resource managers is often provided by a preconfigured Windows account, referred to as a service account. With a Microsoft SQL Server™ resource manager, this implies Windows authentication to SQL Server.

Alternatively, some applications use a nominated SQL account (specified by a user name and password in a connection string) to access SQL Server. In this scenario, the database must be configured for SQL authentication.

For more information about the relative merits of Windows and SQL authentication when communicating with SQL Server, see Chapter 12, Data Access Security.

Using multiple trusted identities

Some resource managers may need to be able to perform slightly more fine-grained authorization, based on the role membership of the caller. For example, you may have two groups of users, one who should be authorized to perform read/write operations and the other read-only operations.

Consider the following approach with SQL Server:

- Create two Windows accounts, one for read operations and one for read/write operations.

  More generally, you have separate accounts to mirror application-specific roles. For example, you might want to use one account for Internet users and another for internal operators and/or administrators.

- Map each account to a SQL Server user-defined database role, and establish the necessary database permissions for each role.
- Map users to roles within your application and use role membership to determine which account to impersonate before connecting to the database.

This approach is shown in Figure 3.2.

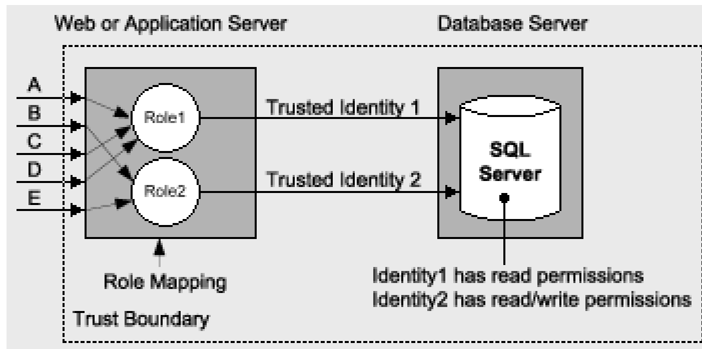

Figure 3.2. Using multiple identities to access a database to support more fine-grained authorization

The Impersonation / Delegation Model

With this model, a service or component (usually somewhere within the logical business services layer) impersonates the client's identity (using operating system-level impersonation) before it accesses the next downstream service. 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.

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 a couple of 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.

As a concrete example of this technique, a middle-tier Enterprise Services component might impersonate the caller prior to accessing a database. The database is accessed using a database connection tied to the security context of the original caller. With this model, the database authenticates each and every caller and makes authorization decisions based on permissions assigned to the individual caller's identity (or the Windows group membership of the caller). The impersonation/delegation model is shown in Figure 3.3.

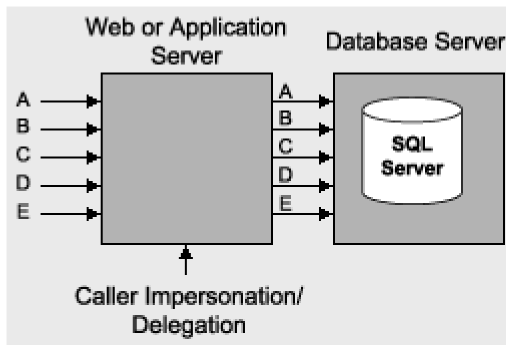

Figure 3.3. The impersonation/delegation model

Choosing a Resource Access Model

The trusted subsystem model is used in the vast majority of Internet applications and large-scale intranet applications, primarily for scalability reasons. The impersonation model tends to be used in smaller-scale applications where scalability is not the primary concern and those applications where auditing (for reasons of non-repudiation) is a critical concern.

Advantage of the impersonation / delegation model

The primary advantage of the impersonation / delegation model is auditing (close to the data). Auditing allows administrators to track which users have attempted to access specific resources. 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.

The impersonation / delegation model supports this by maintaining the user's security context for downstream resource access. This allows the back-end system to authoritatively log the user and the requested access.

Disadvantages of the impersonation / delegation model

The disadvantages associated with the impersonation / delegation model include:

- **Technology challenges**. Most security service providers don't support delegation, Kerberos is the notable exception.

  Processes that perform impersonation require higher privileges (specifically the Act *as part of the operating system* privilege). (This restriction applies to Windows 2000 and will not apply to Windows Server).

- **Scalability**. The impersonation / delegation model means that you cannot effectively use database connection pooling, because database access is performed by using connections that are tied to the individual security contexts of the original callers. This significantly limits the application's ability to scale to large numbers of users.
- **Increased administration effort**. ACLs on back-end resources need to be maintained in such a way that each user is granted the appropriate level of access. When the number of back-end resources increases (and the number of users increases), a significant administration effort is required to manage ACLs.

Advantages of the trusted subsystem model

The trusted subsystem model offers the following advantages:

- **Scalability**. The trusted subsystem model supports connection pooling, an essential requirement for application scalability. Connection pooling allows multiple clients to reuse available, pooled connections. It works with this model because all back-end resource access uses the security context of the service account, regardless of the caller's identity.
- **Minimizes back-end ACL management**. Only the service account accesses back-end resources (for example, databases). ACLs are configured against this single identity.
- **Users can't access data directly**. In the trusted-subsystem model, only the middle-tier service account is granted access to the back-end resources. As a result, users cannot directly access back-end data without going through the application (and being subjected to application authorization).

Disadvantages of the trusted subsystem model

The trusted-subsystem model suffers from a couple of drawbacks:

- **Auditing**. To perform auditing at the back end, you can explicitly pass (at the application level) the identity of the original caller to the back end, and have the auditing performed there. You have to trust the middle-tier and you do have a potential repudiation risk. Alternatively, you can generate an audit trail in the middle tier and then correlate it with back-end audit trails (for this you must ensure that the server clocks are synchronized).
- **Increased risk from server compromise**. In the trusted-subsystem model, the middle-tier service is granted broad access to back-end resources. As a result, a compromised middle-tier service potentially makes it easier for an attacker to gain broad access to back-end resources.

Flowing Identity

Distributed applications can be divided into multiple secure subsystems. For example, a front-end Web application, a middle-tier Web service, a remote component, and a database represent four different security subsystems. Each performs authentication and authorization.

You must identify those subsystems that must flow the caller's identity (and associated security context) to the next downstream subsystem in order to support authorization against the original caller.

Application vs. Operating System Identity Flow

Strategies for flowing identities include using the delegation features of the operating system or passing tickets and/or credentials at the application level. For example:

- To flow identity at the application level, you typically pass credentials (or tickets) using method arguments or stored procedure parameters.

  Note **GenericPrincipal** objects that carry the authenticated caller's identity do not automatically flow across processes. This requires custom code.

  You can pass parameters to stored procedures that allow you to retrieve and process user-specific data. For example:

  ```
  SELECT CreditLimit From Table Where UserName="Bob"
  ```

  This approach is sometimes referred to as a *trusted query parameter* approach.

- Operating system identity flow requires an extended form of impersonation called delegation.

Impersonation and Delegation

Under typical circumstances, threads within a server application run using the security context of the server process. The attributes that comprise the process' security context are maintained by the process' logon session and are exposed by the process level Windows access token. All local and remote resource access is performed using the process level security context that is determined by the Windows account used to run the server process.

Impersonation

When a server application is configured for impersonation, an impersonation token is attached to the thread used to process a request. The impersonation token represents the security context of the authenticated caller (or anonymous user). Any local resource access is performed using the thread impersonation token that results in the use of the caller's security context.

Delegation

If the server application thread attempts to access a remote resource, delegation is required. Specifically, the impersonated caller's token must have network credentials. If it doesn't, all remote resource access is performed as the anonymous user (AUTHORITY\ANONYMOUS LOGON).

There are a number of factors that determine whether or not a security context can be delegated. Table 3.1 shows the various IIS authentication types and for each one indicates whether or not the security context of the authenticated caller can be delegated.

Table 3.1. IIS Authentication types

Authentication Type	Can Delegate	Notes
Anonymous	Depends	If the anonymous account (by default IUSR_MACHINE) is configured in IIS as a local account, it cannot be delegated unless the local (Web server) and remote computer have identical local accounts (with matching usernames and passwords). If the anonymous account is a domain account it can be delegated.
Basic	Yes	If Basic authentication is used with local accounts, it can be delegated if the local accounts on the local and remote computers are identical. Domain accounts can also be delegated.
Digest	No	

Integrated Windows	Depends	Integrated Windows authentication either results in NTLM or Kerberos (depending upon the version of operating system on client and server computer). NTLM does not support delegation. Kerberos supports delegation with a suitably configured environment. For more information, see How To: Implement Kerberos Delegation for Windows 2000 in the References section of this guide.
Client Certificates	Depends	Can be delegated if used with IIS certificate mapping and the certificate is mapped to a local account that is duplicated on the remote computer or is mapped to a domain account. This works because the credentials for the mapped account are stored on the local server and are used to create an Interactive logon session (which has network credentials). Active Directory certificate mapping does not support delegation.

Important Kerberos delegation under Windows 2000 is unconstrained. In other words, a user may be able to make multiple network hops across multiple remote computers. To close this potential security risk, you should limit the scope of the domain account's reach by removing the account from the Domain Users group and allow the account to be used only to log on to specific computers.

Role-Based Authorization

Most .NET Web applications will use a role-based approach to authorization. You need to consider the various role types and choose the one(s) most appropriate for your application scenario. You have the following options:

- .NET roles
- Enterprise Services (COM+) roles
- SQL Server User Defined Database roles
- SQL Server Application roles

.NET Roles

.NET roles are extremely flexible and revolve around **IPrincipal** objects that contain the list of roles that an authenticated identity belongs to. .NET roles can be used within Web applications, Web services, or remote components hosted within ASP.NET (and accessed using the HttpChannel).

You can perform authorization using .NET roles either declaratively using **PrincipalPermission** demands or programmatically in code, using imperative **PrincipalPermission** demands or the **IPrincipal.IsInRole** method.

.NET roles with Windows authentication

If your application uses Windows authentication, ASP.NET automatically constructs a **WindowsPrincipal** that is attached to the context of the current Web request (using **HttpContext.User**). After the authentication process is complete and ASP.NET has attached to object to the current request, it is used for all subsequent .NET role-based authorization.

The Windows group membership of the authenticated caller is used to determine the set of roles. With Windows authentication, .NET roles are the same as Windows groups.

.NET roles with non-Windows authentication

If your application uses a non-Windows authentication mechanism such as Forms or Passport, you must write code to create a **GenericPrincipal** object (or a custom **IPrincipal** object) and populate it with a set of roles obtained from a custom authentication data store such as a SQL Server database.

Custom IPrincipal objects

The .NET Role-based security mechanism is extensible. You can develop your own classes that implement **IPrincipal** and **IIdentity** and provide your own extended role-based authorization functionality.

As long as the custom **IPrincipal** object (containing roles obtained from a custom data store) is attached to the current request context (using **HttpContext.User**), basic role-checking functionality is ensured.

By implementing the **IPrincipal** interface, you ensure that both the declarative and imperative forms of **PrincipalPermission** demands work with your custom identity. Furthermore, you can implement extended role semantics; for example, by providing an additional method such as **IsInMultipleRoles(string [] roles)** which would allow you to test and assert for membership of multiple roles.

More information

- For more information about .NET role-based authorization, see Chapter 8, ASP.NET Security.
- For more information about creating **GenericPrincipal** objects, see How To: Create GenericPrincipal Objects with Forms Authentication in the Reference section of this guide.

Enterprise Services (COM+) Roles

Using Enterprise Services (COM+) roles pushes access checks to the middle tier and allows you to use database connection pooling when connecting to back-end databases. However, for meaningful Enterprise Services (COM+) role-based authorization, your front-end Web application must impersonate and flow the original caller's identity (using a Windows access token) to the Enterprise Services application. To achieve this, the following entries must be placed in the Web application's Web.config file.

```
<authentication mode="Windows" />

<identity impersonate="true" />
```

If it is sufficient to use declarative checks at the method level (to determine which users can call which methods), you can deploy your application and update role membership using the Component Services administration tool.

If you require programmatic checks in method code, you lose some of the administrative and deployment advantages of Enterprise Services (COM+) roles, because role logic is hard-coded.

SQL Server User Defined Database Roles

With this approach, you create roles in the database, assign permissions based on the roles and map Windows group and user accounts to the roles. This approach requires you to flow the caller's identity to the back end (if you are using the preferred Windows authentication to SQL Server).

SQL Server Application Roles

With this approach, permissions are granted to the roles within the database, but SQL Server application roles contain no user or group accounts. As a result, you lose the granularity of the original caller.

With application roles, you are authorizing access to a specific application (as opposed to a set of users). The application activates the role using a built-in stored procedure that accepts a role name and password. One of the main disadvantages of this approach is that it requires the application to securely manage credentials (the role name and associated password).

More information

For more information about SQL Server user defined database roles and application roles, see Chapter 12, Data Access Security.

.NET Roles versus Enterprise Services (COM+) Roles

The following table presents a comparison of the features of .NET roles and Enterprise Services (COM+) roles.

Table 3.2. Comparing Enterprise Services roles with .NET roles

Feature	Enterprise Services Roles	.NET Roles
Administration	Component Services Administration Tool	Custom
Data Store	COM+ Catalog	Custom data store (for example, SQL Server or Active Directory)
Declarative	Yes [SecurityRole("Manager")]	Yes [PrincipalPermission(SecurityAction.Demand, Role="Manager")]
Imperative	Yes ContextUtil.IsCallerInRole()	Yes IPrincipal.IsInRole
Class, Interface and Method Level Granularity	Yes	Yes
Extensible	No	Yes (using custom IPrincipal implementation)
Available to all .NET components	Only for components that derive from ServicedComponent base class	Yes
Role Membership	Roles contain Windows group or user accounts	When using WindowsPrincipals, roles ARE Windows groups–no extra level of abstraction
Requires explicit Interface implementation	Yes To obtain method level authorization, an interface must be explicitly defined and implemented	No

Using .NET Roles

You can secure the following items with .NET roles:

- Files
- Folders
- Web pages (.aspx files)
- Web services (.asmx files)
- Objects
- Methods and properties
- Code blocks within methods

The fact that you can use .NET roles to protect operations (performed by methods and properties) and specific code blocks means that you can protect access to local and remote resources accessed by your application.

Note The first four items in the preceding list (Files, folders, Web pages, and Web services) are protected using the **UrlAuthorizationModule**, which can use the role membership of the caller (and the caller's identity) to make authorization decisions.

If you use Windows authentication, much of the work required to use .NET roles is done for you. ASP.NET constructs a **WindowsPrincipal** object and the Windows group membership of the user determines the associated role set.

To use .NET roles with a non-Windows authentication mechanism, you must write code to:

- Capture the user's credentials.
- Validate the user's credentials against a custom data store such as a SQL Server database.
- Retrieve a role list, construct a **GenericPrincipal** object and associate it with the current Web request.

  The **GenericPrincipal** object represents the authenticated user and is used for subsequent .NET role checks, such as declarative **PrincipalPermission** demands and programmatic **IPrincipal.IsInRole** checks.

More information

For more information about the process involved in creating a **GenericPrincipal** object for Forms authentication, see Chapter 8, ASP.NET Security.

Checking role membership

The following types of .NET role checks are available:

Important .NET role checking relies upon an **IPrincipal** object (representing the authenticated user) being associated with the current request. For ASP.NET Web applications, the **IPrincipal** object must be attached to **HttpContext.User**. For Windows Forms applications, the **IPrincipal** object must be attached to **Thread.CurrentPrincipal**.

- **Manual role checks**. For fine-grained authorization, you can call the **IPrincipal.IsInRole** method to authorize access to specific code blocks based on the role membership of the caller. Both AND and OR logic can be used when checking role membership.
- **Declarative role checks (gates to your methods)**. You can annotate methods with the **PrincipalPermissionAttribute** class (which can be shortened to **PrincipalPermission**), to declaratively demand role membership. These support OR logic only. For example you can demand that a caller is in at least one specific role (for example, the caller must be a teller or a manager). You cannot specify that a caller must be a manager and a teller using declarative checks.
- **Imperative role checks (checks within your methods)**. You can call **PrincipalPermission.Demand** within code to perform fine-grained authorization logic. Logical AND and OR operations are supported.

Role-checking examples

The following code fragments show some example role checks using programmatic, declarative, and imperative techniques.

1. Authorizing Bob to perform an operation:

   Note Although you can authorize individual users, you should generally authorize based on role membership, which allows you to authorize sets of users who share the same privileges within your application.

   - Direct user name check

   ```
   GenericIdentity userIdentity = new GenericIdentity("Bob");
   if (userIdentity.Name=="Bob")
   {
   }
   ```

   - Declarative check

   ```
   [PrincipalPermissionAttribute(SecurityAction.Demand,
     User="Bob")]
   ```

```
public void DoPrivilegedMethod()
{
}
```

- Imperative check

```
PrincipalPermission permCheckUser = new
  PrincipalPermission(
                                               "Bob",
                                                 null);
permCheckUser.Demand();
```

2. Authorizing tellers to perform an operation:

- Direct role name check

```
GenericIdentity userIdentity = new GenericIdentity("Bob");
// Role names would be retrieved from a custom data store
string[] roles = new String[]{"Manager", "Teller"};
GenericPrincipal userPrincipal = new
  GenericPrincipal(userIdentity,
                                                   roles);
if (userPrincipal.IsInRole("Teller"))
{
}
```

- Declarative check

```
[PrincipalPermissionAttribute(SecurityAction.Demand,
Role="Teller")]
void SomeTellerOnlyMethod()
{
}
```

- Ã‚Â…

- Imperative check

```
public SomeMethod()
{
  PrincipalPermission permCheck = new PrincipalPermission(
                                        null,"Teller");
  permCheck.Demand();
  // Only Tellers can execute the following code
  // Non members of the Teller role result in a security
    exception
  . . .
}
```

3. Authorize managers OR tellers to perform operation:

- Direct role name check

```
if (Thread.CurrentPrincipal.IsInRole("Teller") ||
    Thread.CurrentPrincipal.IsInRole("Manager"))
{
  // Perform privileged operations
}
```

- Declarative check

```
[PrincipalPermissionAttribute(SecurityAction.Demand,
  Role="Teller"),
 PrincipalPermissionAttribute(SecurityAction.Demand,
   Role="Manager")]
public void DoPrivilegedMethod()
{
Ã‚Â…
```

```
}
```

- Imperative check

```
PrincipalPermission permCheckTellers = new
  PrincipalPermission(
                                        null,"Teller");
PrincipalPermission permCheckManagers = new
  PrincipalPermission(
                                       null,"Manager");
(permCheckTellers.Union(permCheckManagers)).Demand();
```

4. Authorize only those people who are managers AND tellers to perform operation:

- Direct role name check

```
if (Thread.CurrentPrincipal.IsInRole("Teller") &&
    Thread.CurrentPrincipal.IsInRole("Manager"))
{
  // Perform privileged operation
}
```

- Declarative check

  It is not possible to perform AND checks with .NET roles declaratively. Stacking **PrincipalPermission** demands together results in a logical OR.

- Imperative check

```
PrincipalPermission permCheckTellers = new
  PrincipalPermission(
                                       null,"Teller");
permCheckTellers.Demand();
PrincipalPermission permCheckManagers = new
  PrincipalPermission(
                                     null, "Manager");
```

```
permCheckManagers.Demand();
```

Choosing an Authentication Mechanism

This section presents guidance that is designed to help you choose an appropriate authentication mechanism for common application scenarios. You should start by considering the following issues:

- **Identities**. A Windows authentication mechanism is appropriate only if your application's users have Windows accounts that can be authenticated by a trusted authority accessible by your application's Web server.
- **Credential management**. One of the key advantages of Windows authentication is that it enables you to let the operating system take care of credential management. With non-Windows approaches, such as Forms authentication, you must carefully consider where and how you store user credentials. The two most common approaches are to use:
  - SQL Server databases
  - User objects within Active Directory

  For more information about the security considerations of using SQL Server as a credential store, see Chapter 12, Data Access Security.

  For more information about using Forms authentication against custom data stores (including Active Directory), see Chapter 8, ASP.NET Security.
- **Identity flow**. Do you need to implement an impersonation/delegation model and flow the original caller's security context at the operating system level across tiers? For example, to support auditing or per-user (granular) authorization. If so, you need to be able to impersonate the caller and delegate their security context to the next downstream subsystem, as described in the "Delegation" section earlier in this chapter.
- **Browser type**. Do your users all have Internet Explorer or do you need to support a user base with mixed browser types? Table 3.3 illustrates which authentication mechanisms require Internet Explorer browsers, and which support a variety of common browser types.

  Table 3.3. Authentication browser requirements

Authentication Type	Requires Internet Explorer	Notes
Forms	No	
Passport	No	
Integrated Windows (Kerberos or NTLM)	Yes	Kerberos also requires Windows 2000 or later operating systems on the client and server computers and accounts configured for delegation. For more information, see How To: Implement Kerberos Delegation for Windows 2000 in the Reference section of this guide.
Basic	No	Basic authentication is part of the HTTP 1.1 protocol that is supported by virtually all browsers
Digest	Yes	
Certificate	No	Clients require X.509 certificates

Internet Scenarios

- The basic assumptions for Internet scenarios are:
  - Users do not have Windows accounts in the server's domain or in a trusted domain accessible by the server.

- Q295070, SSL (https) Connection Slow with One Certificate but Faster with Others

IPSec

The following articles in the Knowledge Base provides steps for troubleshooting IPSec issues.

- Q259335, Basic L2TP/IPSec Troubleshooting in Windows

Auditing and Logging

Windows Security Logs

Consult the Windows event and security logs early on in the problem diagnostic process.

More information

For more information on how to enable auditing and monitoring events, see the Knowledge Base and article Q300958, HOW TO: Monitor for Unauthorized User Access in Windows 2000.

SQL Server Auditing

By default, logon auditing is disabled. You can configure this either through SQL Server™ Enterprise Manager or by changing the registry.

SQL Server log files are by default located in the following directory. They are text-based and can be read with any text editor such as Notepad.

```
C:\Program Files\Microsoft SQL Server\MSSQL\LOG
```

To enable logon auditing with Enterprise Manager

1. Start Enterprise Manager.
2. Select the required SQL Server in the left hand tree control, right-click and then click **Properties**.
3. Click the **Security** tab.
4. Select the relevant Audit level—**Failure**, **Success** or **All**.

To enable logon auditing using a registry setting

1. Create the following **AuditLevel** key within the registry and set its value to one of the REG_DWORD values specified below.

```
2.      HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\AuditLevel
```

3. Set the value of this key to one of the following numeric values, which allow you to capture the relevant level of detail.

   3—captures both success and failed login attempts.

   2—captures only failed login attempts.

   1—captures only success login attempts.

   0—captures no logins.

It is recommended that you turn on failed login auditing as this is a way to determine if someone is attempting a brute force attack into SQL Server. The performance impacts of logging failed audit attempts are minimal unless you are being attacked, in which case you need to know anyway.

You can also set audit levels by using script against the SQL Server DMO (Database Management Objects), as shown in the following code fragment.

```
Sub SetAuditLevel(Server As String, NewAuditLevel As SQLDMO_AUDIT_TYPE)

    Dim objServer As New SQLServer2

    objServer.LoginSecure = True  'Use integrated security

    objServer.Connect Server      'Connect to the target SQL Server

    'Set the audit level

    objServer.IntegratedSecurity.AuditLevel = NewAuditLevel

    Set objServer = Nothing

End Sub
```

From SQL Server Books online, the members of the enumerated type, SQLDMO_AUDIT_TYPE are:

```
SQLDMOAudit_All     3 Log all authentication attempts - success or failure

SQLDMOAudit_Failure 2 Log failed authentication

SQLDMOAudit_None    0 Do not log authentication attempts

SQLDMOAudit_Success 1 Log successful authentication
```

Sample log entries

The following list shows some sample log entries for successful and failed entries in the SQL Server logs.

Successful login using Integrated Windows authentication:

```
2002-07-06 22:54:32.42 logon     Login succeeded for user

'SOMEDOMAIN\Bob'. Connection: Trusted.
```

Successful login using SQL standard authentication:

```
2002-07-06 23:13:57.04 logon     Login succeeded for user

'SOMEDOMAIN\Bob'. Connection: Non-Trusted.
```

Failed login:

```
2002-07-06 23:21:15.35 logon     Login failed for user

'SOMEDOMAIN\BadGuy'.
```

IIS Logging

IIS logging can be set to different formats. If you use W3C Extended Logging, then you can take advantage of some additional information. For example, you can turn on Time Taken to log how long a page takes to be served. This can be helpful for isolating slow pages on your production Web site. You can also enable URI Query which will log Query String parameters, which can be helpful for troubleshooting GET operations against your Web pages. The figure below shows the Extended Properties dialog box for IIS logging.

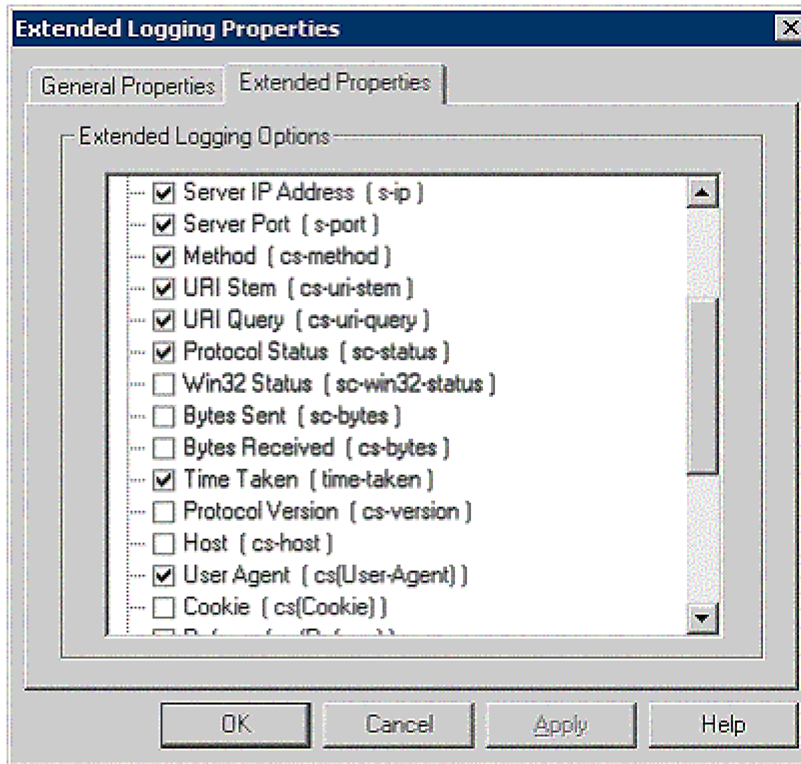

Figure 13.1. IIS extended logging properties

Troubleshooting Tools

The list of tools presented in this section can prove invaluable and will help you diagnose both security and non-security related problems.

File Monitor (FileMon.exe)

This tool allows you to monitor files and folders for access attempts. It is extremely useful to deal with file access permission issues. It is available from Sysinternals.com.

More information

For more information see the Knowledge Base article Q286198, HOWTO: Track 'Permission Denied' Errors on DLL Files.

Fusion Log Viewer (Fuslogvw.exe)

Fusion Log Viewer is provided with the .NET Framework SDK. It is a utility that can be used to track down problems with Fusion binding (see the .NET Framework documentation for more information).

To create Fusion logs for ASP.NET, you need to provide a log path in the registry and you need to enable the log failures option through the Fusion Log Viewer utility.

To provide a log path for your log files, use regedit.exe and add a directory location, such as e:\MyLogs, to the following registry key:

```
[HKLM\Software\Microsoft\Fusion\LogPath]
```

ISQL.exe

ISQL can be used to test SQL from a command prompt. This can be helpful when you want to efficiently test different logins for different users. You run ISQL by typing isql.exe at a command prompt on a computer with SQL Server installed.

Connecting by using SQL authentication

You can pass a user name by using the **-U** switch and you can optionally specify the password with the **-P** switch. If you don't specify a password, ISQL will prompt you for one. The following command, issued from a Windows command prompt, results in a password prompt. The advantage of this approach (rather than using the **-P** switch) is that the password doesn't appear on screen.

```
C:\ >isql -S YourServer -d pubs -U YourUser

Password:
```

Connecting by using Windows authentication

You can use the **-E** switch to use a trusted connection which uses the security context of the current interactively logged on user.

```
C:\ >isql -S YourServer -d pubs -E
```

Running a simple query

Once you are logged in, you can run a simple query, such as the one shown below.

```
1> use pubs

2> SELECT au_lname, au_fname FROM authors

3> go
```

To quit ISQL, type **quit** at the command prompt.

Windows Task Manager

Windows Task Manager on Windows XP and Windows Server 2003 allows you to display the identity being used to run a process.

To view the identity under which a process is running

1. Start **Task Manager**.
2. Click the **Processes** tab.
3. From the **View** menu, click **Select Columns**.
4. Select **User Name**, and click **OK**.

The user name (process identity) is now displayed.

Network Monitor (NetMon.exe)

NetMon is used to capture and monitor network traffic.

More information

See the following Knowledge Base articles:

- Q243270, HOW TO: Install Network Monitor in Windows 2000
- Q148942, HOW TO: Capture Network Traffic with Network Monitor
- Q252876, HOW TO: View HTTP Data Frames Using Network Monitor
- Q294818, Frequently Asked Questions About Network Monitor

There are a couple of additional tools to capture the network trace when the client and the server are on the same machine (this can't be done with Netmon):

- **tcptrace.exe**. Available from PocketSOAP.com. This is particularly useful for Web services since you can set it up to record and show traffic while your application runs. You can switch to Basic authentication and use tcptrace to see what credentials are being sent to the Web service.
- **packetmon.exe**. Available from AnalogX.com. This is a cut down version of Network Monitor, but much easier to configure.

Registry Monitor (regmon.exe)

This tool allows you to monitor registry access. It can be used to show read accesses and updates either from all processes or from a specified set of processes. This tool is very useful when you need to troubleshoot registry permission issues. It is available from Sysinternals.com.

WFetch.exe

This tool is useful for troubleshooting connectivity issues between IIS and Web clients. In this scenario, you may need to view data that is not displayed in the Web browser, such as the HTTP headers that are included in the request and response packets.

More information

For more information about this tool and the download, see the Knowledge Base article Q284285, How to Use Wfetch.exe to Troubleshoot HTTP Connections.

Visual Studio .NET Tools

The Microsoft .NET Framework SDK security tools can be found at the .NET Framework Tools Web site.

More information

See the following Knowledge Base articles:

- Q316365, INFO: ROADMAP for How to Use the .NET Performance Counters
- Q308626, INFO: Roadmap for Debugging in .NET Framework and Visual Studio

WebServiceStudio

This tool can be used as a generic client to test the functionality of your Web service. It captures and displays the SOAP response and request packets.

You can download the tool from the Web Service Tools page at GotDotNet.com.

Windows 2000 Resource Kits

Windows 2000 Resource Kits

Windows 2000 Resource Kit Free Tool Downloads