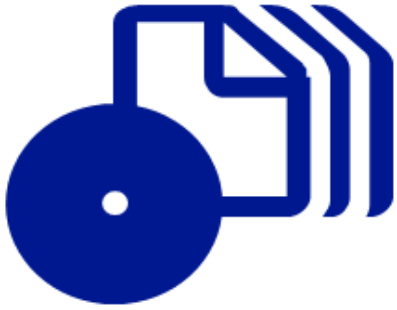
Microsoft

Internet Information Services (IIS) 7.0

Mike Volodarsky,
Olga Londer, Brett Hill,
Bernard Cheah, Steve Schofield,
Carlos Aguilar Mares, and Kurt Meyer
with the Microsoft® IIS Team

Resource Kit

How to access your CD files



The print edition of this book includes a CD. To access the CD files, go to http://aka.ms/624412/files, and look for the Downloads tab.

Note: Use a desktop web browser, as files may not be accessible from all ereader devices.

Questions? Please contact: mspinput@microsoft.com

Microsoft Press

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2008920571

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QWT 3 2 1 0 9 8

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to rkinput@microsoft.com.

Acquisitions Editor: Martin DelRe
Developmental Editor: Karen Szall
Project Editor: Victoria Thulman
Editorial Production: Custom Editorial Productions, Inc.
Technical Reviewers: Bob Dean, Bob Hogan; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Cover: Tom Draper Design; illustration by Todd Daman

Body Part No. X14-14918

Contents at a Glance

Table of Contents

Acknowledgments

The book that you now hold in your hands is the result of the collective effort of many people.

We'd like to start by thanking Bill Staples, Mai-Lan Tomsen Bukovec, and the whole IIS product team for their support. Several of us work in the IIS product team, and we know firsthand that we simply wouldn't be able to work on this book without the team's invaluable assistance.

Secondly, we are very grateful to Martin DelRe of Microsoft Press for his vision, his hard work in getting this project off the ground and ensuring its successful completion, and also for his never-ending support and encouragement.

It takes a lot of people and a lot of work to bring a book like this to life. There are several people in particular who we would like to acknowledge; the book would not be there without them. Brett Hill started this project and soldiered through till its completion. Special thanks to Mike Volodarsky, whose passion for quality and completeness resulted in him stepping up as the lead author. Kurt Meyer helped a lot as a project manager coordinating the writing and ensuring that the project milestones were not widely missed.

Many of our colleagues on the IIS product team had significant input into the book content. In fact, each chapter was reviewed by at least one member of the product team. Other product team members wrote the "Direct from the Source" sidebars that are peppered throughout the book, bringing you a unique insight into the design and development of IIS 7.0. We would like to express our sincere gratitude to the following members of the IIS product team who worked with us on this book, listed in alphabetical order by first name: Anil Ruia, Bill Staples, Edmund Chou, Eric Deily, Fabio Yeon, Jaroslav Dunajsky, Kanwaljeet Singla, Nazim Lala, Michael Brown, Thomas Marquardt, Tobin Titus, Ulad Malashanka, and Wade Hilmo.

We would also like to thank Tito Leverette for his guidance on and contributions to Chapter 17, "Performance and Tuning."

Many other teams in Microsoft provided technical reviews and shared their experience and insights. In particular, we are grateful to Tom Hawthorn of the Windows Performance team, as well as George Holman and the whole Microsoft.com Operations team. Nick McCollum of Quixtar Inc. also helped with technical reviews and suggestions in Chapters 5, 15, and 17.

Next, we would like to acknowledge our outstanding editorial team. In particular, we would like to thank the project editors, Karen Szall and Victoria Thulman of Microsoft Press, for their professionalism, mentoring, excellent editorial work, and, more than anything, their patience.

Bob Hogan and Bob Dean conducted the book technical reviews, ensuring the writing was consistent and easy to understand. Jean Findley of Custom Editorial Productions, Inc., did a great job managing the book production on a tight schedule.

In addition, we would like to thank Susan Chory and Isaac Roybal for helping us to get this project off the ground. We are also grateful to Simon Brown and Arvindra Sehmi for their encouragement for this work.

Thanks to everyone!

Sincerely,

The Author Team: Mike, Olga, Brett, Bernard, Steve, Carlos, and Kurt

Introduction

Welcome to the *Internet Information Services (IIS) 7.0 Resource Kit*! This book is a detailed technical resource for planning, deploying, and operating Microsoft Internet Information Services (IIS) 7.0, Microsoft's next generation Web server platform. Though this resource kit is intended primarily for IT professionals who have had experience with previous versions of IIS, anyone who is interested in learning about how to deploy and operate IIS 7.0 will find this resource kit extremely valuable.

Within this resource kit, you'll find in-depth information about the improvements introduced by IIS 7.0 and the underlying architectural concepts that will help you better understand the principles behind deploying and managing IIS 7.0 Web servers, and you'll discover techniques for taking advantage of new IIS 7.0 features and capabilities. You will also review detailed information and task-based guidance on managing all aspects of IIS 7.0, including deploying modular Web servers; configuring Web sites and applications; and improving Web server security, reliability, and performance. You'll also find numerous sidebars contributed by members of the IIS product team that provide deep insight into how IIS 7.0 works, best practices for managing the Web server platform, and invaluable troubleshooting tips. Finally, the companion media includes additional tools and documentation that you can use to manage and troubleshoot IIS 7.0 Web servers.

What's New in IIS 7.0

IIS 7.0 has been re-engineered at its core to deliver a modular and extensible Web server platform, forming the foundation for lean, low-footprint Web servers that power customized workloads and Web applications. The new extensible architecture enables the Web server to be completely customized; you can select only the required IIS features and add or replace them with new Web server features that leverage the new rich extensibility application programming interfaces (APIs). In addition, the Web server enables the use of a new distributed configuration system and management tools that simplify Web server deployment and management. The core feature set of IIS 7.0 continues to leverage the reliability and security-focused architecture established by its predecessor, IIS 6.0, and it adds additional improvements to enhance the reliability and security of the Web server platform. IIS 7.0 also includes extended support for application frameworks, including better integration with ASP.NET and built-in support for FastCGI-compliant application frameworks.

Among its many improvements, IIS 7.0 delivers the following:

- **Modular Web server architecture** Unlike its monolithic predecessors, IIS 7.0 is a completely modular Web server, containing more than 40 components that the administrator can individually install to create low-footprint, reduced surface-area Web server deployments that play a specific role in the application topology. Furthermore,

the new extensibility architecture enables any of the built-in modular features to be replaced with customized implementations that Microsoft and third parties provide.

- **.NET Extensibility through ASP.NET integration** The new ASP.NET integration capabilities enable you to develop IIS 7.0 features with the power of ASP.NET and the .NET Framework, reducing development and maintenance costs for custom Web server solutions. You can use existing ASP.NET services in this mode to enhance any application technologies, even those that were not developed with ASP.NET in mind. These abilities enable Web applications using IIS 7.0 to further customize the Web server to their needs without incurring the higher development costs associated with the previously used Internet Server Application Programming Interface (ISAPI).
- **Enhanced application framework support** In addition to improved ASP.NET integration for extending the Web server, IIS 7.0 provides more options for hosting other application frameworks. This includes the built-in support for the FastCGI protocol, a protocol used by many open source application frameworks such as PHP Hypertext Preprocessor (PHP) so that they can be reliably hosted in a Windows environment.
- **Distributed configuration system with delegation support** IIS 7.0 replaces the centralized metabase configuration store with a new configuration system based on a distributed hierarchy of XML files, which enables applications to control their own configuration. The new configuration system enables simplified application deployment without the overhead of required administrative involvement and provides the foundation for more flexible Web server configuration management.
- **Improved management tools** IIS 7.0 offers a host of management tools that leverage the new configuration system to provide more flexible and simpler configuration management for the Web server. This includes a brand new task-based IIS Manager tool, which offers remote delegated management; a new tool for command line management (Appcmd); and several APIs for managing Web server configuration from scripts, Windows Management Instrumentation (WMI), and .NET Framework programs.
- **Enhanced diagnostics and troubleshooting** IIS 7.0 provides diagnostic features to help diagnose Web server errors and troubleshoot hard-to-reproduce conditions with a Failed Request Tracing infrastructure. The diagnostic tracing features are integrated with ASP.NET applications to facilitate end-to-end diagnostics of Web applications.

Overview of Book

The four parts of this book cover the following topics:

- **Part I: Foundation** Provides an overview of IIS 7.0 features, describes the improvements introduced in IIS 7.0, and introduces the core architecture of the Web server

- **Part II: Deployment** Explains the modular installation architecture for deploying IIS 7.0 and provides procedures for installing IIS 7.0 for common Web server workloads
- **Part III: Administration** Describes the key concepts for managing IIS 7.0 and describes how to perform management tasks using the management tools that IIS 7.0 provides
- **Part IV: Troubleshooting and Performance** Describes how to use the logging and tracing infrastructure to provide for smooth operation of the Web server and troubleshoot error conditions, as well as how to monitor and improve Web server performance

The book also includes several appendixes on various topics and a glossary for reference.

Document Conventions

The following conventions are used in this book to highlight special features or usage.

Reader Aids

The following reader aids are used throughout this book to point out useful details.

Reader Aid	Meaning
Note	Underscores the importance of a specific concept or highlights a special case that might not apply to every situation
Important	Calls attention to essential information that should not be disregarded
Caution	Warns you that failure to take or avoid a specified action can cause serious problems for users, systems, data integrity, and so on
On the CD	Calls attention to a related script, tool, template, or job aid on the companion CD that helps you perform a task described in the text

Sidebars

The following sidebars are used throughout this book to provide added insight, tips, and advice concerning different IIS 7.0 features.

Sidebar	Meaning
Direct from the Source	Contributed by experts at Microsoft to provide from-the-source insight into how IIS 7.0 works, best practices for managing IIS 7.0, and troubleshooting tips
How It Works	Provides unique glimpses of IIS 7.0 features and how they work

Command Line Examples

The following style conventions are used in documenting command line examples throughout this book.

Style	Meaning
Bold font	Used to indicate user input (characters that you type exactly as shown)
Italic font	Used to indicate variables for which you need to supply a specific value (for example, *file_name* can refer to any valid filename)
Monospace font	Used for code samples and command line output
%SystemRoot%	Used for environment variables

Companion Media

The companion media is a valuable addition to this book and includes the following:

- **Electronic book** The complete text of the print book, in a searchable PDF eBook
- **Scripts** Scripts to help you automate IIS tasks
- **Tools** Links to tools for IIS, Windows® PowerShell, and more that you can put to use right away
- **Product information** Links to information about the features and capabilities of IIA NS Windows Server® 2008 and other products to help you optimize Windows Server 2008 in your enterprise
- **Resources** Links to guides, technical resources, webcasts, forums, and more to help you use and troubleshoot the features of IIS, Windows Server 2008, and other products
- **Sample Chapters** Preview chapters from 15 Windows Server 2008 books, in PDF format

Find Additional Content Online

As new or updated material becomes available that complements your book, it will be posted online on the Microsoft Press Online Windows Server and Client Web site. Based on the final build of Windows Server 2008, the type of material you might find includes updates to book content, articles, links to companion content, errata, sample chapters, and more. This Web site will be available soon at: *http://www.microsoft.com/learning/books/online/serverclient* and will be updated periodically.

Digital Content for Digital Book Readers: If you bought a digital-only edition of this book, you can enjoy select content from the print edition's companion CD.
Visit **http://www.microsoftpressstore.com/title/9780735624412.** to get your downloadable content. This content is always up-to-date and available to all readers.

Resource Kit Support Policy

We have made every effort to ensure the accuracy of this book and the content of the companion media. Microsoft Press provides corrections to this book through the Web at: *http://www.microsoft.com/learning/support/search.asp*.

If you have comments, questions, or ideas regarding the book or companion media content, or if you have questions that are not answered by querying the Knowledge Base, please send them to Microsoft Press by using either of the following methods:

E-mail:

rkinput@microsoft.com

Postal Mail:

Microsoft Press
Attn: *Microsoft Internet Information Services 7.0 Resource Kit,* Editor
One Microsoft Way
Redmond, WA 98052-6399

Please note that product support is not offered through the preceding mail addresses. For product support information, please visit the Microsoft Product Support Web site at: *http://support.microsoft.com*.

Chapter 12

Managing Web Server Modules

In this chapter:

Extensibility in IIS 7.0

On the Microsoft Internet Information Services (IIS) 7.0 team, there is a running joke that every release of IIS has to be a complete rewrite of the previous version. However, when looking at the history of the product, there have been strong reasons behind each of the rewrites. IIS 6.0, which was shipped with Windows Server 2003, was a complete rewrite of Windows XP's IIS 5.1, in the wake of the infamous CodeRed and Nimbda security exploits that plagued it in the summer of 2001. The rewrite, focused on producing an extremely reliable, fast, and secure Web server, was an overwhelming success—as evidenced by the rock-solid reliability and security track record of IIS 6.0 to date.

IIS 7.0 is again a major rewrite of the Web server, but this time for a different reason—to transform the reliable and secure codebase of IIS 6.0 into a powerful next-generation Web application platform. To achieve this, the IIS 7.0 release makes a huge investment in providing complete platform extensibility. The result? The most full-featured, flexible, and extensible Web server that Microsoft has ever released.

The extensible architecture of IIS 7.0 is behind virtually all of the critical platform improvements delivered in this release. Almost all functionality of the Web server, starting with the run-time Web server features and ending with configuration and the IIS Manager features, can be removed or replaced by third parties. This enables customers to build complete end-to-end solutions that deliver the functionality needed by their applications.

What's more, IIS 7.0's very own feature set is built on top of the same extensibility model third parties can take advantage of to further customize the server. This is a key concept, because it insures that the extensibility model available to third parties is at least powerful and flexible enough to build any of the features that come in the box. It also provides a unified way to think about and manage Web server features, whether it be IIS 7.0 built-in features or those provided by third parties. This is the heart of the modularity of IIS 7.0. This modularity,

together with the power of the extensibility model, enables you to turn your Web server into an efficient, specialized server that does exactly what you need and nothing more.

With IIS 7.0, you can for the first time:

- Build a low-footprint, reduced attack surface area Web server optimized for a specific workload.
- Replace any built-in feature with a custom feature developed in-house or by a third-party independent software vendor (ISV).
- Build complete end-to-end solutions that integrate seamlessly into the Web server, including request processing functionality, configuration, diagnostics, and administration.

Though traditionally topics concerning extensibility have been reserved mostly for developer audiences, with IIS 7.0, they become a critical component of deploying and operating the Web server. Thus, they necessitate a solid level of know-how from the IT staff. The ability to properly deploy, configure, tune, and lock down the feature set that comprises the Web server is critical to achieving a functional, scalable, and secure IIS production environment. Properly taking advantage of the flexibility IIS 7.0 offers can allow you to reap huge benefits in achieving small-footprint, fast, and secure Web server deployments. On the other hand, the complexity introduced by this same flexibility must be managed correctly, so that you can guarantee proper operation and reduce the total cost of ownership for your Web server farm.

This chapter takes the IT professional's perspective on the end-to-end extensibility platform provided by IIS 7.0. In this chapter, you will learn how to manage the modular feature set in IIS 7.0 to provide for an efficient, reliable, and secure IIS environment.

IIS 7.0 Extensibility Architecture at a Glance

The IIS 7.0 Web server platform is a complex system, including a number of parts necessary to operate, manage, and support Web applications running on top of it. This includes the Web server itself, the configuration system that supports the Web server and its features, the administration stack that provides an object model for managing the server, run-time state reporting application programming interfaces (APIs), and several management tools and APIs that expose the configuration and administration functionality to the user. Each of these subsystems provides a public extensibility layer and surfaces built-in functionality as modular components built on top of it. This design supports both server specialization as well as complete server customization via third-party additions or replacements to the built-in feature set. Figure 12-1 shows an overview of the extensibility architecture.

The main extensibility point lies in the Web server engine, which supports receiving HTTP requests, processing them (often with the help of application frameworks such as ASP.NET or PHP), and returning responses to the client. This is where most of the magic happens—IIS 7.0 ships with more than 40 Web server modules, which are responsible for everything from authentication, security, and response compression to performance enhancements and support for application frameworks such as ASP. These modules leverage one of the two

run-time extensibility models provided by IIS 7.0—the new C++ core extensibility model or the integrated ASP.NET extensibility model—both of which provide the flexibility to replace any built-in IIS 7.0 functionality or add new functionality of their own.

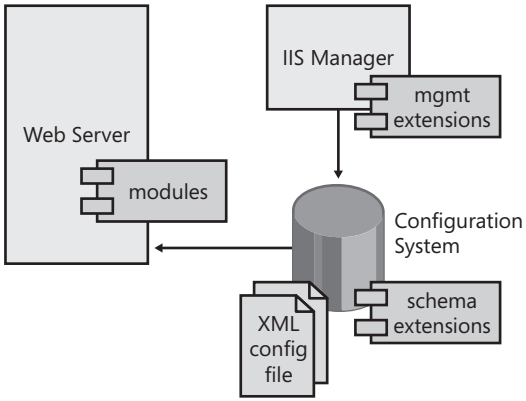


Figure 12-1 IIS 7.0 extensibility across the Web server, configuration system, and IIS Manager.

However, as you know, IIS is more than just a run time for processing requests. It also provides a brand new distributed configuration system for configuring the functionality of the Web server and its modules, with many features designed to simplify configuration, allow delegated configuration for non-administrator, and xcopy-based deployment of configuration settings with applications. In the face of increasing Web server complexity, the configuration system is more critical than ever before, and it is required to support the multitude of unique configurations and operational requirements of modern applications.

The IIS 7.0 team designed its configuration system to meet many of these challenges and to allow third-party solutions to do so by leveraging configuration extensibility. Just like the Web server features themselves, the configuration components leverage the same configuration extensibility layer that can be used to create custom configuration for third party Web server modules. This means that any custom Web server module can easily expose its own configuration settings, which can then be stored in the same configuration files and managed with the same standard APIs and tools that are used with the rest of the IIS 7.0 configuration.

In addition to static file-based configuration, IIS 7.0 provides support for administration objects, which enable dynamic configuration or management functionality to be exposed through the IIS 7.0 configuration object model. This enables new IIS management objects to be added or custom management functionality to be exposed on existing IIS objects, such as the site or application pool object, and consumed by the standard APIs and tools. Again, the administration stack is extensible, and IIS administration objects leverage that very extensibility model themselves.

Finally, IIS Manager (which replaces the old Microsoft Management Console-based InetMgr.exe) provides its own extensibility to enable graphical user interface (GUI) management pages to be added into IIS Manager, thus benefiting from the navigation, delegation, and remote management capabilities.

Together, these extensibility mechanisms provide the foundation for end-to-end solutions that can be developed on IIS 7.0, where custom Web server features can also expose custom configuration and management functionality, as well as a GUI administration experience with IIS Manager.

Direct from the Source

The IIS 7.0 release is a testament to the power of a system architecture that builds on its own extensibility foundation, rather than adding extensibility as an afterthought. Almost all components of the server, including the Web server features, their configuration, and IIS Manager management pages, are built on top of the same extensibility models that are exposed to third parties. Over the four years of working on the project, I've witnessed this power firsthand, through the unique opportunity to design and build both the core extensibility platform and the features that leverage it. In a way, IIS 7.0 has been our own best customer, helping us get the platform right before the real customers started to use it.

Mike Volodarsky

Program Manager, IIS 7.0

Managing Extensibility

The modular architecture serves as the foundation for many of the exciting new capabilities in IIS 7.0, from the ability to create specialized servers to securing and tuning server performance. However, it also exposes a fair amount of complexity to the administrator, which must be managed to harness the benefits of componentization. This moves the task of planning and managing extensibility to the IT domain, rather than being a developer-only task, as it has often been in the past. With this in mind, this chapter focuses on the key tasks around managing extensibility, rather than the information about developing extensibility components.

The first topic of interest is of course installing the extensibility components so that you can begin using them on the server. The built-in IIS 7.0 features are fully integrated with Windows Setup and can be installed using Server Manager on Windows Server 2008 and the Turn Windows Components On And Off UI on Windows Vista (you can learn more about this in Chapter 5, "Installing IIS 7.0"). Windows Setup implements all the information necessary to install and configure these components by using the IIS 7.0 configuration APIs. As such, these components typically do not require any additional work to be installed, although in some cases they may require additional configuration to control their availability to specific applications on the server.

However, this is not so for third-party components developed by ISVs or your own in-house development team. Without the support of Windows Setup, third-party components must be installed using the IIS 7.0 configuration directly. In doing so, it is often necessary to consider

deployment and installation options that suit your component. In fact, because the ability to customize and tailor IIS 7.0 to the specific application solutions often relies on leveraging its modularity, the ability to properly install IIS 7.0 extensibility components and manage the enabled feature set on the server becomes critically important. Thankfully, IIS 7.0 provides a number of management tools that can be used to perform the installation tasks, and so armed with the proper know-how, you can become a pro at deploying IIS 7.0 extensibility. To that end, this chapter describes how to install and manage enabled Web server modules. You can learn how to manage configuration and IIS Manager extensions in Chapter 13, "Managing Configuration and User Interface Extensions."

After covering initial deployment, you will also review common configuration and management tasks for each extensibility type. Though not always required, these tasks are often helpful to get your component to do exactly what you want in specific situations, and they include things such as insuring the correct execution order for your modules and enabling your modules to function correctly in a mixed 32-bit/64-bit environment. These tasks vary between the different extensibility types and are largely based on the developer's experience developing and using the extensibility layers during the development of IIS 7.0.

IIS 7.0 continues the IIS 6.0 tradition of emphasizing security, providing additional lockdown by default, and introducing new security features to help you further secure your Web server assets. One of the powerful ways to improve your server's security is to take advantage of componentization and remove all unused components, which will result in the smallest attack surface area possible for your server. When adding new components, then, you need to be aware of the resulting increase in the surface area of your server, and you must understand the security implications of the new code that is now running on your server. Proper understanding of the security impact of Web server components is critical to maintaining a secure operating environment and being able to take advantage of the functionality afforded by IIS 7.0 extensibility without compromising its security.

This chapter will cover what you need to know to securely deploy Web server modules and will review key security tactics you can use to lock down your server. You will also review the specific points to watch out for when configuring a shared hosting server or departmental server, which allows extensibility components to be published by nonadministrators. You can learn about securing configuration and IIS Manager extensions in Chapter 13.

Runtime Web Server Extensibility

The Web server extensibility model provides a foundation for IIS 7.0's modular architecture, enabling the low-footprint, low attack surface area, as well as highly specialized Web server deployments. All of this is possible because built-in IIS 7.0 features are implemented as pluggable modules on top of the same extensibility APIs that are exposed to third-party modules and are configured and managed with the same configuration and management tools.

What Is a Module?

Logically, a *module* is a Web server component that takes part in the processing of some or all requests and typically provides a service that can involve anything from supporting specific authentication methods (such as the Windows Authentication module) to recording and reporting requests that are currently executing (such as the Request Monitor module). The modules operate by executing during different stages of the request processing pipeline and influencing the request processing by using the APIs exposed by the Web server extensibility model. The majority of modules provide independent services to add functionality to the Web application or otherwise enhance the Web server.

The application developer or IT administrator can then essentially put together the Web server with the precise functionality that is required by controlling which modules are enabled for the application, much like building a structure from a set of LEGO blocks. IIS 7.0 provides a fine degree of control over which modules are enabled, giving administrators control of functionality on both the server as a whole and of specific applications, which we'll cover in depth later in this chapter.

Physically, IIS 7.0 modules are implemented as either native dynamic-link libraries (DLLs) developed on top of the new IIS 7.0 native C++ extensibility model, or as managed .NET Framework classes that leverage the new ASP.NET integration model available in IIS 7.0. Both of these APIs enable modules to participate in the IIS 7.0 request processing pipeline, and manipulate the request and response processing as needed. Though these two extensibility models use two different APIs and have a number of different characteristics from both the developer and IT administrator perspective, they both implement the logical module concept. This enables IIS 7.0 to provide a consistent development abstraction to both C++ and .NET Framework developers for extending the Web server. The logical module concept also enables IIS 7.0 to expose the administrator to a largely unified view of managing the Web server feature set. For information on the differences between native and managed Web server modules and how they affect the installation and management of modules, see the section titled "Differences Between Managed (.NET) and Native (C++) Modules" later in this chapter.

The Request Processing Pipeline

The IIS 7.0 request processing pipeline is the foundation of the modular architecture, enabling multiple independent modules to provide valuable services for the same request.

Note In IIS 7.0, the amount of processing the Web server engine itself performs is minimal, with the modules providing most of the request processing.

The pipeline itself is essentially a deterministic state machine that enables modules to interact with the request during a fixed set of processing stages, also known as *events*. As shown in

Figure 12-2, when the request is received, the state machine proceeds from the initial stage toward the final stage, raising the events and giving each module an opportunity to do its work during the stages it is interested in.

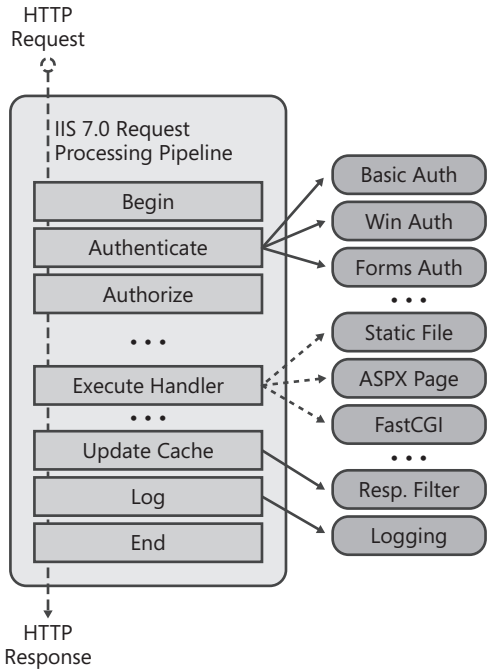


Figure 12-2 The request processing pipeline.

The majority of events in the request processing pipeline are intended for a specific type of task, such as authentication, authorization, caching, or logging. Modules that subscribe to these events can provide a specific service appropriate for the particular stage. For example, the authenticate event is home to a number of IIS 7.0 modules, including the Windows Authentication module (NTLM and Kerberos authentication), the Basic Authentication module, the ASP.NET Forms Authentication module, and so on. These events enable multiple modules to execute during the request processing and perform typical Web server processing tasks in the correct order. For example, determining the user associated with the request during the authentication stage needs to happen before determining whether that user has access to the requested resource during the authorization stage.

Other events are present for additional flexibility, enabling modules to perform tasks at a specific time during request processing (typically between the events that have specific intended roles such as authentication and authorization). Table 12-1 lists all the events, along with some IIS 7.0 modules that subscribe to them.

Table 12-1 Request Processing Events

Event	Description	Modules
BeginRequest	The request processing is starting.	Request Filtering module, IP Restrictions module
AuthenticateRequest	The authenticated user for the request is determined.	Authentication modules, including Windows Authentication module, Basic Authentication module, ASP.NET Forms Authentication module
AuthorizeRequest	Access to the requested resource is checked for the authenticated user, and the request is rejected if access is denied.	URL Authorization module
ResolveRequestCache	The server checks if the response to this request can be retrieved from a cache.	IIS Output Cache module, ASP.NET Output Cache module
MapRequestHandler	The handler for this request is determined.	
AcquireRequestState	The required state for this request is retrieved.	ASP.NET Session State module
PreExecuteRequestHandler	The server is about to execute the handler.	
ExecuteRequestHandler	The handler for the request executes and produces the response.	All modules that provide request handling, including Static File module, Directory Listing module, Default Document module, ISAPI extension module, ASP.NET PageHandler
ReleaseRequestState	The state is released.	ASP.NET Session State module
UpdateRequestCache	The cache is updated.	IIS 7.0 Output Cache module, ASP.NET Output Cache module
LogRequest	The request is logged.	Custom Logging module
EndRequest*	The request processing is about to finish.	Request Monitor module

* All of the events in this table except the *EndRequest* event also have a corresponding Post event, such as *PostBeginRequest* for *BeginRequest*. Post events exist primarily to provide additional flexibility to modules so that they can perform tasks that need to happen between specific events.

It is important to understand that though the majority of modules are self-contained and provide independent services during request processing, they do operate on a common set of request and response state and can affect the other's operation. In some cases, these relationships are part of formal patterns (such as the authentication and authorization pattern), and in others they may be unintentional. In the latter case, some modules may not be compatible

with each other, or they may require a specific ordering to function correctly. Module ordering is discussed in the section titled "Controlling Module Ordering" later in this chapter.

Differences Between Managed (.NET) and Native (C++) Modules

As we mentioned earlier, IIS 7.0 supports modules developed with the native IIS 7.0 C++ API as well as modules developed using the ASP.NET module API, sometimes referred to as *managed* modules.

The IIS 7.0 C++ module API replaces the legacy ISAPI filter and extension API as the new native extensibility model for IIS 7.0 and future versions of IIS. Existing ISAPI filters and extensions are still supported, but developers are encouraged to take advantage of the module extensibility model to build new server components. In fact, the support for ISAPI filters and extensions in IIS 7.0 is implemented as a native module, developed with the new native API, which hosts and executes ISAPI DLLs. Modules developed using the new native API are similar to the ISAPI filters in that they are Win32 DLLs loaded in-process by each IIS worker process, and they can affect the processing of every request. Because they execute under the rights and privileges of the IIS worker process, they have the same security impact and therefore have to be trusted by the server administrator.

However, this is where many of the similarities end, because IIS 7.0 modules use a much more refined and significantly more powerful C++ API, have access to many more extensibility points by subscribing to one or more of the events in the request processing pipeline, and can accomplish much richer tasks. The new C++ API also significantly improves the server development experience and reduces the potential for reliability issues that plagued the overly complex ISAPI interface. This makes IIS 7.0 native modules the most powerful—and yet simpler and more reliable—way to extend IIS.

Also, for the first time in the history of IIS, IIS 7.0 provides a full-fidelity .NET extensibility model based on ASP.NET. This makes server development significantly more accessible to developers and enables them to rapidly build server features while taking advantage of powerful features of ASP.NET and the .NET Framework. This is made possible by the new ASP.NET integration engine, which elevates ASP.NET from being an application framework to being a first-class extensibility mechanism for IIS 7.0.

As a server administrator, extending IIS with the .NET Framework enables you to delegate IIS extensibility to application owners who do not have administrator privileges on the server. This is possible because of the Code Access Security (CAS)–based ASP.NET hosting model, which constrains the execution of code in ASP.NET applications when configured to run with partial trust. Unlike native modules that execute with full privileges of the IIS worker process, managed ASP.NET modules can execute with limited privileges that can prevent them from negatively affecting the server itself or other applications on the server. This enables IIS 7.0 applications to deploy IIS features to the server without requiring administrative action (such as installing COM objects or ISAPI filters), without compromising server security.

Table 12-2 is a summary of the differences between native and managed modules.

Table 12-2 Comparing Native and Managed Modules

	Native Modules	Managed Modules
Developed with	IIS 7.0 C++ module API	ASP.NET module API, any .NET language
Represented by	Win32 DLL	.NET class in a .NET assembly DLL
Scope of execution	IIS worker process	ASP.NET application domain
Execution privilege	IIS worker process identity	IIS Worker process identity, plus constrained by the ASP.NET Trust Level
Deployment model	Globally for the entire server	Globally for the entire server by using the .NET Global Assembly Cache, or xcopy-deploy inside a specific application
Deployment privilege	Administrators only	Application owners can deploy with application

IIS 7.0 configuration is aware of the differences between native and managed modules. It also enables administrators to take full advantage of the constrained execution nature of managed modules by enabling managed modules to be added on a per-application basis by packaging them together with the application content. Application-based deployment of managed modules allows for simple xcopy deployment of IIS applications because they can specify their own IIS configuration and modules.

How It Works: ASP.NET Integrated Pipeline

With the unified pipeline model that IIS 7.0 provides, both native modules developed using the IIS 7.0 native extensibility model and managed modules developed using the ASP.NET module model can participate in the Web server's request processing (when using the ASP.NET Integrated mode). Both native and managed modules can participate in all request processing stages and operate on a shared set of request and response intrinsic objects.

In practice, however, ASP.NET and IIS are two separate software products. Moreover, ASP.NET Integrated mode uses the standard ASP.NET interfaces that are used to provide request processing services to the ASP.NET application framework on previous versions of IIS. How, then, is such a tight integration possible?

The answer lies in the special native module, ManagedEngine, that is installed on IIS 7.0 when the ".NET Extensibility" Windows Setup component (Windows Vista) or Role Service (Windows Server 2008) is installed. This module implements the ASP.NET Integrated mode engine that enables the ASP.NET request processing pipeline to be overlaid on the IIS request processing pipeline, proxying the event notifications and propagating the required request state to support the pipeline integration. This module is responsible for reading the managed modules and handler entries in the IIS module and handler configuration and working together with the new ASP.NET engine

implementation in *System.Web.dll* to set up the integrated pipeline. As a result, it enables ASP.NET modules and handlers to act as IIS modules and handlers.

So, when you see the ManagedEngine module in the IIS modules list, pay it some respect—it is arguably the most complex and powerful module ever written for IIS 7.0. Also keep in mind that this module must be present for the integrated pipeline and ASP.NET applications in general to work in IIS 7.0 Integrated mode application pools.

Mike Volodarsky

IIS Core Program Manager

However, IIS 7.0 also provides a consistent view of managing modules, whether they are native or managed, so that administrators can control the server feature set in a standard manner regardless of the module type. You will review the differences in the installation of native and managed modules, as well as standard management tasks, later in this section.

Installing Modules

The modules that comprise the IIS 7.0 feature set in Windows Vista or Windows Server 2008 can be installed via Windows Setup. Thanks to the modular architecture, Windows Setup enables very fine-grained installation of IIS 7.0 features—you can install most of the IIS 7.0 modules separately (along with all of their supporting configuration and administration features). You can also install the *.NET Extensibility* role service, which enables ASP.NET managed modules to run on IIS 7.0, or the *ASP.NET* role service, which also installs all of the of the ASP.NET managed modules and handlers to support fully functional ASP.NET applications. You can learn more about installing IIS 7.0 features in Chapter 5.

Windows Setup actually uses the same IIS 7.0 configuration APIs that you can use to manually install a third-party module on the server. In fact, Windows Setup uses Appcmd.exe, the IIS 7.0 command line tool, to perform module installation, which is just one of the ways that you can install modules on IIS 7.0. Later in this chapter, you will look at the most common ways to perform the installation, which are IIS Manager and Appcmd.exe, as well as editing server configuration directly. Of course you also have the option of using any of the programmatic APIs, including the .NET *Microsoft.Web.Administration* API, the IIS 7.0 configuration COM objects from C++ programs or script, or WMI. The choice is yours.

Installing Native Modules

To install a native module, it must be registered with the system.webServer/globalModules configuration section at the server level, in the ApplicationHost.config configuration file. Because only server administrators have access to this file, the installation of native modules requires Administrative privileges on the server. This is by design—allowing native code to execute in the IIS worker process is a potential security risk, and so Administrators must be sure to trust the source of the module.

The globalModules section contains an entry for each native module installed on the server, specifying the module *name* and the module *image*, which is the physical path to the module DLL.

```
        <globalModules>
            <add name="UriCacheModule"
image="%windir%\System32\inetsrv\cachuri.dll" />
            <add name="FileCacheModule"
image="%windir%\System32\inetsrv\cachfile.dll" />
            <add name="TokenCacheModule"
image="%windir%\System32\inetsrv\cachtokn.dll" />
            <add name="HttpCacheModule"
image="%windir%\System32\inetsrv\cachhttp.dll" />
            <add name="StaticCompressionModule"
image="%windir%\System32\inetsrv\compstat.dll" />
            <add name="DefaultDocumentModule"
image="%windir%\System32\inetsrv\defdoc.dll" />
            ...
        </globalModules>
```

The *image* attribute is an expanded string, which means that it can contain environment variables (as it does for modules installed by Windows Setup). This is a good practice to make sure that the ApplicationHost.config file remains portable and can be copied between servers and works on servers with different system drives.

Note Native module DLLs should be located on the server's local file system and not on remote network shares. This is because the server attempts to load them under the application pool identity and not the identity of the authenticated user or the configured virtual path (UNC) identity. This identity will not typically have access to network shares.

The act of registering a native module instructs IIS worker processes in all application pools to load the module DLL. The globalModules configuration section is also one of the few sections that cause IIS worker processes to recycle whenever changes are made. This means that you can install new modules, or uninstall existing modules, and IIS will automatically pick up those changes without needing to manually recycle application pools, restart IIS services, or run IISRESET.

Note By adding the module to globalModules, you are instructing IIS worker processes to load the module DLL. This alone does not enable the module to run. To do that, you also need to enable the module on the server or for a particular application.

After the module is installed, it will be loaded by all IIS worker processes on the server. Unfortunately, IIS 7.0 does not enable native modules to be installed for a particular application pool, so there is no easy way to load a native module only into certain application pools and not into others.

Note IIS 7.0 does provide a way to load native modules selectively into a specific application pool, by using the application pool name preconditions. See the section titled "Understanding Module Preconditions" later in this chapter for more information on this. Though loading native modules in this way is possible, you should not use this mechanism in most situations because of its management complexity.

However, loading the module alone is *not* sufficient to enable the module to execute. It also needs to be enabled by listing its name in the system.webServer/modules configuration section. This is an important distinction that serves to provide more flexible control over the enabled module set. Unlike the globalModules section, which can only be specified at the server level, the modules configuration section can be specified at the application level, such as in the application's root Web.config. This enables each application to control the set of enabled modules that process requests to itself.

Typically, a native module is also enabled at the server level (in ApplicationHost.config) during its installation, which enables it for all applications on the server by default (except for applications that specifically remove it in their configuration). This is the case for most of the built-in native modules.

```
<modules>
    <add name="HttpCacheModule" />
    <add name="StaticCompressionModule" />
    <add name="DefaultDocumentModule" />
    ...
<modules>
```

Each native module is enabled simply by listing its name in the modules collection.

Inside Global Web Server Events

If you read the globalModules section carefully, you will notice that some modules, such as the TokenCacheModule, are listed there but yet are not listed in the modules list by default. Does this mean that this module is disabled by default? No, not entirely.

Native modules loaded inside the IIS worker process can participate in global server events, which are events that are not associated with request processing. These events enable native modules to extend certain server functionality at the worker process level, such as by providing the ability to cache logon tokens for improved performance.

Native modules that offer this kind of global functionality do not need to be listed in the modules list and are able to offer it by simply being loaded in the worker process. However, only modules listed in the modules list can provide request processing functionality.

Note When the modules section changes, the IIS worker process does not need to recycle. Instead, it picks up the changes automatically and applies the resulting module set to subsequent requests. However, ASP.NET applications whose modules configuration changes will restart.

Uninstalling Native Modules

Caution Before removing modules, you should consider the security and performance implications that the module removal will have on your server. The section titled "Securing Web Server Modules" later in this chapter covers these implications in more detail.

To uninstall a native module, you need to remove the corresponding module entry from the globalModules list. This prevents the module from being loaded in IIS worker processes on the entire server. Removing the module from globalModules causes all IIS worker processes to gracefully recycle.

In addition, when the native module is removed from the globalModules list, references to it in the modules list also must be removed. Otherwise, all requests to the server or an application that enables the missing module will generate an "HTTP 500 – Internal Server Error" error until the module entry is removed. Typically, you should remove both the globalModules and modules entry for the module at the same time. However, if you do it in two separate steps, changing the modules section will not cause a worker process recycle—IIS will automatically pick up this change by recycling any affected applications. Be sure to make a configuration backup in case you need to restore the original configuration later.

When uninstalling a native module that is part of the IIS 7.0 feature set, you should instead uninstall the corresponding IIS Windows Setup component (Windows Vista) or Role Service (Windows Server 2008). Doing so has the benefit of removing the module binaries and related configuration components, as well as indicating that the corresponding feature is not installed to the Windows Setup infrastructure. The binaries remain stored in the OS installation cache, where they are inaccessible to anyone other than the OS TrustedInstaller subsystem. This ensures that you can re-install these modules later, and that any required patches are applied to these binaries even when the patches are not installed on your server.

Caution You should *not* remove built-in IIS 7.0 modules manually. Use Windows Setup instead to uninstall the corresponding feature or role service.

When a custom module is uninstalled and all IIS worker processes have recycled, you can remove the module binary from the machine if necessary.

Look in the sections titled "Using IIS Manager to Install and Manage Modules" and "Using Appcmd to Install and Manage Modules" later in this chapter to find steps detailing how you can use IIS Manager or the Appcmd command line tool to uninstall a native module.

Installing Managed Modules

Managed modules developed using the ASP.NET APIs are not required to be installed globally on the server. Instead, they simply need to be enabled in configuration for the application where they are to be used, similar to classic ASP.NET applications in previous versions of IIS. This enables simple xcopy deployment of applications containing managed modules, since unlike native modules they do not require Administrative privileges to be deployed.

Needless to say, this makes managed modules very appealing in scenarios in which the application administrator does not have administrative privileges on the server, such as on shared hosting servers or departmental servers. Such applications can now deploy Web server features without contacting the server administrator to install a global and trusted component, which is often not possible. In these environments, the server administrator can constrain the execution of managed modules by limiting the trust of the ASP.NET applications. The section titled "Securing Web Server Modules" later in this chapter covers constraining the execution of managed modules in more detail and discusses locking down module extensibility.

Note Running managed modules requires installation of the ".NET Extensibility" Windows Setup component (Windows Vista) or Role Service (Windows Server 2008). This installs the ManagedEngine module that enables managed modules to run inside Integrated mode applications pools.

Installing the "ASP.NET" setup component/role service automatically installs the ".NET Extensibility" component and also adds the modules and handler mappings used by the ASP.NET framework. It also installs the classic ASP.NET handler mappings that enable application pools that use Classic integration mode to run ASP.NET using the legacy ASPNET_ISAPI.dll integration mechanism.

To install a managed module, the module simply needs to be added to the modules configuration section. This is the same section that enables installed native modules, except managed modules do not have to be listed in the globalModules configuration section. The modules section, therefore, provides a unified view of enabled modules, whether they are native or managed. Because this configuration section can be delegated down to the application level, each application can specify the complete set of enabled modules (managed or native) by using its modules configuration. Here is a more complete example of the modules configuration section at the server level after the ASP.NET feature is installed.

```
<modules>
    <add name="HttpCacheModule" />
    <add name="StaticCompressionModule" />
    <add name="DefaultDocumentModule" />
    <add name="DirectoryListingModule" />
```

```
                ...
                <add name="FormsAuthentication"
type="System.Web.Security.FormsAuthenticationModule"
preCondition="managedHandler" />
                <add name="DefaultAuthentication"
type="System.Web.Security.DefaultAuthenticationModule"
preCondition="managedHandler" />
                <add name="RoleManager"
type="System.Web.Security.RoleManagerModule" preCondition="managedHandler"
/>
                 ...
            </modules>
```

As you can see, this section contains both native modules that are simply identified by the *name* attribute, and managed modules that also specify a *type* attribute. For each application, the server resolves the enabled modules by looking up the native modules' names in the globalModules section and directly loading the specified .NET type for managed modules. The type is the fully qualified .NET type name that refers to the class that implements this module and resolves to an assembly that is packaged with the application or an assembly installed in the machine's Global Assembly Cache (GAC).

Important ASP.NET applications that define modules in the system.web/httpModules configuration section and ASP.NET handler mappings in the system.web/httpHandlers configuration section need to have their configurations migrated to the IIS system.webServer/modules and system.webServer/handlers configuration sections to operate correctly in Integrated mode. The server will generate a HTTP 500 error notifying you of this requirement if you attempt to run such an application in Integrated mode. You can migrate the application easily by using the *Appcmd Migrate Config ApplicationPath* command. To learn more about why this is necessary and the options you have for running legacy ASP.NET applications, see Chapter 11, "Hosting Application Development Frameworks."

Deploying Assemblies Containing Managed Modules

Managed modules are classes implemented inside .NET assemblies. To support delegated deployment of managed modules, the server provides several options, as shown in Table 12-3, for deploying the module assemblies so that they can be added both globally on the server and for a specific application only.

Table 12-3 Managed Modules and Deployment Options

Deployment Option	Assembly Location	Module Registration Location
Server	Global Assembly Cache (GAC)	Server level modules section in ApplicationHost.config
Application	Assembly in application's /BIN directory OR Source code in application's /App_Code directory	Application's modules section in application root's Web.config

Deploying the Module Assembly at the Server Level If the module is to be installed globally for all applications on the server, it needs to be registered with the machine's Global Assembly Cache (GAC). Before the managed assembly can be deployed to the GAC, it needs to be strongly signed by the developer (for more information on strongly signing .NET assemblies, see *http://msdn2.microsoft.com/en-us/library/xc31ft41.aspx*). In particular, Microsoft Visual Studio makes the signing process simple. Then, the managed assembly can be added to the GAC by running the following command.

```
gacutil.exe /if AssemblyPath
```

Note The gacutil.exe command line tool is *not* available in the .NET Framework run-time installation that comes with the operating system, so you have to download the .NET Framework SDK to obtain it. After you obtain it, though, you can copy the tool to use on other machines.

After your assembly is added to the Global Assembly Cache, you can add any of the modules it contains to the server level modules section by specifying their type. This type name must be fully qualified; that is, it must contain the full namespace path to the class (for example, *System.Web.Security.FormsAuthenticationModule*). Because when it creates your module, ASP.NET needs to locate an assembly that contains this type, the assembly must either be listed in the system.web/compilation/assemblies configuration collection or included in the type name by using the strong name notation. Here is an example of a strong name for the built-in FormsAuthentication module.

```
System.Web.Security.FormsAuthenticationModule, System.Web, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=x86
```

Note You can get the assembly part of the strong name by using the gacutil.exe tool you used earlier when you installed the assembly to the Global Assembly Cache. Run "gacutil.exe /l *AssemblyName*" to display the assembly's strong name signature. You can omit all parts of the assembly's strong name signature except for the assembly name, and ASP.NET will attempt to find the first matching assembly based on the attributes you do include.

You may wonder why the default module entries for ASP.NET modules do not specify the strong names and simply specify the fully qualified type names. This is because their parent assembly, *System.Web.dll*, is configured to be automatically preloaded by the ASP.NET applications (by being listed in the system.web/compilation/assemblies configuration collection in .NET Framework's root Web.config). Thus, ASP.NET can locate the types of built-in ASP.NET modules by searching the preloaded assemblies, without having to specify the assembly signature in the module type string.

Deploying the Module Assembly with the Application If the module is to be available in a specific application only, it can be xcopy-deployed with that application without registering

anything globally on the server. In this case, the application owner can provide the module in two ways: as a compiled .NET assembly DLL in the /BIN subdirectory of the application root or as a source code file in the /App_Code subdirectory of the application root.

Note It is not necessary to sign the assembly in the application's /BIN subdirectory.

The /App_Code deployment model is more appropriate for development and test environments, because it enables editing of the module source code on the live server without recompiling the module DLL. The /BIN deployment model is recommended for production servers, because it does not require run-time compilation of the assembly and provides a more compact way to deploy large codebases than source code does.

Because the module type deployed inside the application is available only in the application, it can be used only in that application (unlike assemblies placed in the Global Assembly Cache, which are available to all applications on the server). To add the module, you simply need to add the fully qualified type into the modules configuration section in the application's root Web.config file. For modules whose assemblies are in the /BIN directory, you can optionally specify the assembly name, although it is not necessary—ASP.NET by default preloads all /BIN assemblies. This is also true for modules that are deployed as source code to the /App_Code directory, because ASP.NET automatically compiles and loads it.

Packaging IIS 7.0 managed modules in the application is a powerful way to create self-contained applications that can be xcopy-deployed to a server and immediately function without globally installing any functionality on the server.

Be sure to also read the section titled "Locking Down Extensibility" later in this chapter to understand the security impact of allowing managed module delegation and how to properly control it.

Uninstalling Managed Modules

Caution Before removing modules, you should consider the security and performance implications that this action will have on your server. You can find more detail in the section titled "Securing Web Server Modules" later in this chapter.

Unlike native modules, you can install managed modules simply by adding them to the modules list. Therefore, uninstalling managed modules is identical to disabling them and requires a single step.

Managed modules installed as part of ASP.NET installation cannot be individually uninstalled using Windows Setup (Windows Vista) or Server Manager (Windows Server 2008). So, if you need to remove any one of them, you have to do so by manually removing their entries

from the modules section. Be sure to make a configuration backup in case you need to restore the original configuration later.

The section titled "Controlling What Modules Are Enabled" later in this chapter discusses removing managed modules. Look in the sections titled "Using IIS Manager to Install and Manage Modules" and "Using Appcmd to Install and Manage Modules" later in this chapter to find steps detailing how you can use IIS Manager or the Appcmd command line tool to remove managed modules.

Understanding Module Preconditions

The modular architecture of IIS 7.0 relies heavily on controlling which modules are installed and enabled on the server and at the application level. Sometimes, making this determination based on static configuration is not sufficient, and the decision to use the module in a specific scenario must be made based on factors known only at run time. To support this functionality, IIS 7.0 introduces the concept of preconditions, which are configured conditions that the server evaluates at run time to determine whether a particular module should be used.

The following types of preconditions are supported:

- **Module load preconditions** These preconditions may be associated with each installed native module in the globalModules configuration section, and they determine whether a particular module is loaded by each worker process when it starts. If any of the preconditions do not evaluate to true, the module is not loaded in the worker process. These preconditions can also be used in the isapiFilters configuration section to control the loading of ISAPI filters.
- **Module enablement preconditions** These preconditions may be associated with each enabled module in the modules configuration section, and they determine whether the module is enabled for a particular application (or request). If any of the preconditions do not evaluate to true, the module does not run.
- **Handler mapping preconditions** These preconditions may be associated with each handler mapping in the handlers configuration section, and they determine whether this handler mapping is considered when mapping a request to handlers. If any of the preconditions do not evaluate to true, the handler mapping is ignored.

In each case, one or more precondition strings may be specified to allow the configuration entry to be selectively used in cases where all of the specified preconditions evaluate to true. If any of the preconditions fail, the module is not loaded or enabled, or the handler mapping is not considered, depending on the scenario in which the precondition is being used. Here is an example of ASP.NET setup using preconditions to load the "ManagedEngine" native module only in application pools that use Integrated mode, run Framework version 2.0, and are configured to execute in a 32-bit mode.

```
<globalModules>
  …
  <add name="ManagedEngine"
```

```
image="%windir%\Microsoft.NET\Framework\v2.0.50727\webengine.dll"
preCondition="integratedMode,runtimeVersionv2.0,bitness32" />
</globalModules>
```

Table 12-4 lists the supported precondition strings and scenarios in which they can be used.

Table 12-4 Precondition Strings

Precondition	Applicable To
bitness32, bitness64 Matches the "bitness" of the application pool	globalModules, isapiFilters, modules, handlers
classicMode, integratedMode Matches the configured managed pipeline mode of the application pool	globalModules, isapiFilters, modules, handlers
runtimeVersionv1.1, **runtimeVersionv2.0** Matches the configured .NET run-time version of the application pool	globalModules, isapiFilters, modules, handlers
appPoolName=*Name*, appPoolName!=*Name* Matches the application pool name; this precondition can be used to selectively load a native module into a specific application pool	globalModules, isapiFilters, modules, handlers
managedHandler Matches requests to handler mappings with managed handlers	modules only

The bitness32 and bitness64 preconditions match the bitness of the worker process and can be used to selectively load modules in 32-bit or 64-bit application pools. In mixed 32-bit and 64-bit IIS environments, it may be necessary to load 32-bit native modules only in 32-bit application pools, because IIS will fail to load the 32-bit native DLLs into the 64-bit worker process. To help with this, the 32-bit native modules should configure the bitness32 precondition, which selectively loads them in 32-bit application pools only. For more information about running IIS in mixed 32-bit and 64-bit, please refer to the section titled "Installing Modules for x64 Environments" later in this chapter.

The classicMode and integratedMode preconditions match the configured *managedPipelineMode* attribute of each application pool. Together with the runtimeVersion preconditions, they provide a foundation for the ASP.NET versioning in IIS 7.0 and also allow for selecting the right set of ASP.NET handler mappings based on the integration mode of the application pool. In application pools that either use the Classic ASP.NET integration mode or use a .NET version that does not support direct integration, IIS 7.0 uses legacy ISAPI-based handler mappings for ASP.NET. Both of these sets of handler mappings are configured at the server level, and they use the classicMode/integratedMode and runtimeVersion preconditions to automatically select the right set of handler mappings based on the application pool's managed pipeline mode and Framework version.

You can use the applicationPoolName precondition to selectively load/enable modules and handler mappings in a particular application pool. An IIS 7.0 mechanism is provided to enable specific customer scenarios primarily on shared Web hosting servers. IIS 7.0 does not use it by default.

Finally, the managedHandler precondition enables modules to be enabled only for requests to ASP.NET handlers. For ASP.NET Integrated mode applications, IIS 7.0 enables managed modules to execute for all requests, whether or not they are mapped to ASP.NET handlers. However, by default, all ASP.NET modules use the managedHandler precondition to run only for requests to managed handlers. This also enables the ASP.NET appdomain creation to be delayed until the first request to an ASP.NET handler is made. This precondition can be removed from each module to allow it to run for all content types, regardless of whether they are managed or native. For example, to allow ASP.NET Forms-based authentication to occur for all content on the site, you need to remove the managedHandler precondition from the "FormsAuthentication" module. You can learn more about this in the "Enabling Managed Modules to Run for All Requests" section further in this chapter.

Preconditions solve a number of key problems in IIS 7.0. However, they can also add management complexity, and if configured incorrectly, they can result in unintended behavior. The largest cause of precondition-related problems is due to preconditions preventing modules from being loaded/enabled or handler mappings from being used, resulting in missing functionality. Though the module or handler mapping may appear present, its precondition can be preventing it from being active. These types of problems may be hard to diagnose because missing functionality does not always manifest in errors.

Another common problem is precondition inconsistency, where related configuration is not preconditioned correctly and results in configuration errors. For example, if a native module has a bitness32 load precondition, but the corresponding enablement entry in the modules list does not, requests to 64-bit application pools will produce a "bad module" error because the module being enabled is not loaded. Likewise, if a handler mapping refers to a module whose enablement precondition in the modules list prevents it from being enabled, requests that are mapped to this handler mapping will encounter an error.

To avoid these problems, remember that preconditions primarily serve to prevent a module/handler mapping from being used in scenarios where it cannot function. Make sure that the preconditions do not restrict the module from being available in scenarios where it's needed.

Also keep in mind the precondition relationships between the different configuration sections where they exist. Preconditions must get more restrictive as they go from module load preconditions, to module enablement preconditions, and finally to the handler mapping precondition for the module. For example, if the module load precondition in globalModules is "bitness32", the module enablement precondition for the corresponding entry in the modules section must at least contain that precondition. If the module is referenced in a handler mapping in the handlers configuration, the precondition of that entry must contain

at least the precondition strings from the modules entry (which in turn contains at least the preconditions from globalModules entry).

Installing Modules for x64 Environments

When IIS 7.0 is installed on 64-bit versions of the operating system, it functions in native 64-bit mode by default. This means that all application pools create native 64-bit worker processes and load 64-bit IIS core engine components and modules. However, by allowing any of its application pools to use the 32-bit emulation mode called wow64, IIS 7.0 supports hosting both native 64-bit and 32-bit applications. Unlike IIS 6.0, which also provided the ability to use wow64, IIS 7.0 allows each application pool to configure this individually, enabling side by side hosting of native 64-bit and 32-bit applications on the same server.

Each application pool that has the *enable32BitAppOnWin64* property set to true will create 32-bit worker processes and load the 32-bit version of the IIS core and modules. This is possible because IIS setup on 64-bit operating systems installs both 64-bit and 32-bit versions of all IIS components and modules—the native 64-bit versions go into the standard *%windir%*\System32\Inetsrv directory, and the 32-bit versions go into the *%windir%*\Syswow64\Inetsrv directory. At run time, when IIS tries to load modules located in the *%windir%*\System32\Inetsrv directory in a 32-bit wow64 worker process, the wow64 file system redirection feature automatically redirects the file access to the \Syswow64 directory where the 32-bit versions of the DLLs are located.

This mechanism enables IIS or third-party modules installed under the system32 directory to provide 32-bit versions under the \Syswow64 directory and then automatically load the correct version based on the "bitness" of the worker process.

However, the entire reason mixed 64-bit and 32-bit environments exist is that some functionality may not be available in native 64-bit flavors, requiring the worker process to operate in 32-bit mode. This is often needed for ASP applications that invoke in-process 32-bit COM components, 32-bit only ISAPI filters, or 32-bit only native modules. Likewise, some components may be available only in 64-bit flavors, and therefore they are not supported in 32-bit worker processes. To support such a scenario, you must be able to install native modules, ISAPI filters, and ISAPI extensions so that IIS never attempts to load a 32-bit component in a 64-bit worker process, and vice versa. IIS 7.0 provides this support via the bitness preconditions (see the section titled "Understanding Module Preconditions" earlier in this chapter), which enable native modules, ISAPI filters, and handler mappings to indicate whether they are available only in 32-bit or 64-bit application pools.

For example, handler mappings that map requests to the 32-bit version of the ASP.NET ISAPI use the bitness32 precondition to insure that they are used only inside 32-bit worker processes.

```
<handlers accessPolicy="Read, Script">

  <add name="PageHandlerFactory-ISAPI-2.0" path="*.aspx"
```

```
verb="GET,HEAD,POST,DEBUG" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi
dll" preCondition="classicMode,runtimeVersionv2.0,bitness32"
responseBufferLimit="0" />
  …
</handlers>
```

By default, the 64-bit version of the .NET Framework also registers an identical mapping to the 64-bit version of the aspnet_isapi.dll, which uses the bitness64 precondition so that it is selected only in 64-bit worker processes.

Using the bitness32 and bitness64 preconditions can therefore allow native modules, ISAPI filters, and ISAPI extensions specified in the handler mapping configuration to directly target 64-bit or 32-bit application pools, without using the file system redirection mechanism to provide both 32-bit and 64-bit flavors.

Common Module Management Tasks

Besides enabling you to choose which modules are installed on the server, IIS 7.0 enables you to further fine-tune its functionality by selecting which modules are enabled on the server or even for a particular application. Furthermore, you will sometimes want to tweak other aspects of module operation, such as their relative order, or the specific scenarios in which modules should execute. This section will illustrate some of these common module management tasks.

Controlling What Modules Are Enabled

Despite the differences between installation procedures for native and managed modules, the modules configuration section provides a unified view of the enabled modules. By manipulating the module entries in the modules section, you can control which modules will be allowed to function on the server by default or for a specific application:

- Adding a module at the server level allows it to execute by default in all applications on the server, except for applications that specifically remove it.
- Removing a module at the server level prevents it from executing in all applications on the server, except for applications that specifically add it back.
- Adding a module at the application level allows it to execute in that specific application.
- Removing a server-level defined module at the application level removes this module from the specific application, while allowing other applications to use it.

In a nutshell, modules that are enabled on the server level provide a default feature set for all applications on the server. Each application can then tweak this feature set by removing unneeded modules and then adding additional modules in its modules section. It is important to remember that though you can add new managed modules at the application level, new native modules must be installed at the server level to be enabled at the application

level. This means that applications cannot introduce new native modules—they can only remove existing ones that are enabled, or add back native modules that are installed but not enabled by default at the server level.

Note You can manage the enabled modules for your application by using the IIS Manager. After selecting your application in the tree view and opening the Modules feature, use the Add Managed Module action to add a new managed module, the Configure Native Modules action to enable or disable existing native modules, or the Edit or Remove actions to edit or remove existing module entries in the list. See the section titled "Using IIS Manager to Install and Manage Modules" later in this chapter for more information.

Note You can also use the Appcmd command line tool to manage the enabled modules. See the section titled "Using Appcmd to Install and Manage Modules" later in this chapter for more information.

Enabling Managed Modules to Run for All Requests

The ability to extend IIS with managed modules that execute for all content types is one of the central breakthroughs of IIS 7.0. However, for backward compatibility reasons, all of the built-in ASP.NET modules are configured to execute only for requests to managed (ASP.NET) handlers. Because of this, useful ASP.NET services such as Forms Authentication are by default available only for requests to ASP.NET content types, and they are not applied to requests to static content or ASP pages. The ASP.NET setup does this, adding the "managedHandler" precondition to each ASP.NET module element when it is added to the modules configuration section. See the section titled "Understanding Module Preconditions" earlier in this chapter for more information.

Because of this, it is necessary to remove this precondition from each ASP.NET module whose service is desired for all application content. This can be done by using Appcmd or IIS Manager to edit the specified modules element, or by manually removing the precondition from the module element. When this is desired at the application level for a module element inherited from the server level configuration, it is necessary to remove and redefine the module element without the precondition.

```
<modules>
  <remove name="FormsAuthentication" />
  <add name="FormsAuthentication"
type="System.Web.Security.FormsAuthenticationModule" />
</modules>
```

This clears the default "managedHandler" value of the *preCondition* attribute and enables the FormsAuthentication module to run for all requests.

When you use IIS Manager or Appcmd to edit the module element, this configuration is automatically generated whenever you make changes at the application level.

Note New managed modules you add will not have the managedHandler precondition by default and will run for all requests. If you want to restrict the managed module to run only for requests to managed handlers, you need to manually add the managedHandler precondition.

Alternatively, you can configure your application to ignore all managedHandler preconditions and effectively always execute all managed modules for all requests without needing to remove the precondition for each one. This is done by setting the runAllManagedModules-ForAllRequests configuration option in the modules configuration section.

```
<modules runAllManagedModulesForAllRequests="true" />
```

Controlling Module Ordering

Due to the pipeline model of module execution, module ordering is often important to ensure that the server "behaves" as it should. For example, modules that attempt to determine the authenticated user must execute before modules that verify access to the requested resource, because the latter needs to know what the authenticated user is. This ordering is almost always enforced by the stages of the request processing pipeline. By doing their work during the right stage, modules automatically avoid ordering problems. However, in some cases, two or more modules that perform a similar task—and therefore execute in the same stage—may have ordering dependencies. One prominent example is built-in authentication modules. They are run during the AuthenticateRequest stage, and to authenticate the request with the strongest credentials available, they should be in the strongest to weakest order. To resolve such relative ordering dependencies, the administrator can control the relative ordering of modules by changing the order in which they are listed in the modules section.

This works because the server uses the order in the modules configuration section to order module execution within each request processing stage. By placing module A before module B in the list, you can allow module A to execute before module B.

This also means that when an application enables a new module (by adding a new managed module, or enabling a native module that was not previously enabled), that module is listed after the modules enabled by higher configuration levels due to the configuration collection inheritance. This can sometimes be a problem if the new module should run before an existing module defined at the higher level, because the configuration system does not provide a way to reorder inherited elements. In this case, the only solution is to clear the modules collection and re-add all of the elements in the correct order at the application level.

```
<modules>
    <clear/>
    <add name="HttpCacheModule" />
```

```
        …
        <add name="MyNewModule" type="Modules.MyNewModule" />
        …
    <modules>
```

Note You can also use IIS Manager to perform the ordering task. After selecting your application in the tree view and opening the Modules feature, choose the View Ordered List action and use the Move Up and Move Down actions to adjust the sequence. If you use this feature, the tool will use the <clear/> approach that we discussed earlier to reorder the modules for your application.

Caution By using the <clear/> approach, you are effectively disconnecting the application's module configuration from the configuration at the server level. Therefore, any changes made at the server level (removing or adding modules) will no longer affect the application and will need to be manually propagated if necessary.

Adding Handler Mappings

Though modules typically execute for all requests so that the modules can provide a content-independent service, some modules may opt to act as handlers. Handlers are responsible for producing a response for a specific content type and are mapped in the IIS 7.0 handler mapping configuration to a specific verb/extension combination. For handlers, the server is responsible for mapping the correct handler based on the handler mapping configuration, and they are also responsible for invoking that handler during the ExecuteRequest request processing stage to produce the response for this request. Examples of handlers include StaticFileModule, which serves static files; DirectoryListingModule, which displays directory listings; and the ASP.NET PageHandler, which compiles and executes ASP.NET pages.

The main conceptual difference between modules and handlers is that the server picks the handler to produce the response for requests to a specific resource, whereas modules typically process all requests in a resource-independent way and typically do not produce responses. Because of this, only the one handler mapped by the server is executed per request. If you are familiar with IIS 6.0, this is similar to the distinction between the ISAPI extensions, which provide processing for a specific extension, and ISAPI filters, which intercept all requests.

Traditionally, most application frameworks including ASP.NET, ASP, PHP, and ColdFusion are implemented as handlers that process URLs with specific extensions.

You register a handler on the server by creating a handler mapping entry in the collection located in the system.webServer/handlers configuration section. This concept is similar to the script maps configuration in previous releases of IIS, but in IIS 7.0 it is extended to allow for more flexibility and to accommodate more handler types. For applications using the Integrated mode, this section also supports managed handlers that in previous IIS versions are registered in the ASP.NET httpHandlers configuration section.

After it receives the request, the server examines the collection of handler mappings configured for the request URL and selects the first handler mapping whose path mask and verb match the request. Later, during the ExecuteRequestHandler stage, the handler mapping will be used to invoke a module to handle the request.

Each handler mapping collection entry can specify the attributes shown in Table 12-5.

Table 12-5 Attributes Specified by Handler Mappings

Attribute	Description
name (required)	The name for the handler mapping.
path (required)	The path mask that must match the request URL so that this handler mapping can be selected.
verb (required)	The verb list that must match the request verb so that this handler mapping can be selected.
resourceType	Whether the physical resource mapped to the request URL must be an existing file, directory, either, or unspecified (if the physical resource does not have to exist).
requireAccess	The accessFlag level that is required for this handler to execute.
precondition	The precondition that determines if this handler mapping is considered.
allowPathInfo	Whether or not the PATH_INFO / PATH_TRANSLATED server variables contain the path info segment; may cause security vulnerabilities in some CGI programs or ISAPI extensions that handle path info incorrectly.
responseBufferLimit	The maximum number of bytes of the response to buffer for this handler mapping. Response buffering is new in IIS 7.0 and enables modules to manipulate response data before it is sent to the client. The default is 4 MB, although ISAPI extensions installed with legacy APIs will have it automatically set to 0 for backward compatibility reasons.
Modules	List of modules that attempt to handle the request when this mapping is selected.
scriptProcessor	Additional information that is passed to the module to specify how the handler mapping should behave. Used by ISAPI extension module, CGI module, and FastCGI module.
type	The managed handler type that handles the request when this mapping is selected.

The information in the handler mapping is used as follows.

1. The *precondition* is first used to determine if the handler mapping is to be used in a particular application pool. If any of the preconditions fail, the mapping is ignored.
2. The *path* and *verb* are matched against the request URL and verb. The first mapping that matches is chosen. If no mappings matched, a "404.4 Not Found" error is generated.
3. If the accessPolicy configuration does not meet the *requireAccess* requirement for the handler mapping, a "403 Access Denied" error is generated.

4. If the *resourceType* is set to File, Directory, or Either, the server makes sure that the physical resource exists and is of the specified type. If not, a "404 Not Found" error is generated. Also, check that the authenticated user is allowed to access the mapped file system resource. If resourceType is set to Unspecified, these checks are not performed.

Note The path attribute in IIS 7.0 enables you to specify more complex path masks to match the request URL than previous versions of IIS, which enable only * or *.ext* where ext is the URL extension. IIS 7.0 enables you to use a path mask that may contain multiple URL segments separated by / and to use wildcard characters such as * or *?*.

Even though the majority of IIS 7.0 handlers are added at the server level and inherited by all applications on the server, you can specify additional handlers at any level. Handler mappings added at a lower level are processed first when matching handler mappings, so new handlers may override handlers previously declared at a higher configuration level. Because of this, if you want to remap that path/verb pair to another handler for your application, it is not necessary to remove a handler added at a server level—simply adding that handler mapping in your application's configuration does the job.

Note IIS 7.0 continues to support wildcard mappings, which enable a handler to act like a filter, processing all requests and possibly delegating request processing to another handler by making a child request. Though the majority of such scenarios can now be implemented with normal modules, quite a few legacy ISAPI extensions take advantage of this model (including ASP.NET in some configurations). To create a wildcard mapping, you need to set the path and verb attributes to *, set the *requireAccess* attribute to *None*, and set the *resourceType* attribute to *Either*.

Types of Handler Mappings

Though it provides a standard way to map handlers to requests, the handlers configuration also supports a number of different types of handlers, as shown in Table 12-6.

Table 12-6 Handler Types

Handler Type	Configuration	IIS 7.0 Examples
Native module(s) The module must support the *ExecuteRequestHandler* event	*modules* specifies the list of native modules that will handle this request (typically just specifies one module)	TraceVerbHandler, OptionsVerbHandler, StaticFileModule, DefaultDocumentModule, DirectoryBrowsingModule
ASP.NET handler The application must be using the Integrated ASP.NET mode	*type* specifies fully qualified .NET type that implements ASP.NET handler interfaces	ASP.NET PageHandlerFactory (aspx pages), ASP.NET WebResourceHandler

Table 12-6 Handler Types

Handler Type	Configuration	IIS 7.0 Examples
ISAPI extension	*modules* specifies the ISAPIModule; *scriptProcessor* specifies the path to the ISAPI extension DLL to load	ASP.dll (asp pages)
CGI program	*modules* specifies the CGIModule; *scriptProcessor* specifies the path to the CGI executable	Any CGI executable
FastCGI program	*modules* specifies the *FastCGIModule*; *scriptProcessor* specifies the path and arguments for a FastCGI executable registered in the FastCGI configuration section	Any FastCGI executable (such as PHP-CGI.EXE)

Unlike script maps in previous versions of IIS, which provide hardcoded support for ISAPI extensions and CGI programs, IIS 7.0 hardcodes nothing—all types of handlers are implemented on top of the standard native or managed module API. IIS 7.0 supports ISAPI extensions by hosting them with the ISAPIModule, supports CGI programs with the CGI module, and features new support for FastCGI programs with FastCgiModule. The IsapiModule, CgiModule, and FastCgiModule modules are all native modules, much like StaticFileModule, except they support interfacing with external handler frameworks to handle the request, using the ISAPI, CGI, and FastCGI protocols respectively.

If you look at the handler mappings created by default by a full IIS 7.0 install, you will see some of the following.

```
            <handlers accessPolicy="Read, Script">
                <add name="ASPClassic" path="*.asp" verb="GET,HEAD,POST"
modules="IsapiModule" scriptProcessor="%windir%\system32\inetsrv\asp.dll"
esourceType="File" />
                <add name="ISAPI-dll" path="*.dll" verb="*"
modules="IsapiModule" resourceType="File" requireAccess="Execute"
allowPathInfo="true" />
                ...
                <add name="PageHandlerFactory-Integrated" path="*.aspx"
verb="GET,HEAD,POST,DEBUG" type="System.Web.UI.PageHandlerFactory"
preCondition="integratedMode" />
                ...
                <add name="PageHandlerFactory-ISAPI-2.0" path="*.aspx"
verb="GET,HEAD,POST,DEBUG" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi
dll" preCondition="classicMode,runtimeVersionv2.0,bitness32"
responseBufferLimit="0" />
                ...
                <add name="StaticFile" path="*" verb="*"
modules="StaticFileModule,DefaultDocumentModule,DirectoryListingModule"
resourceType="Either" requireAccess="Read" />
            </handlers>
```

This configuration fragment shows a good cross-section of the kinds of handler mappings that you can create. First is IsapiModule handler mapping, which enables ASP pages to be executed with the ASP.dll ISAPI extension. Second is the IsapiModule mapping, which supports direct requests to ISAPI extensions located in the application directories, which require the Execute permission.

Then, you see two mappings for the ASP.NET PageHandlerFactory, which supports the processing of ASPX pages. The first mapping uses the aspnet_isapi.dll ISAPI extension to process the request, and the second uses the Integrated mode for executing ASP.NET handlers directly. Each of these mappings uses a precondition to make sure that only one of the mappings is active in each application pool based on the ASP.NET integration mode. Classic mode application pools use the ISAPI mapping, and Integrated mode application pools use the integrated mapping. You can read more about ASP.NET integration and ASP.NET handler mappings in Chapter 11. Finally, you see the static file handler mapping, designed to be a catch-all mapping that is mapped to all requests that do not match any of the other handler mappings by specifying "*" for both path and verb. This is similar to previous versions of IIS where any requests not mapped to an ISAPI extension scriptmap are handled by the static file handler in IIS. This mapping also illustrates letting multiple modules attempt to handle the request as part of a single handler mapping. First, StaticFileModule attempts to serve a physical file if one is present, then DefaultDocumentModule performs the default document redirect, and finally DirectoryBrowsingModule attempts to serve a directory listing.

Security Alert The fact that the catch-all mapping uses StaticFileModule means that requests to resources that have not yet had a handler configured *but* are not listed in the server's MIME type configuration will result in a "404.3 Not Found" error. This error typically indicates that either you need to add a MIME map entry for the file's extension to the server's staticContent configuration section to allow the file to be downloaded, or you need to add a handler mapping to appropriately process the file. This is an important security measure that prevents scripts from being downloaded as source code on servers that do not yet have the right handler mappings installed. For more information on adding MIME type entries, see Chapter 11.

You will find out more about using IIS Manager and Appcmd to create handler mappings in the sections titled "Using IIS Manager to Install and Manage Modules" below and "Using Appcmd to Install and Manage Modules" later in this chapter.

Using IIS Manager to Install and Manage Modules

IIS Manager provides a powerful UI for managing modules on the server. This UI can be used to install both native and managed modules, as well as manage enabled modules on the server and for specific applications.

The Modules feature provides this functionality, and it can be accessed at two separate levels for slightly different functionality:

- By server administrators at the server level, to install native modules on the server, add new managed modules, and configure modules that are enabled on the server by default.
- By server or site administrators at the application level, to add new managed modules and configure enabled modules for the application.

At the server level, you can select the machine node in the tree view and then double-click the modules to access the Modules feature, as shown in Figure 12-3.

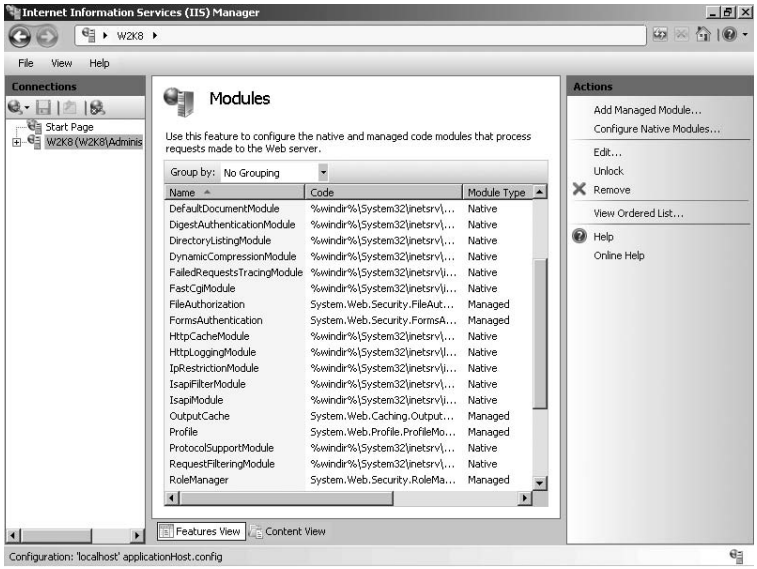
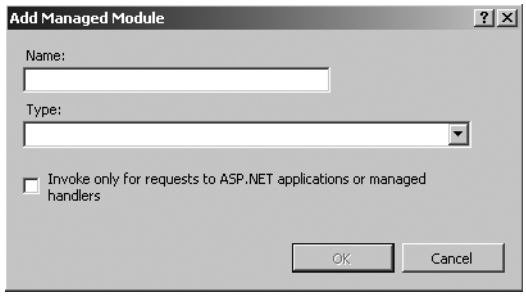


Figure 12-3 The Modules feature in IIS Manager.

You will see the list of modules enabled at the server level, which corresponds to the list of modules in the modules configuration section at the server level in ApplicationHost.config. For each module, you will see its name. For native modules, you'll also see the path to the image DLL. For managed modules, you'll also see the module type name. You will also see three actions available on this page:

- **Add Managed Module** Enables you to add a new managed module to the list of enabled modules. In the resulting dialog box, you can specify the name of your new module, as well as the module's type.



You can select a module type from the Type drop-down list, which contains all module types available from the assemblies located in the machine's Global Assembly Cache listed in the system.web/compilation/assemblies section of the .NET Framework's root Web.config. You can also type in your own type if it's not yet listed. Select the Invoke Only For Requests To ASP.NET Applications Or Managed Handlers check box if you want your module to use the managedHandler precondition and only execute for requests to ASP.NET handlers (see the section titled "Understanding Module Preconditions" for more information about this).

- **Configure Native Modules** Enables you to enable an already installed native module, install a new native module, or uninstall an existing native module.

Here, you can enable native modules that are installed but not currently enabled by selecting one or more of them. You can also use the Register button to install a new native module, the Edit button to edit the name or the path for an installed module, or the Remove button to uninstall the selected native module.

- **View Ordered List** Enables you to display the list of modules in an ordered list so that you can adjust their sequence by using the Move Up and Move Down actions. When this is done at the server level, you can reorder the modules without resorting to clearing the modules collection (see the section titled "Controlling Module Ordering" earlier in this chapter for more information).

Also, when you select a module entry in the list, three additional actions become available:

1. **Edit.** This action enables you to edit the modules entry. For native modules, you can directly change the native module installation information including its name and path to native image DLL. For managed modules, you can edit the module name and the module type.
2. **Lock/Unlock.** These actions enable you to lock the specific module item at the server level, such that it cannot be removed or modified at the application level. See the section titled "Locking Down Extensibility" later in this chapter for more information about locking modules.
3. **Remove.** This action enables you to remove the module entry. For native modules, this disables the module by default. For managed modules, this removes the module entry, requiring you to later re-add this entry at the application level to enable it there.

To access the Modules feature at the application level, go to the tree view and select the application you would like to administer. Then double-click the Modules feature icon. You will be presented with the same view as before, except for the following differences:

- You will no longer be able to add new native modules from the Configure Native Modules dialog box. Remember, this is because native modules are installed for the entire server, and you must have Administrative privileges to do that. Instead, you will only be able to enable already installed native modules that are not currently enabled for your application.
- You will no longer be able to edit native module information or edit managed module information for managed modules that are enabled at the server level (you can still edit module information for managed modules added in your application).
- You will not be able to lock/unlock modules.
- When adding managed modules, the tool will also inspect all assemblies in the /BIN and /App_Code source files for possible modules to add.
- You can use the Revert To Inherited action to discard whatever changes were made to the modules configuration section at the application level and then revert to the default module configuration at the server level.

Despite these limitations, site administrators can still use IIS Manager to install new managed modules or manage their applications' module feature set without requiring Administrative privileges on the machine. This is especially valuable given the ability of IIS Manager to enable remote delegated administration for application owners. Of course, server administrators can also benefit from IIS Manager for unrestricted module management.

Using IIS Manager to Create and Manage Handler Mappings

IIS Manager also provides a convenient interface to manage handler mappings, thus removing some of the complexity involved with editing the handler mappings manually. This functionality is provided by the Handler Mappings feature, which both server administrators and site administrators can access.

After selecting the node at which you'd like to manage handler mappings, you can select the feature by double-clicking the Handler Mappings icon. This presents the Handler Mappings view, as shown in Figure 12-4.

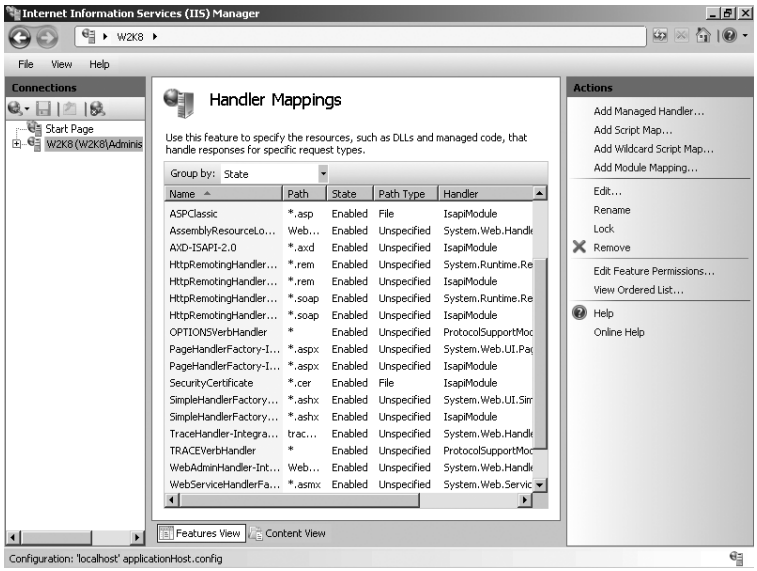


Figure 12-4 The Handler Mappings feature in IIS Manager.

Here, you will see a list of handler mappings including the handler mapping name, path, and other useful information. The Handler column provides a summary of how the handler is implemented, showing either the modules list for native handlers or the type for managed handlers. The Path Type indicates the resourceType of the handler mapping. The State column indicates if this handler is enabled (this applies only to handler mappings using IsapiModule or CgiModule) and indicates whether the ISAPI extension or CGI program specified by the mapping is enabled in the system.webServer/security/isapiCgiRestrictions configuration (analogous to the Web Service Restriction List in IIS 6.0).

The tool provides a number of ways to add new handler mappings, breaking them out by type to simplify handler mapping creation:

- **Add Managed Handler** This action enables you to create handler mapping to an ASP.NET handler. This mapping is available only in application pools that are using Integrated mode. It is shown in Figure 12-5.

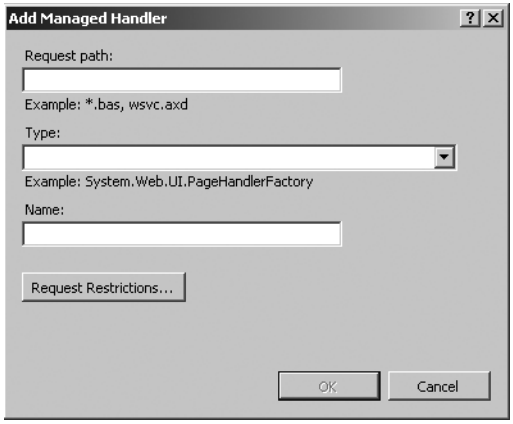


Figure 12-5 Add Managed Handler dialog box.

In this dialog box, you specify the path mask for the handler mapping, as well as the ASP.NET handler type that should provide processing for it. Much like when adding managed modules, the tool searches for usable types in the assemblies from the Global Assembly Cache (GAC) that are referenced in the ASP.NET compilation/assemblies collection. It also searches for application assemblies when you're adding the handler at the application level. You can use the Request Restrictions dialog box to also specify the verb list for the handler mapping (otherwise, it defaults to "*"), restrict the mapping to physical resources such as a file or directory (the default is unspecified), and set access level as required to execute the mapping (defaults to Script). You should indeed set these to the strictest possible levels for added security instead of leaving them at default values.

- **Add Script Map** This action enables you to create an ISAPI extension or CGI program handler mapping, similar to the IIS 6.0 scriptmap. This is shown in Figure 12-6.

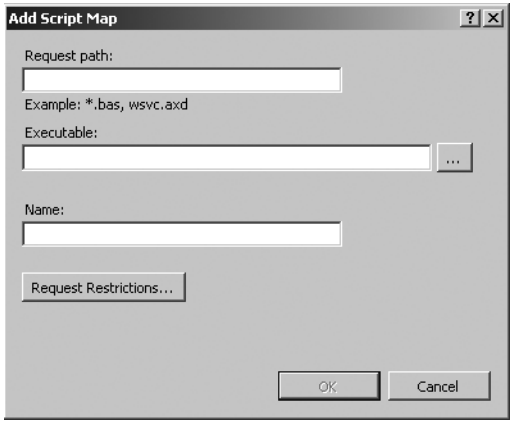


Figure 12-6 Add Script Map dialog box.

This is similar to the previous dialog box, except instead of the .NET handler type, you select the ISAPI extension, DLL, or CGI program executable on the server's local file system. This dialog box also prompts you to automatically create the isapiCgiRestriction entry to allow the ISAPI extension or CGI program to execute on the server.

- **Add Wildcard Script Map** This is identical to the Add Script Map dialog box, except that it enables you to create a wildcard script map that intercepts all requests.
- **Add Module Mapping** This is the most interesting action, because it enables you to create a general handler mapping that specifies one or more native modules and optionally a script processor. You can use this dialog box to create ISAPI extension, CGI, and FastCGI handler mappings in addition to simple handler mappings for native modules, as shown in Figure 12-7.

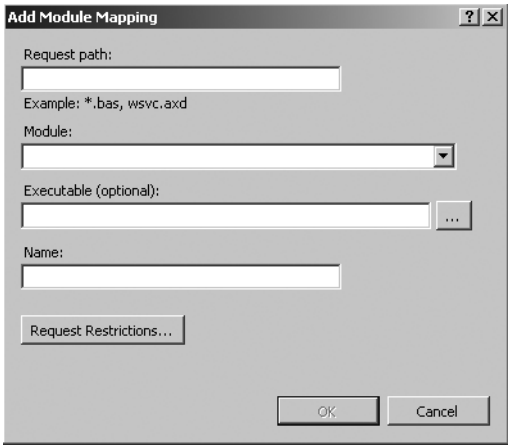


Figure 12-7 Add Module Mapping dialog box.

The dialog box enables you to specify the Module (and it displays a list of available native modules, although not all of them can function as handlers) and the Executable (the script processor). It also provides special handling for IsapiModule, CgiModule, and FastCgiModule handler mappings by prompting you to automatically create the corresponding isapiCgiRestrictions or FastCGI configuration needed for the mapping to work correctly.

Note It is not possible to edit preconditions via the Handler Mappings feature. Instead, the tool automatically generates the correct preconditions for handler mappings to make it function correctly by default. If you would like to change the preconditions, you need to edit configuration directly by hand, with Appcmd, or with other configuration APIs.

In addition to adding new handler mappings, you can also edit or delete existing handler mappings by clicking the item in the list and using the Edit or Delete commands that become available. At the server level, you can also lock handler mappings to prevent them from being

removed at the lower level, although this has less effect than locking modules because handler mappings can be overridden by adding new handler mappings for the same path/verb.

You can also use the "View Ordered List" action to order handler mappings. Keep in mind that just as it is for modules, ordering inherited elements requires the tool to use <clear/> to clear the handlers collection and add again all parent items into the configuration level being edited, essentially orphaning the configuration from the parent handler mappings configuration.

If you edit handler mappings as a site administrator, you have virtually the same functionality available to you for managing the handler mappings for your site, application, or any URL inside your site. However, the tool will not prompt you to enable ISAPI extensions, CGI programs, and FastCGI programs that the administrator has not already enabled at the server level.

Using Appcmd to Install and Manage Modules

The Appcmd command line tool also provides top-level support for managing modules. To begin with, you can always edit the IIS configuration via the Appcmd Config object to perform all the tasks necessary to install and manage modules. However, the tool also provides a module object, which directly supports common module management tasks. This is why Windows Setup calls into Appcmd to install modules and is the reason Appcmd is often the quickest way to install and manage modules. Appcmd is also the only tool available for managing modules on Windows Server 2008 Server Core installations, which do not support IIS Manager.

Appcmd is located in the *%windir%*\System32\Inetsrv directory, which is not present in the PATH variable by default, so you need to use the full path to the tool to run it.

```
%windir%\system32\inetsrv\AppCmd
```

You must be logged in as an administrator when using Appcmd. Also, be sure to execute Appcmd commands from an elevated command line prompt (click Start, right-click Command Prompt, and choose Run As Administrator). You can refer to Chapter 7, "Using Command Line Tools," for more information about the tool. If you get lost while learning Appcmd, be sure to check out the built-in command line help, which provides parameters and examples.

```
AppCmd module /?           - See all commands on the module object
AppCmd install module /?   - See the usage help for the install module command
```

In the help output for the MODULE object, you can see that the tool supports the following commands:

- **List** Enables you to examine enabled modules at the server level or for a specific application.
- **Install** Enables you to install new native modules on the server.

- **Uninstall** Enables you to uninstall currently installed native modules on the server.
- **Add** Enables you to add a new module. You can also opt to enable an installed native module for a specific application.
- **Delete** Enables you to disable a managed module You can also opt to disable a native module, optionally for a specific application.
- **Set** Enables you to edit specific entries in the modules configuration section for the purposes of changing the name or preconditions of a module or editing type information for managed modules.

In the next section, we'll provide some examples of using these commands to manage modules with Appcmd.

Installing and Uninstalling Modules

The Install and Uninstall module commands provide support for—you guessed it—installing and uninstalling native modules on the server.

To install native modules, you can use the following syntax.

```
AppCmd Install Module /name:string /image:string [/precondition:string]
[/add:bool] [/lock:bool]
```

This command accepts the parameters shown in Table 12-7.

Table 12-7 Appcmd Install Module Parameters

Parameter	Description
name (required)	Module name.
image (required)	The path to the module DLL.
preCondition	Optionally specifies the list of valid load preconditions for the module. If specified, controls whether the module is loaded by the IIS worker processes in particular application pools. The default is empty.
add	Optionally specifies whether to also enable the module at the server level. The default is TRUE.
lock	Optionally specifies whether to also lock the module entry so that it cannot be disabled by applications. This only applies if the module is being enabled. The default is FALSE.

The simplest case is installing a module by simply specifying its name and image path.

```
AppCmd install module /name:MyNativeModule /image:c:\modules\mymodule.dll
```

This installs and automatically enables the new module at the server level so that it's loaded by all IIS worker processes and enabled for all applications by default. If you wanted to install the module but enable it selectively for specific applications only, you would include the /add:false parameter. Then, you could use the Appcmd *Add Module* command later to enable it for a specific application by using the /app.name parameter. If, on the other hand, you

wanted to install and enable this module and prevent applications from disabling it, you would include the /lock:true parameter for the install command. You could also set the precondition parameter to one of the supported values (see the section titled "Understanding Module Preconditions" earlier in this chapter) to control which application pools the module is loaded and available in.

To uninstall a module, you can use the following syntax.

```
AppCmd Uninstall Module ModuleName
```

This command accepts the module's name as the identifier.

Tip The uninstall command uses the standard Appcmd identifier pattern for specifying the module name, as opposed to using the /name parameter that the Install command uses. This is done so that all commands that work on specific existing instances of objects can use the identifier format instead of using different parameter names to identify objects.

Here is an example of how you can uninstall the module you installed earlier.

```
AppCmd uninstall module MyNativeModule
```

Note This also automatically disables the module at the server level, which makes sense because allowing this module to be enabled would result in an incorrect configuration after the module is uninstalled. However, if for some reason you wanted to uninstall a module but leave it enabled (this would cause request errors on the entire server if left unchanged), you can specify the /remove:false parameter.

Enabling and Disabling Modules

You can use the *Add Module* and *Delete Module* commands to manage the modules enabled on the server or for a particular application. You can also use the *Add Module* command to add new managed modules. These commands manipulate the modules configuration section, which contains the entries for enabled native modules and defines managed modules. Therefore, you can perform a number of different tasks around module management.

One of the most common uses for the *Add Module* command is to add new managed modules. Because native modules have to be installed on the server first before they are enabled, the *Install Module* command is more appropriate for installing and automatically enabling them. You can add a new managed module at the server level or for any particular application using the following syntax.

```
AppCmd Add Module /name:string [/type:string]  [/precondition:string]
[/app.name:string] [/lockItem:bool]
```

This command supports the parameters shown in Table 12-8.

Table 12-8 Appcmd Add Module Parameters

name (required)	Module name.
type (required)	The fully qualified .NET type of the module. If the module is being added at the server level, the type must be in an assembly registered with the machine's Global Assembly Cache. If it's being added for a specific application, then it can also be deployed with the application. Refer to the section titled "Deploying Assemblies Containing Managed Modules" for more information.
preCondition	Optionally specifies the list of valid enablement preconditions for the module. If specified, controls whether the module is enabled in specific application pools and for specific requests. The default is empty.
app.name	Optionally specifies the application path for the modules to be added to. The default is empty, meaning the server level.
lockItem	Optionally specifies whether or not to lock the module entry against removal at lower configuration levels.

For example, to add a new managed module at the server level, you can do the following.

```
AppCmd add module /name:MyManagedModule /type:MyModules.MyModule
```

This makes this module enabled by default for all applications on the server. If you wanted to add it only to the root application in the default Web site, you would also add the /app.name:"Default Web Site/" parameter. You could also set the preCondition parameter to one of the supported values (see the section titled "Understanding Module Preconditions" earlier in this chapter) to control for which application pools and requests the module is enabled.

You can also use the lockItem parameter just like you can when creating new configuration collection entries to lock the module entry, which prevents lower configuration levels from removing the module configuration entry. You can leverage this when adding new managed modules at the server level to prevent them from being disabled at the application level. This is discussed more in the section titled "Locking Down Extensibility" later in this chapter.

Another common use of the *Add Module* command is to enable a native module that is not currently enabled. For example, if you installed a native module with the /add:false parameter, resulting in it being installed but not enabled by default, you can directly enable it.

```
AppCmd add module /name:MyNativeModule
```

You can use the /app.name parameter here to specify the application where the module should be enabled. This only works for native modules; that is, when re-enabling managed modules for a specific application, you always have to specify the type because this information is not available elsewhere like it is for native modules.

You can use the *Delete Module* command to do the opposite—to disable a module that is currently enabled. You can also use this command to disable native module or managed modules at the server level, or for a specific application, using the following syntax.

```
AppCmd Delete Module ModuleName [/app.name:string]
```

Tip The delete command uses the standard Appcmd identifier pattern for specifying the module name, as opposed to using the /name parameter that the add command uses. This syntax exists so that all commands that work on specific existing instances of objects can use the identifier format instead of using different parameter names to identify objects.

For example, to disable a module that is enabled at the server level, you can do the following.

```
AppCmd delete module MyModule
```

This works for both managed and native modules, with a slight caveat: if you disable a native module, you can re-enable it simply by using the *Add Module /name:ModuleName* command. However, if you disable a managed module, you will need to specify its full type to re-enable it. If you delete a managed module at the level where it's defined for the first time, you may lose the type information in case you need to re-add it later.

Examining Enabled Modules

In addition to installing/uninstalling and enabling/disabling modules, Appcmd supports examining the enabled modules with the LIST command. This can prove valuable when diagnosing module-related issues by enabling you to quickly determine which modules are enabled for a specific application, or if a specific module is enabled.

The *List Module* command, much like the LIST commands for other Appcmd objects, enables you to display all modules as well as query for modules that satisfy a specific query. In the simplest use, it enables you to quickly list the modules that are enabled at the server level.

```
AppCmd list modules
```

To see which modules are enabled for a specific application, use the /app.name:*AppPath* parameter to specify the application path for which the enabled modules should be displayed.

Note Because applications can remove certain modules or add new ones, their module set can often be different from that of the server itself. Be sure to check the module list for the application you're interested in when you're investigating problems with that application.

You can also specify queries for one or more of the module attributes, including *precondition* and *type*, to find modules that have that attribute. For example, to find all managed modules that have the managedHandler precondition set, you can use the following code.

```
AppCmd list modules "/type:$<>" "/precondition:$=*managedHandler*"
```

You can also look for specific modules by using the module name as the identifier.

```
AppCmd list module DefaultDocumentModule
```

Creating and Managing Handler Mappings

Though Appcmd provides a top-level Module object for managing modules, it does not provide a top-level view of the handler mappings configuration. However, you can always use the *CONFIG* object to directly manage the system.webServer/handlers configuration section to accomplish any needed handler mapping management task.

Discussing the full collection editing syntax of the CONFIG object is out of scope for this section (see Chapter 7), but here are some examples for using it to do basic handler mapping management tasks.

Adding a Handler Mapping

To add a handler mapping to a native module, run the following command on one line.

```
AppCmd set config /section:handlers "/+[name='TestHandler',path='*test',
verb='GET,POST,HEAD',modules='TestModule']"
```

Note Long commands are sometimes shown formatted on multiple lines to fit on the printed page.

This adds a server-level handler mapping that maps GET, POST, and HEAD requests to *.test URLs to the "TestModule" native module. Though only the name, path, and verb attributes are required (and the modules attribute should be set for the native module handler mapping), you can also specify any of the other attributes that apply. You can also specify the configuration path at which this mapping should be added instead of adding it at the server level, as shown in the next example.

To add a handler mapping to a managed handler type, run the following command on one line.

```
AppCmd set config "Default Web Site/" /section:handlers
"/+[name='ASPNHandler',path='*.aspn',verb='*',
type='MyHandlers.ASPNHandler',precondition='integratedMode']"
```

This adds a handler mapping for the root of the "Default Web Site" Web site, which maps all requests to *.aspn URLs to the .NET MyHandlers.ASPNHandler handler type. Notice that

the handler mapping is also preconditioned to be used only in application pools running in Integrated mode. This is required for handler mappings to managed types, because only Integrated mode supports adding managed handler types directly into IIS handler mappings.

Editing a Handler Mapping

To edit a handler mapping, use the following code.

```
AppCmd set config /section:handlers /[name='TestHandler'].verb:GET
```

This sets the verb attribute of the handler mapping identified by the name TestHandler to the new value of GET. Note that the name attribute serves as the unique collection key for the handlers configuration section.

You can use this syntax to edit any of the handler mapping attributes. You can also edit the handler mapping you created at the Default Web Site/ level by specifying that path after the *SET CONFIG* command.

Deleting a Handler Mapping

To delete a handler mapping, you can use the /- prefix for deleting configuration collection elements.

```
AppCmd set config /section:handlers /-[name='TestHandler']
```

This deletes the handler mapping you created earlier, identified by the name *TestHandler*. You can also delete the handler mapping you created at the "Default Web Site/" level by specifying that path after the *SET CONFIG* command.

Adding Entries to the ISAPI CGI Restriction List (Formerly Web Service Restriction List)

When creating handler mappings that use CgiModule or IsapiModule to support CGI programs or ISAPI extensions respectively, you also need to allow the specific CGI program or ISAPI extension by adding it to the ISAPI CGI Restriction List. In IIS 6.0, this is known as the Web Service Restriction List and is a key security measure to control the execution of third-party code on the server.

For example, to add an ISAPI extension DLL to the system.webServer/security/isapiCgi-Restriction list, use the following command.

```
appcmd set config /section:isapiCgiRestriction /+[path='c:\myisapi.dll',
allowed='true']
```

To allow (or disallow) an existing entry in the list, use the following command.

```
appcmd set config /section:isapiCgiRestriction
/[path='c:\myisapi.dll'].allowed:true
```

To delete an entry in the list, use the following command.

```
appcmd set config /section:isapiCgiRestriction /-[path='c:\myisapi.dll']
```

You can specify both CGI programs (executables) and ISAPI extensions (DLLs) in this list.

Note FastCGI program executables are not added to the isapiCgiRestriction list. Instead, they must be registered in the system.webServer/fastCGI configuration section to allow FastCGI application pools to be created for this executable.

Securing Web Server Modules

The extensibility architecture in IIS 7.0 is in many ways the recognition of the fact that it's not really the server but rather the application that runs on it that makes all the difference. Unfortunately, history shows that it is also the application that is most commonly the cause of security vulnerabilities. The lockdown approach to security that IIS 6.0 provides—restricting the ability to run new code on the server and reducing the privilege of that code—has been immensely successful in reducing Web server exploits. Now, with IIS 7.0, server administrators must strike a balance between the functionality afforded by the new extensibility model and server security. Therefore, it is now more important than ever to understand the security impact of hosting server extensibility and to know how to properly lock it down to avoid weakening the security of your server.

When it comes to securing the server, one of the biggest problems administrators face today is dealing with system complexity and being able to properly apply key security best practices rather than being bogged down in the details. This approach, though not a replacement for proper security threat modeling and penetration testing at the application level, enables you to significantly reduce the general security risk to the server. Basically, you need to be able to answer the following question: Assuming you cannot trust the code running on your server to be completely foolproof, how can you control what it can and cannot do, and at the same time prevent your server from being compromised if this code is exploited?

The best answer to this question is to approach it in the following stages. First, you need to know how to reduce the server's surface area to a minimum, removing all functionality that is not essential to its operation. Second, you need to understand the privilege of code that executes on the server and then reduce it as much as possible. Finally, you need to maintain control over the extensibility that is allowed on the server, preventing undesired functionality from being added or desired functionality from being removed. You should also apply the best practices listed in Chapter 14, "Implementing Security Strategies," to secure individual features.

Taking Advantage of Componentization to Reduce the Security Surface Area of the Server

To completely secure a Web server, you would need to disconnect it from the network, unplug it, and bury it in a thick slab of concrete. This would guarantee its security by removing the

possibility of any malicious interactions with the system. However, because this will also make the Web server useless, you have to find other ways to apply the security principle of reducing the attack surface area.

Direct from the Source: The Most Secure Web Server in the World

My first demo for the IIS 7.0 breakout session at the conference TechEd 2005 was to showcase the componentization capabilities of IIS 7.0 by showing off the "most secure Web server in the world."

As part of the demo, I showed editing the configuration in the ApplicationHost.config file, removing all of the modules, and removing handler mappings. After saving the file, the IIS worker process automatically picked up the changes and restarted, loading absolutely no modules. After making a request to the default Web site, I got back a swift empty 200 response (this configuration currently returns a "401 Unauthorized" error because no authentication modules are present). The server performed virtually no processing of the request and returned no content, thus becoming the "most secure Web server in the world." After a pause, I commented that though secure, the server was also fairly useless. Then I segued into adding back the functionality that we needed for our specific application.

I have done this demo before for internal audiences to much acclaim, but I will always remember the audience reaction during the TechEd presentation. The people in the audience went wild, some breaking out into a standing ovation. This was a resounding confirmation of our efforts to give administrators the ability to start from nothing, building up the server with an absolutely minimal set of features to produce the most simple-to-manage, low-surface-area Web server possible.

Mike Volodarsky

Program Manager, IIS 7.0

The ability of IIS 7.0 to remove virtually all features of the Web server is fundamental here, enabling us to eliminate the threat of any known or unknown attack vectors that may exist in those features. In addition, removing extra functionality reduces management complexity and reduces the chance of your server being forced offline if a patch affects the removed features. You can leverage this ability by doing the following:

1. Determine the set of features that you need for your server/application.
2. Install *only* the required IIS 7.0 features by using Windows Setup.
3. Manually remove any modules that your application does not need.

At the end of step 2, you should have the minimum required set of functionality installed globally for your server. You can then further reduce the surface area by disabling the modules

you do not need in each of your applications, in the case where each application requires slightly different server functionality.

In some other cases, you may need to disable modules for the entire server if the setup packaging is not granular enough. This is often the case with ASP.NET, which installs all of the ASP.NET modules and handlers whether or not they are all necessary in your application.

Sounds simple, right? Unfortunately, the biggest challenge lies in step 1, determining the set of features your application needs. Doing this requires knowing which functionality can be safely removed without negatively affecting your application's functionality or compromising its security. That's right—you can actually end up making your server *a lot* less secure if you remove a security-sensitive feature. For example, if you remove an authorization module that is responsible for validating access to an administration page in your application, you may end up allowing anonymous users to take administrative action on your server! Likewise, removing certain features may contribute to decreased performance or reduced stability. Or removing these features may simply break your application.

Caution It is possible to have IIS configuration that configures an IIS feature to perform a certain function and yet to not have this function be performed by the server if the module that is responsible for this function is not enabled. This can happen if someone disables the module or configures its preconditions to prevent it from running in a specific application pool. Because of this, you need to make sure that the required modules are present and are correctly preconditioned to insure the correct operation of your application.

Therefore, it is important to understand what features your application requires, and which it does not. To help with this, you can consult Table 12-9, which illustrates the function played by each built-in IIS 7.0 module whose removal may have a security impact on the server.

Table 12-9 Function of Built-In Modules with Security Implications

Module	Purpose and Removal Effect
Anonymous Authentication Module	**Purpose:** Authenticates the request with an anonymous user if no other authentication mechanism is present. **If removed:** Access to resources will be denied for anonymous requests.
Basic Authentication Module	**Purpose:** Supports basic authentication. **If removed:** Clients will not be able to authenticate with basic authentication.
Certificate Mapping Authentication Module	**Purpose:** Supports client certificate authentication. **If removed:** Clients will not be able to authenticate with client certificates.
Configuration Validation Module	**Purpose:** Validates ASP.NET configuration in integrated mode **If removed: Strong warning—**ASP.NET applications that define modules and handlers using legacy configuration will silently run in integrated mode, but the modules will not be loaded. This may result in unexpected application behavior and security vulnerabilities for unmigrated ASP.NET applications.

Table 12-9 Function of Built-In Modules with Security Implications

Module	Purpose and Removal Effect
CustomError Module	**Purpose:** Detailed error messages will not be generated for IIS errors. **If removed: Strong warning—**Sensitive application error information may be sent to remote clients.
Default Authentication	**Purpose:** Supports the ASP.NET DefaultAuthentication_OnAuthenticate event. When ASP.NET is configured to use the Forms Authentication mode, removing this module may lead to errors in other ASP.NET modules during anonymous requests. **If removed: Warning—**The DefaultAuthentication_OnAuthenticate event will not be raised, so any custom authentication code depending on this event will not run. This is not common.
Digest Authentication Module	**Purpose:** Supports digest authentication. **If removed:** Clients will not be able to use digest authentication to authenticate.
File Authorization	**Purpose:** Verifies that the authenticated client has access to physical resources. **If removed: Strong warning—**Access may be granted to resources that deny access to the authenticated user.
Forms Authentication	**Purpose:** Supports forms-based authentication. **If removed:** Clients will not be able to use forms authentication to authenticate.
HttpCache Module	**Purpose:** Supports IIS output caching and kernel caching of responses. **If removed: Warning**—Response output caching will not occur, possibly resulting in increased load on the server and in the worst case Denial of Service (DoS) conditions.
HttpLogging Module	**Purpose:** Supports request logging. **If removed: Warning—**Requests may not be logged.
IISCertificate Mapping Authentication Module	**Purpose:** Supports IIS configuration–based client certificate authentication. **If removed:** Clients may not be able to authenticate with client certificates against IIS configuration.
HttpRedirection Module	**Purpose:** Supports configuration-based redirection rules. **If removed: Warning—**If the application depends on redirection rules for restricting access to content, removing this module may make otherwise protected resources available.
IsapiFilter Module	**Purpose:** Supports ISAPI filters. **If removed: Strong warning—**ISAPI filters that enforce access or have other security functionality will not run.
OutputCache	**Purpose:** Supports ASP.NET response output caching. **If removed: Warning—**ASP.NET response output caching will not occur, possibly resulting in increased load on the server and in the worst case Denial of Service (DoS) conditions.

Table 12-9 Function of Built-In Modules with Security Implications

Module	Purpose and Removal Effect
RequestFiltering Module	**Purpose:** Enforces various request restrictions and protects hidden content. **If removed: Strong warning**—Removing this module may result in protected content being served. It may also lead to nonspecific security vulnerabilities resulting from allowing unrestricted request input into the application.
RoleManager	**Purpose:** Supports the ASP.NET roles functionality. **If removed: Strong warning—**Roles for the authenticated user may not be available, which may cause authorization decisions to be affected. Typically, this will only restrict access, but in some cases where access is denied based on roles, this may grant access to otherwise unauthorized users.
Static Compression Module	**Purpose:** Supports compression of static resources. **If removed: Warning—**Removing this module may result in significantly higher bandwidth for the site, because compression of static resources will be disabled.
Url Authorization	**Purpose:** Supports declarative access rules. **If removed: Strong warning—**URL authorization access rules will be ignored, and access may be granted to unauthorized users.
Url Authorization Module	**Purpose:** Supports declarative ASP.NET access rules. **If removed: Strong warning—**ASP.NET url authorization access rules will be ignored, and access may be granted to unauthorized users.
Windows Authentication	**Purpose:** Supports NTLM and Kerberos authentication. **If removed:** Clients will be unable to authenticate with NTLM or Kerberos Windows authentication.
Windows Authentication Module	**Purpose:** Supports raising the ASP.NET WindowsAuthentication_OnAuthentication event. **If removed: Warning**—WindowsAuthentication_OnAuthentication event will not be raised, so any custom ASP.NET authentication code dependent on this event will not run. Note that this module is not required for Windows authentication.

You should always verify that the application does indeed have all of the required modules enabled after deployment. In addition, you should always test the application whenever the module configuration changes to insure its correct operation with the new module set. Armed with the knowledge of which modules can be safely removed, you can take advantage of IIS 7.0's modularity to significantly reduce the surface area of your server, without accidentally reducing its security.

Understanding and Reducing the Privilege of Code that Runs on Your Server

Now that you have reduced the surface area of your server to the acceptable minimum, you need to secure the remaining functionality. This is typically done in two ways: by restricting

the inputs to the application as much as possible by using security features such as authorization and request filtering (IIS 7.0's version of UrlScan) and by reducing the privilege with which the code in the application executes so that even if it is compromised, it is limited in the amount of harm it can do. You can learn more about both of these approaches in Chapter 14.

The former approach is an extension of reducing the surface area approach you took earlier, attempting to block as many of the attack vectors as possible by constraining the input to the server. The latter approach uses the principle of least privilege and focuses on what happens if the functionality on the server is compromised. With an understanding of how the extensibility code executes in the context of IIS, you can do much to reduce its privilege, which often makes compromising the server a lot harder or impossible. Also, this knowledge helps you understand the trust requirements for adding features or application components to the server.

Table 12-10 illustrates the privilege with which IIS Web server modules execute on the server.

Table 12-10 Module Privileges

Feature	Execution Scope	Privilege Level	Who Can Add
Native modules	IIS worker process	Application pool identity	Administrator
Managed modules and handlers	ASP.NET appdomain	Application pool identity (default) OR Authenticated user AND Limited by ASP.NET trust level	Application owner
ISAPI filters	IIS worker process	Application pool identity	Administrator
ISAPI extensions	IIS worker process	Authenticated user (default) OR Application pool identity	Administrator
CGI programs	CGI program process (single-request)	Authenticated user (default) OR Application pool identity	Administrator
FastCGI programs	FastCGI program process	Application pool identity	Administrator

The majority of IIS extensibility is hosted within a long-running IIS worker process (everything except CGI and FastCGI programs that execute out of process), which executes under the configured application pool identity (Network Service by default). This includes native modules as well as ISAPI extensions and filters (managed modules and handlers are also included, but they provide an additional constrained execution model that is discussed later in this chapter). This code, therefore, can do everything that the application pool identity is allowed to do, based on the privileges granted to the identity by the system and rights granted by ACLs on Windows resources. Reducing the privilege of this identity from Local System in

IIS 5.1 to Network Service in IIS 6.0 was one of the fundamental security improvements that enabled IIS 6.0 to achieve its stellar security record. You can learn how to take advantage of reducing the privilege of the IIS application pools in Chapter 14.

Remember that despite all other constraining measures that may be in place, including ASP.NET Code Access Security, the privileges and rights granted to worker process that contains the code define what code in the process may or may not do (when impersonating, you also need to consider the rights and privileges assigned to the impersonated identity). In other words, when you add code to the server, even if it is application code, assume that it can do everything that your worker process is allowed to do. By maintaining least privilege application pools, you can significantly reduce the damage to the server in the case of an application compromise.

The story is slightly different when it comes to managed (ASP.NET) module and handler components. These components are hosted within the ASP.NET application, and in addition to being limited by the IIS worker process, they are also limited by the .NET Code Access Security (CAS) policy configured for the ASP.NET appdomain. This means that managed modules and handlers can execute with a lower privilege than the one granted by the IIS worker process identity.

By default, ASP.NET applications are configured to execute with Full trust, which means that they are not additionally constrained. By configuring the application to execute with lower trust levels via the system.web/trust configuration section, you can create an additional limitation on the execution of .NET code that precludes managed modules from performing certain actions and accessing resources outside of those located in their subdirectories. You can learn more about the built-in trust levels in Chapter 14.

Note You can also use IIS Manager to configure the default trust level for ASP.NET applications or the trust level for a particular application on your server.

The recommended trust level is Medium. At this trust level, the application cannot access resources that do not belong to it, though it can still use most ASP.NET features and execute code that affects its own operation. At this trust level, multiple applications running within a single application pool are largely isolated from each other, making this level the correct one to use for shared hosting (although it is preferable to host each customer in a separate fully isolated application pool), where hosted applications are allowed to upload code to the server.

You should take advantage of the Medium trust level where possible to further reduce the privilege of the managed components of your application. Be aware that some ASP.NET applications or modules may not function in Medium trust, due to the use of .NET APIs that required a higher trust level. The number of these applications is getting smaller, due to both API improvements in .NET Framework 2.0+ and application improvements to facilitate operation in partial trust environments. Also, the ASP.NET run time may experience reduced

performance in a partial trust. This needs to be evaluated in each specific case to determine whether it is a significant enough factor to warrant using higher trust levels.

Note Though ASP.NET trust levels provide an additional way to constrained the execution of managed code on your server, they should not be used as a substitute for reducing the privilege of the hosting IIS worker process.

CGI and FastCGI programs are not hosted inside the IIS worker process, but instead execute inside their own processes that are spawned by the IIS worker process. CGI programs by default execute under the identity of the authenticated user, although you can configure them to execute with the identity of the worker process. Therefore, when using CGI programs, be aware that the code has the privilege of the invoking user. If that user is an administrator on the server, the code can perform administrative tasks that can lead to complete server compromise if the code is successfully exploited.

Note When using anonymous authentication while impersonating, the new IIS_IUSR account will be used to execute the CGI, FastCGI, and ISAPI extension–based applications (unless the server is configured to use the process identity instead). Unless you are using the IIS_IUSR account to use lower privilege for the executing code than the one provided by the worker process identity, you should consider using the worker process identity instead to manage one less account. You can do this by setting the *userName* attribute of the anonymousAuthentication section to "".

FastCGI programs always execute with the identity of the IIS worker process, so they have the same privilege as code in the IIS worker process. FastCGI does provide a way for the FastCGI program to impersonate the authenticated user, which can be done by PHP in FastCGI mode. In this case, the same warning applies to code running in the worker process as when running CGI programs.

Important In a number of cases, server code impersonates the authenticated user. This is done by default for all ISAPI extensions, ASP pages, PHP pages running in CGI, ISAPI or FastCGI mode, and ASP.NET applications that enable impersonation. If impersonation is used, be aware that the code will execute with very high privileges when the request is made by a user who has administrative privileges on this machine. You should strongly consider disallowing administrative users from using your application except when necessary.

All that said, the bottom line is that you must trust the source of the code running on your server as far as the privilege level under which this code executes. If you do not trust the code, you must insure that it runs with a low privilege level by constraining it with a very low privilege application pool identity. If the code is native, that is the best you can do. If the code

is managed, you can use ASP.NET's partial trust levels to further constrain it, which provides a foundation for allowing third-party application code to run on your server.

If you do trust the code, you can harden it against unforeseen exploits by reducing its privilege as much as possible using the techniques described earlier in this chapter. Though you can never be completely sure that a piece of code is foolproof against attacks, using the least privilege principle can significantly limit or eliminate damages.

Locking Down Extensibility

Now that you have built a minimal surface area Web server that runs with least privilege, you need to make sure it stays that way. If you are running a dedicated server, this is less of an issue because you are the one who controls the configuration that defines which components are enabled and how they execute. However, if you delegate application management to someone else, as is the case with shared hosting servers and sometimes departmental servers, things are different.

To understand this, let's first look at the difference in extensibility delegation between IIS 6.0 and IIS 7.0. In IIS 6.0, the administrator in the metabase controls the server functionality. If a user wants to serve .php3 pages with the PHP scripting engine, they need to contact the administrator to create a handler mapping for their application. The same is the case for adding or removing an ISAPI filter. In IIS 7.0, the delegated configuration system enables applications to remove or add new functionality in some cases, without requiring administrator level changes. This is nice for allowing xcopy deployment of applications that define their own configuration and functionality, and reducing total cost of ownership. However, in some cases, this may be undesired from a security perspective, and so the administrator has a fine degree of control over what changes are allowed at the application level. This is done by controlling configuration delegation for the system.webServer/handlers and system.webServer/modules configuration sections via configuration locking.

In the default IIS 7.0 installation, both of these sections are locked at the server level. This means that application A cannot add new modules or create new handler mappings, and application B cannot remove or change existing modules or handler mappings.

This is a very restrictive state that prevents many ASP.NET applications from working correctly, because ASP.NET applications often need to declare new modules and create new handler mappings. In general, it prevents IIS applications from defining their handler mappings and modules in their configuration. Because of this, when the ".NET Extensibility" or the "ASP.NET" role service is installed on the server, these sections are unlocked. This allows applications to specify a configuration that does the following:

1. Enable/add new managed modules.
2. Disable/remove existing modules.
3. Add new handler mappings.
4. Override/remove existing handler mappings.

Because adding new native modules requires installing them at the server level, and adding new ISAPI filters/extensions and CGI/FastCGI programs also requires configuration changes at the server level (the isapiCgiRestrictions and fastCgi sections), applications cannot introduce new native code. However, they *can* introduce new managed modules and handlers. Because of this, in shared environments where the application is not trusted by the administrator, server administrators must do one of the following:

- Prevent new managed modules/handlers from being added by locking the modules and handlers sections. This will break many applications (especially ASP.NET applications running in Integrated mode).
- Reduce the trust level of the application to Medium trust to constrain the execution of managed code.
- Use a low-privilege application pool to host the application.

The application can also by default disable any of the modules defined at the server level. This can be a problem if the server administrator wants to mandate the presence of a particular module, such as a bandwidth monitor module or logging module that tracks the operation of the application. To counteract that, the server administrator can lock each module that should not be removable at the server level, preventing it from being removed by the application. This can be done by adding a *lockItem = "true"* attribute to each module element in the modules configuration section at the server level. In fact, when the modules section is unlocked, ASP.NET setup automatically locks each native module that is installed against removal (in some cases, you may want to unlock some of these modules if you do not mind them being disabled by the application).

Because the application can also create new handler mappings, the application can override mappings defined at the server level. The locking approach does not work here because new mappings take precedence over earlier mappings, so it is not necessary to remove existing mappings to redirect them to other handler types. However, the ability to remap requests in the application to another handler is not a security concern outside of the ability to execute the code, which is already controlled by the application trust level and/or the application pool identity. The handlers section does expose the *accessPolicy* attribute, which controls what access levels are granted to the handlers. This attribute is by default locked at the server level, so the application cannot modify it.

The trust level configuration for the application is also something that should be locked at the server level. By default, it isn't—so the application can elevate its own trust level to Full to remove the constraint on the execution of the .NET code. Server administrators should always lock this configuration in the framework's root Web.config file to prevent it from being overridden when they are relying on the partial trust security model for applications. For convenience, you can do this using the following Appcmd command.

```
AppCmd lock config /section:trust /commit:MACHINE/WEBROOT
```

This prevents applications from overriding the trust level setting in the framework root Web.config.

On the Disc Browse the CD for additional tools and resources.

Summary

The modular architecture of IIS 7.0 Web server provides the foundation for many key production scenarios, enabling you to build low-footprint specialized servers and leverage rich add-on functionality provided by end-to-end extensibility. Though traditionally the domain of developers, managing Web server extensibility becomes a central IT theme in IIS 7.0, providing both opportunities and challenges for the administrator. Armed with the right know-how, you can effectively leverage the modularity of the server to achieve your business goals today with built-in IIS 7.0 features and when you take advantage of Microsoft or third-party modules to enhance your server in the future.

In the next two chapters, we will cover the extensibility model exposed by the configuration system, the administration stack, and IIS Manager, which complete the end-to-end extensibility picture for the server.

Additional Resources

These resources contain additional information and tools related to this chapter:

- Chapter 11, "Hosting Application Development Frameworks," for information on enabling and hosting common application framework technologies on IIS 7.0.
- Chapter 14, "Implementing Security Strategies," for information on locking down the server.
- The blog at *http://www.mvolo.com* for in-depth coverage of many IIS 7.0 extensibility and module management topics.
- The Web site *http://www.iis.net* for articles on extending IIS 7.0.
- The IIS 7.0 Operations Guide, available at *http://technet2.microsoft.com/windowsserver2008/en/library/d0de9475-0439-4ec1-8337-2bcedacd15c71033.mspx*.

Chapter 14

Implementing Security Strategies

In this chapter:

On the Disc Browse the CD for additional tools and resources.

The predecessor of Internet Information Services (IIS) 7.0—IIS 6.0—established a high bar for providing a secure Web server platform. IIS 7.0 builds on much of the IIS 6.0 feature codebase and secure practices formulated during the IIS 6.0 development life cycle. It also builds on many of the design principles that contributed to the excellent security track record of IIS 6.0, taking them further to improve the security of the Web server and applications that run on it. This includes the secure by default design and an emphasis on reducing the surface area and using least privilege to minimize the risk of exposed application vulnerabilities being successfully exploited by an attacker. IIS 7.0 makes it easier than ever before to apply these crucial security principles by offering modularity that enables you to build minimal surface area Web servers and running your applications in an isolated environment.

Many of the security features in IIS 6.0 remain applicable in IIS 7.0. In this chapter, we will start by reviewing the changes to the security features as well as the new features that IIS 7.0 introduces to help improve the Web server and application security. Then, we will look at applying the general security principles of reducing the surface area and using least privilege to further strengthen the security of the Web server. Finally, we will take a detailed look at using the security features in IIS 7.0, including the authentication and authorization features, securing network communications with Transport Layer Security (TLS), and safeguarding the configuration.

Security Changes in IIS 7.0

IIS 7.0 builds on the security focus established in its predecessor, IIS 6.0. As a result, the overwhelming majority of the core security principles and features established in IIS 6.0 are still in use today. However, IIS 7.0 does introduce improvements to help enhance the security of the Web server:

- **The anonymous user configured by default for anonymous authentication is the new built-in IUSR account.** This account is built in and does not require a password that needs to be renewed and synchronized between servers. Additionally, permissions set for IUSR accounts are effective when copied to another IIS 7.0 server because the IUSR account has a well-known Security Identifier (SID) that is the same on every computer. For more information, see the section titled "Anonymous Authentication" later in this chapter.
- **The IIS_WPG group has been replaced with the built-in IIS_IUSRS group.** This group is built in and enables permissions set for IIS_IUSRS to remain effective when copied to another IIS 7.0 server because it has a well-known Security Identifier (SID). In addition, this SID is automatically injected into the worker process token for each IIS worker process, eliminating the need for manual group membership for any custom application pool identities. For more information, see the section titled "Set NTFS Permissions to Grant Minimal Access" later in this chapter.
- **Anonymous authentication can be configured to use the application pool identity.** This enables the content to require permissions only for the application pool identity when using anonymous authentication, simplifying permission management. For more information, see the section titled "Set NTFS Permissions to Grant Minimal Access" later in this chapter.
- **IIS worker processes automatically receive a unique application pool Security Identifier (SID) that you can use to grant access to the specific application pool to enable application isolation.** For more information, see the section titled "Isolating Applications" later in this chapter.
- **Configuration isolation automatically isolates server-level configuration for each application pool.** The global server-level configuration contained in applicationHost.config is automatically isolated by creating filtered copies of this configuration for each application pool and preventing other applications pools from being able to read this configuration. For more information, see the section titled "Understanding Configuration Isolation" later in this chapter.
- **Virtual directories can specify fixed credentials regardless of whether they point to Universal Naming Convention (UNC) shares or a local file system.** Unlike IIS 6.0, which supports fixed credentials for specifying access to UNC shares only, IIS 7.0 enables fixed credentials to be used for any virtual directory.
- **Windows Authentication is performed in the kernel by default.** This improves the configurability of the Kerberos protocol on the server. It also improves the performance

of Windows Authentication. However, it may affect some applications that have custom clients that presend authentication credentials on the first request. This behavior can be turned off in the configuration. For more information, see the section titled "Windows Authentication" later in this chapter.

- **The new Request Filtering feature provides extended URL Scan functionality.** You can use the new Request Filtering feature to protect your Web server against nonstandard or malicious request patterns and additionally protect specific resources and directories from being accessed. For more information, see the section titled "Request Filtering" later in this chapter.
- **The new URL Authorization feature enables applications to control access to resources through configuration-based rules.** The new URL Authorization feature provides flexible configuration-based rules to control access to application resources in terms of users and roles, and it supports the use of the ASP.NET Roles service. For more information, see the section titled "URL Authorization" later in this chapter.

Additionally, IIS 7.0 introduces several changes to existing security features and removes several deprecated security features that could impact your application. These changes to security-related features are listed here:

- **IIS 6.0 Digest Authentication is no longer supported.** It is being replaced by Advanced Digest Authentication (now simply referred to as Digest Authentication), which does not require the application pool to run with LocalSystem privileges. See the section titled "Digest Authentication" later in this chapter for more information.
- **.NET Passport Authentication is no longer supported.** The .NET Passport Authentication support is not included in Windows Vista and Windows Server 2008, and therefore IIS 7.0 does not support it.
- **IIS 6.0 URL Authorization is no longer supported.** The IIS 6.0 URL Authorization was overly complex and not often used. It has been replaced by the new configuration-based URL Authorization feature. See the section titled "URL Authorization" later in this chapter for more information.
- **IIS 6.0 Sub-Authentication is no longer supported.** The Sub-Authentication feature enabled IIS 6.0 Digest Authentication (which has been removed) and synchronized anonymous account passwords (the anonymous account now uses the new built-in IUSR account that does not have a password). It is no longer needed in IIS 7.0 and therefore has been retired.
- **IIS Manager no longer provides support for configuring IIS Client Certificate Mapping Authentication.** You can edit the configuration directly, use Appcmd from the command line, or use another configuration application programming interface (API) to configure this feature. For more information, see the section titled "IIS Client Certificate Mapping Authentication" later in this chapter.

- **Several authentication and impersonation differences exist in ASP.NET applications when running in the default Integrated mode.** This includes an inability to use both Forms authentication and an IIS authentication method simultaneously, and an inability to impersonate the authenticated user in certain stages of request processing. For more information on security changes impacting ASP.NET applications, see the list of breaking changes at *http://mvolo.com/blogs/serverside/archive/2007/12/08/IIS-7.0-Breaking-Changes-ASP.NET-2.0-applications-Integrated-mode.aspx*.
- **Metabase access control lists (ACLs) are no longer supported.** With the new configuration system, you cannot set permissions on individual configuration settings. IIS 7.0 provides built-in support for delegating configuration settings to Web site and application owners, replacing metabase ACLs as a mechanism for configuration delegation. For more information, see the section titled "Controlling Configuration Delegation" later in this chapter.
- **Metabase auditing is no longer supported.** The ability to audit changes to specific configuration settings is not supported out of the box. This is a consequence of IIS 7.0 not supporting metabase ACLs.

Reducing Attack Surface Area

Reducing the attack surface area of the Web server is a key strategy in reducing the risk of a security vulnerability being successfully exploited by an attacker. The principle of attack surface area reduction is not exclusive to Web servers—it is generally accepted as one of the most direct ways to improve the security of any software system. When applied to IIS 7.0, it provides the following benefits:

- It directly reduces the number of features and services exposed by the Web server to outside clients, minimizing the amount of code available for an attacker to exploit.
- It reduces complexity, which makes it easier to configure the Web server in a secure manner.
- If a vulnerability is exposed, the uptime of the Web server is not affected as much, because if the component affected by the vulnerability is not installed, it is not necessary to take the Web server offline or patch it immediately.

IIS 7.0 gives you an unparalleled ability to reduce the attack surface area of the Web server through its modular architecture by enabling you to remove all functionality other than what is absolutely necessary to host your application. By leveraging this ability, you can deploy low-footprint Web servers with minimal possible surface area.

After installing the minimal set of features, you can further reduce the surface area of the Web server by configuring your application to operate with the minimal functionality, for example, configuring which application resources should be served.

In the rest of this section, we will review the cumulative process for reducing the surface area of the Web server and your application. This process includes the following steps:

1. Reduce the surface area of the Web server.
    a. Install the minimal required set of Web server features.
    b. Enable only the required Internet Server Application Programming Interface (ISAPI) filters.
    c. Enable only the required ISAPI extensions.
    d. Enable only the required Common Gateway Interface (CGI) applications.
    e. Enable only the required FastCGI applications.
2. Reduce the surface area of the application.
    a. Enable only the required modules.
    b. Configure the minimal set of application handler mappings.
    c. Set Web site permissions.
    d. Configure a minimal set of MIME types.

The modular architecture of IIS 7.0 gives you the ability to install only the Web server features required for the correct operation of your Web server. This forms the foundation of the surface area reduction strategy.

In addition, you can continue to control what extensions that do not use the IIS 7.0 modular extensibility model can execute on the server. This includes ISAPI extensions and filters and CGI and FastCGI programs.

Installing the Minimal Required Set of Web Server Features

The IIS 7.0 modular feature set comprises more than 40 individual Web server modules that provide various request processing and application services. The Web server core engine retains only the minimal set of functionality needed to receive the request and dispatch its processing to modules. You can leverage this architecture to deploy minimal surface area Web servers by installing only the modules that are required for the Web server's operation.

The modular feature set provided in IIS 7.0 is fully integrated with Windows Setup. This means that you can install or uninstall most of the IIS 7.0 modules by installing IIS 7.0 features directly from the Turn Windows Features On Or Off page in Windows Vista, or the Web Server (IIS) role in Server Manager on Windows Server 2008 as shown in Figure 14-1. Each feature typically corresponds to one module (or in some cases several modules) and installs any corresponding configuration information as well as feature dependencies.

The default installation of IIS 7.0 includes only the features necessary for IIS 7.0 to function as a static file Web server. In many cases, this may not be sufficient to properly host your

application, so you will need to install additional features, including support for hosting dynamic application technologies. When you do this, you will be prompted to install any dependencies of the feature you are installing and configure the proper default configuration for that feature.

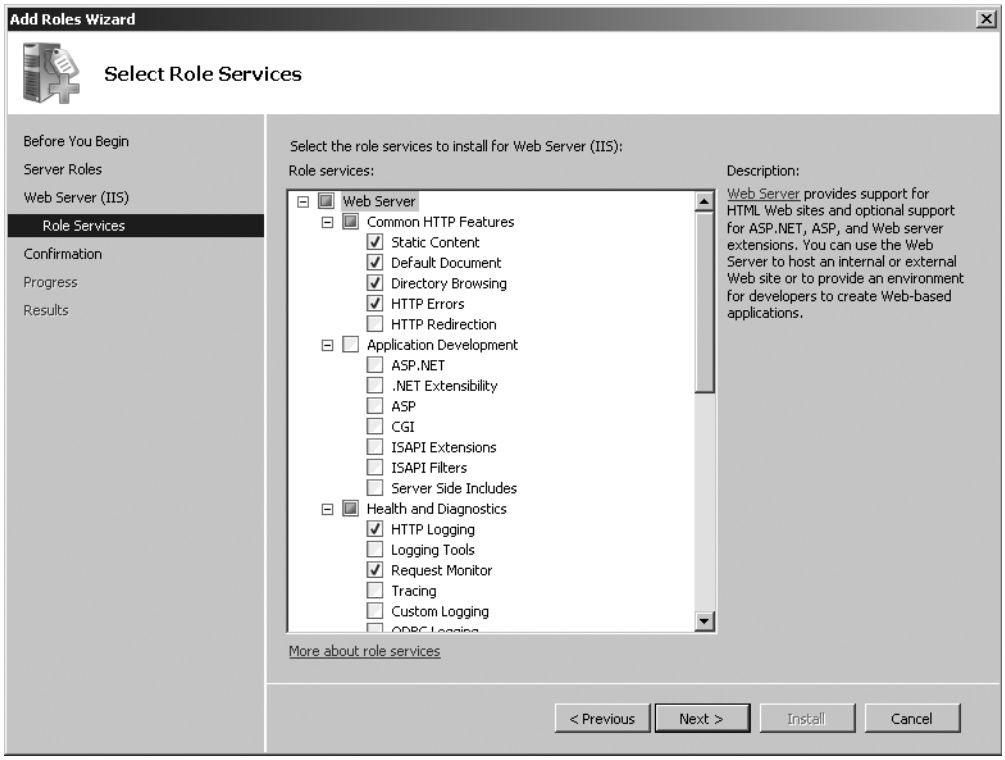


Figure 14-1 Installing IIS 7.0 using the Add Roles Wizard.

Caution Do not install all the IIS 7.0 features if you are unsure of what you need. Doing so can unnecessarily increase the surface area of the Web server.

By ensuring that only the required modules are installed, you can significantly reduce the surface area of the Web server. This provides the following benefits:

- Removes the potential for an attacker to exploit known or future threats in features that are not installed.
- Reduces management complexity, making it easier to configure the server in a secure manner.
- Reduces the downtime and costs associated with reacting to a vulnerability or applying patches. If the patched component is not installed, you do not need to take the server

offline to perform the patch. You can also perform patching on your own schedule instead of being forced to perform it immediately if a vulnerability is found.

Note When you apply a patch to a component of a Web server feature that is not installed, it is stored in the operating system installation cache. This way, when you install the feature in the future, it will use the patched version automatically. Therefore, be sure to continue installing all operating system updates, even if the corresponding features are not currently being used on the server.

To reduce the surface area of the Web server, you should take the following steps:

1. **Determine the set of features your applications need.** In the majority of cases, you should be able to tell what features are required by your application by reviewing the list of setup components and comparing it with your application's requirements. As a guide, you can often use the recommended set of modules for specific application workloads. You can find more information on recommended installation workloads at *http://www.iis.net/articles/view.aspx/IIS7/Deploy-an-IIS7-Server/Installing-IIS7/Install-Typical-IIS-Workloads?Page=2*. You should exercise caution when removing Web server features that are security sensitive, because doing so may have a negative impact on your server's security. To review the list of modules that have a security impact, see the section titled "Taking Advantage of Componentization to Reduce the Security Surface Area of the Server" in Chapter 12, "Managing Web Server Modules."
2. **Install only the required features.** After you have determined the required features, you should install them using the roles or features wizards. For more information on the options for installing IIS 7.0 features, see Chapter 5, "Installing IIS 7.0." When in doubt, do not install all features, because doing so will result in an unnecessary surface area increase.
3. **Install only the required third-party modules.** IIS 7.0 applications may require third-party modules to be installed to add additional functionality or replace a built-in IIS 7.0 feature. You should exercise caution when installing any module on the Web server and make sure that you trust its source. Installing untrusted or buggy modules can compromise the security of the Web server or negatively affect its reliability and performance. For information about installing third-party modules, see Chapter 12.
4. **Test your application.** You should always test your application to ensure that it operates correctly given the installed feature set. Your application may experience errors if a required module is not installed. The symptoms of this error will depend on the service provided by the missing module. If your testing shows an error and you believe that it is due to a missing feature, make sure that the error is removed or changed by installing that specific feature. If the error remains, uninstall the feature and try again. Never blindly install multiple or all features to get the application to work.

When you run multiple applications on the same Web server, you will need to install the superset of the modules required by each application. You can then further reduce the surface area of each application by controlling which modules are enabled at the application level. We will review how to do this in the section titled "Enabling Only the Required Modules" later in this chapter.

For more information on using the roles or features wizards to install IIS 7.0 features and other available features, see Chapter 5. For more information on installing and enabling modules, including third-party modules, see Chapter 12.

Enabling Only the Required ISAPI Filters

IIS 6.0 provides support for ISAPI filters, to allow third parties to extend IIS request processing. IIS 7.0 replaces ISAPI filters with IIS 7.0 modules as the preferred mechanism for extending the Web server. However, IIS 7.0 continues to support ISAPI filters for backward compatibility reasons.

Note To enable ISAPI filters to work on IIS 7.0, the ISAPI Filters role service must be installed. This role service installs the IsapiFilterModule module, which provides support for hosting ISAPI filters. If this module is removed, ISAPI filters will not be loaded. This role service is not enabled by default; it is however enabled when the ASP.NET role service is installed.

If your Web server uses ISAPI filters, to minimize the Web server surface area you should ensure that only the required ISAPI filters are enabled.

Note You must be a server administrator to enable ISAPI filters.

To properly configure ISAPI filters, you should take the following steps:

1. **If your Web server uses ISAPI filters, install the ISAPI Filters role service.** Without this role service, the ISAPI filters will not be loaded and therefore may create a security risk if they are responsible for security-sensitive functionality.
2. **If your Web server does not use ISAPI filters, *do not* install the ISAPI Filters role service.** This eliminates the possibility of unwanted ISAPI filters being configured on your server.
3. **Determine the ISAPI filters that your application requires.** In the majority of cases, your Web server should not require any ISAPI filters (with the exception of the ASP.NET ISAPI filter; see the note later in this section). Therefore, you will typically need to configure ISAPI filters only if you are migrating an existing application from previous versions of IIS that require specific ISAPI filters, or if you are installing a new third-party ISAPI filter.

4. **Enable the required ISAPI filters.** You can control which ISAPI filters are enabled on your server, and for a specific Web site, by using IIS Manager.

To use IIS Manager to configure the ISAPI filters, click the Web server node or Web site node in the tree view and then double-click ISAPI Filters, as shown in Figure 14-2. Exercise extreme caution when installing third-party ISAPI filters and be sure you trust their source. Installing untrusted or buggy ISAPI filters can compromise the security of the Web server or negatively affect its reliability.

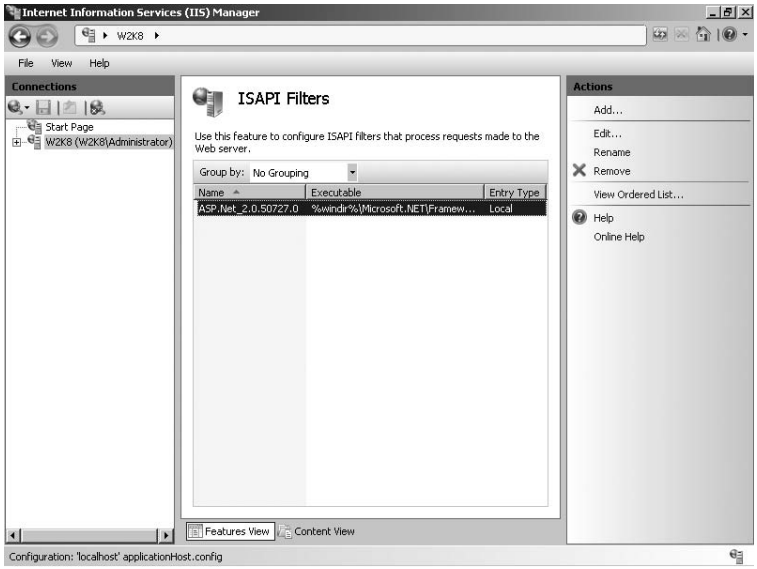


Figure 14-2 Using IIS Manager to configure ISAPI filters.

Note IIS 7.0 does not install any ISAPI filters by default. However, ASP.NET 1.1 and ASP.NET 2.0 will install an ISAPI filter named ASP.NET_2.0.50727.0. This filter is required for cookie-less ASP.NET features to work properly. You should not remove this filter.

You can also control which filters are enabled for the Web server or for a specific Web site by editing the *system.webServer/isapiFilters* configuration section directly, with the Appcmd command line tool, or with another configuration API.

Enabling Only the Required ISAPI Extensions

IIS 6.0 provides support for ISAPI extensions, which allows third parties to extend IIS request processing by returning responses for specific content types. IIS 7.0 replaces ISAPI extensions with IIS 7.0 modules as a preferred mechanism for extending IIS. However, IIS 7.0 continues to support ISAPI extensions for backward compatibility reasons.

Note To enable ISAPI extensions to work on IIS 7.0, the ISAPI Extensions role service must be installed. This role service installs the IsapiModule module, which provides support for hosting ISAPI extensions. If this module is removed, ISAPI extensions will not be loaded. This role service is not enabled by default, but it is enabled when ASP.NET is installed.

Today, dynamic application framework technologies frequently use ISAPI extensions to interface with IIS. Therefore, it is likely that if you are using dynamic application technologies, you will need to use ISAPI extensions. For example, both ASP.NET (for Classic mode applications) and ASP are implemented as ISAPI extensions.

If your Web server uses ISAPI extensions, to minimize the Web server surface area you should ensure that only the required ISAPI extensions are enabled.

Note You must be a server administrator to enable ISAPI extensions.

To properly configure ISAPI extensions, you should take the following steps:

1. **If your Web server uses ISAPI extensions, install the ISAPI Extensions role service.** Without this role service, the ISAPI extensions will not be loaded, and requests to resources mapped to ISAPI extensions will return errors.
2. **If your Web server does not use ISAPI extensions, *do not* install the ISAPI Extensions role service.** This eliminates the possibility of unwanted ISAPI extensions being configured on your server.
3. **Configure the allowed ISAPI extensions.** Each ISAPI extension must be allowed to execute on the server before it can be used. You can use IIS Manager to configure all ISAPI extensions that are allowed to execute on the server. Doing so is explained in more detail later in this section. Exercise extreme caution when allowing third-party ISAPI extensions and be sure you trust their source. Installing untrusted or buggy ISAPI extensions can compromise the security of the Web server or negatively affect its reliability.
4. **Configure the desired handler mappings.** To use ISAPI extensions, you need to create handler mappings that map allowed ISAPI extensions to specific content types in your application. For more information on creating handler mappings for ISAPI extensions, see Chapter 12. We will discuss securing handler mappings in the section titled "Configuring the Minimal Set of Handler Mappings" later in this chapter.

You must explicitly allow any ISAPI extension that has to execute on your server. When you allow a specific ISAPI extension path, any application on the server can load this extension, if the server configures a handler mapping to this extension. Table 14-1 specifies the common ISAPI extensions and when they are installed.

Table 14-1 Common ISAPI Extensions

ISAPI Extension	Default State	When Installed
Active Server Pages	Allowed	ASP role service is installed
ASP.NET v1.1.4322	Not Allowed	.NET Framework v1.1 SP1 is installed
ASP.NET v2.0.50727	Allowed	ASP.NET role service is installed

On IIS 6.0, you have to explicitly allow the ISAPI extensions corresponding to ASP and ASP.NET 2.0. On IIS 7.0, these ISAPI extensions are automatically allowed when you install the corresponding role services. In addition, only ASP.NET applications running in Classic mode use the ASP.NET 2.0 ISAPI extension. It is a more reliable practice to use the roles or features wizards to control the availability of these features, instead of allowing or not allowing them in the ISAPI and CGI Restrictions. However, you still need to manually enable the ISAPI extension for ASP.NET v1.1.

On IIS 6.0, you can allow an ISAPI extension in the Web Service Extension Restriction List. On IIS 7.0, you can use IIS Manager to do this by clicking the Web server node in the tree view and then double-clicking ISAPI And CGI Restrictions to open the feature shown in Figure 14-3. To add a new ISAPI extension, click Add in the Actions pane and then enter the exact path of the ISAPI extension. If you would like to allow the ISAPI extension to execute, check the Allow Extension Path To Execute check box. You can also allow or deny existing extensions.

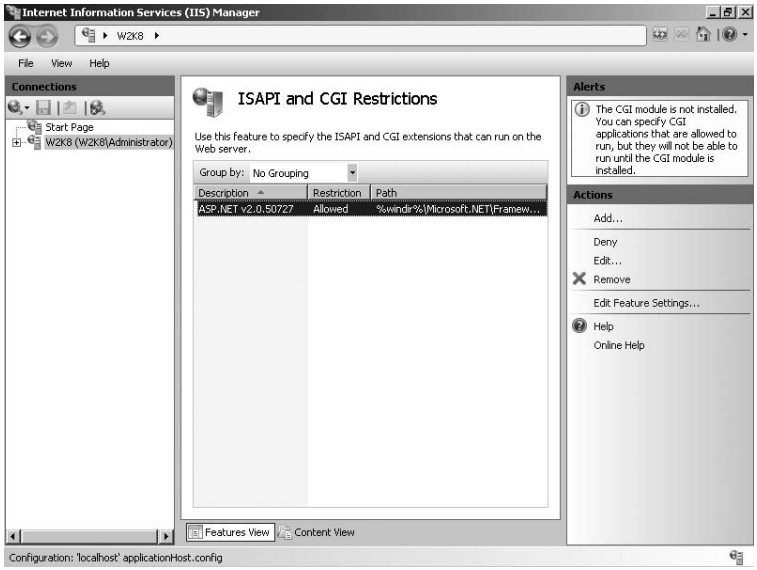


Figure 14-3 Allowing ISAPI extensions in the ISAPI and CGI Restrictions by using IIS Manager.

In addition to using IIS Manager, you can also edit the *system.webServer/security/isapiCgiRestriction* configuration section directly by using the Appcmd command line tool or with another configuration API. For more information about configuring enabled ISAPI

extensions, see the section titled "Adding Entries to the ISAPI CGI Restriction List (Formerly Web Service Restriction List)" in Chapter 12.

Enabling Only the Required CGI Programs

IIS 7.0 continues to support CGI programs as one of the ways to extend the functionality of the Web server.

Note To enable CGI programs to work on IIS 7.0, the CGI role service must be installed. This role service installs the CgiModule module, which provides support for launching CGI programs. If this module is removed, CGI programs will not be usable. This role service is not enabled by default.

By default, IIS 7.0 does not provide any CGI programs, so they should be used only if your application uses third-party CGI programs. If it does, you should ensure that only the required CGI programs are allowed to minimize the Web server surface area.

Note You must be a server administrator to allow CGI programs.

To properly configure CGI programs, you should take the following steps:

1. **If your Web server uses CGI programs, install the CGI role service.** Without this role service, the CGI programs will not be created, and requests to resources mapped to CGI programs will return errors.
2. **If your Web server does not use CGI programs, *do not* install the CGI role service.** This eliminates the possibility of unwanted CGI programs being configured on your server.
3. **Configure the allowed CGI programs.** Each CGI program must be allowed to execute on the server before it can be used. You can use IIS Manager to configure all CGI programs that are allowed to execute on the server. This is explained in more detail later in this section. Exercise extreme caution when allowing third-party CGI programs and be sure you trust their source. Installing untrusted or buggy CGI programs can compromise the security of the Web server or negatively affect its reliability.
4. **Configure the desired handler mappings.** To use CGI programs, you need to create handler mappings that map allowed CGI programs to specific content types in your application. For more information on creating handler mappings for CGI programs, see Chapter 12. We will discuss securing handler mappings in the section titled "Configuring the Minimal Set of Handler Mappings" later in this chapter.

Similar to ISAPI extensions, you must explicitly allow any CGI program that has to execute on your server. When you allow a specific CGI program path, this CGI program can now be launched by any application on the server that configures a handler mapping to this CGI program. To be allowed, each allowed CGI program entry must specify the full path and arguments exactly the same way they are specified in each handler mapping. CGI programs are allowed in the ISAPI and CGI Restrictions feature, similar to the process described in the section titled "Enabling Only the Required ISAPI Extensions" earlier in this chapter.

Enabling Only the Required FastCGI Programs

IIS 7.0 supports hosting FastCGI programs by using the FastCGI feature, which provides a more reliable way to host many application frameworks than CGI does.

Note To enable FastCGI programs to work on IIS 7.0, the CGI role service must be installed. This role service installs the FastCgiModule module, which provides support for launching FastCGI programs. If this module is removed, FastCGI programs will not be usable. This role service is not enabled by default.

By default, IIS 7.0 does not provide any FastCGI programs, so they should be used only if your application uses third-party FastCGI programs. If so, to minimize the Web server surface area, you should ensure that only the required FastCGI programs are allowed.

Note You must be a server administrator to allow FastCGI programs.

To properly configure FastCGI programs, you should take the following steps:

1. **If your Web server uses FastCGI programs, install the CGI role service.** Without this role service, the FastCGI programs will not be usable, and requests to resources mapped to FastCGI programs will return errors.
2. **If your Web server does not use FastCGI programs, *do not* install the CGI role service.** This eliminates the possibility of unwanted FastCGI programs being configured on your server.
3. **Configure the allowed FastCGI programs.** Each FastCGI program must be allowed to execute on the server before it can be used. Though there is no IIS Manager support for configuring FastCGI programs that are allowed to execute on the server, you can do this by editing the *system.webServer/fastCgi* configuration section. For more information on configuring FastCGI programs, see Chapter 11, "Hosting Application Development Frameworks." Exercise extreme caution when allowing third-party FastCGI programs and be sure you trust their source. Installing untrusted or buggy FastCGI programs can compromise the security of the Web server or negatively affect its reliability.

4. **Configure the desired handler mappings.** To use FastCGI programs, you need to create handler mappings that map allowed FastCGI programs to specific content types in your application. For more information on creating handler mappings for FastCGI programs, see Chapter 12. We will discuss securing handler mappings in the section titled "Configuring the Minimal Set of Handler Mappings" later in this chapter.

Unlike ISAPI extensions and CGI programs, FastCGI programs are not allowed through the ISAPI and CGI Restriction feature. Instead, in the *system.webServer/fastCgi* configuration section, you need to create an entry for each allowed FastCGI program. For more information on configuring FastCGI programs, see Chapter 11.

Reducing the Application's Surface Area

Installing only the required Web server features and locking down the enabled ISAPI extensions, ISAPI filters, and CGI and FastCGI programs is a great way to reduce the surface area of the Web server as a whole. You can take it a step further by reducing the set of functionality available at the application level, by limiting the modules enabled in each application, and by constraining the set of resources that the application is configured to serve.

Enabling Only the Required Modules

When your Web server is configured to run a single dedicated application, you should install only the modules necessary to host this application. However, if your Web server hosts multiple applications, you may need to install a superset of all IIS features and third-party modules that each application requires. In this case, you can further reduce the surface area of each application by disabling at the application level any modules that the application does not need.

To do this, you can configure the set of enabled IIS modules (managed or native) for each application. You can do this by using IIS Manager: select your application node in the tree view, double-click Modules, and remove any modules that are not needed in the application. You can learn more about the process for removing modules in Chapter 12.

Caution Exercise caution when removing modules because removing security-sensitive modules that perform tasks—for example, those that perform authorization—can result in weakening application security. See the section titled "Securing Web Server Modules" in Chapter 12 for information about removing modules and which built-in IIS modules may be security-sensitive.

If you are operating a Web server on which third parties are able to publish application content, be aware that they can by default enable new managed modules that are included with their application to process requests to their application. Likewise, they can disable any module that is installed and enabled at the server level, as long as it is not locked by default.

If this is not what you want, you should consider locking the *system.webServer/modules* configuration section or using fine-grain configuration locking to lock specific modules against being removed. For more information on locking down modules, see the section titled "Locking Down Extensibility" in Chapter 12.

Note It is not possible to add new native modules at the application level. Similarly, it is not possible to remove native modules associated with IIS features at the application level by default because IIS setup locks them at the server level.

Configuring the Minimal Set of Handler Mappings

Handler mappings directly determine what resource types the Web server is configured to serve. They do this by mapping extensions or URL paths to modules or ASP.NET handlers that provide processing for the corresponding resource type. Similar to modules, handler mappings are typically installed at the server level when IIS features or third-party application frameworks are installed. This is done to enable all applications on the server to serve the associated content.

If your application does not serve specific content types or does not use specific application framework technologies installed on the Web server, you should remove the associated handler mapping entries in the *system.webServer/handlers* configuration section at the application level to prevent the Web server from attempting to use them to satisfy requests to your application. This reduces the risk of unintended script functionality executing in your application, or an application framework specific vulnerability being exploited. Note that the latter may occur even if your application does not contain any resources or scripts for a particular application framework, if the application framework contains a vulnerability that manifests before it attempts to locate the requested script.

Use the following techniques to configure the minimal set of handler mappings for your application:

- **Review the handler mappings to understand what resource types can be processed in your application.** Keep in mind that the Web server will attempt to satisfy each incoming request with the first handler mapping that matches the URL *path* and *verb* of the incoming request. Typically, the *StaticFileModule* will process all requests that have not matched other handler mappings, thus serving the requested resource as a static file if its extension is listed in the application's MIME type configuration. For more information on how handler mappings are selected, see the section titled "Adding Handler Mappings" in Chapter 12.
- **Remove any unused handler mappings in your application.** You can do this by removing the specific handler mappings. If possible, remove all handler mappings by clearing the *system.webServer/handlers* configuration section and re-adding only the handler mappings that your application uses.

- **Be aware of preconditions.** Because preconditions can be set on handler mappings to disable the use of these mappings in some application pools, some handler mappings may be ignored, resulting in the request being served using another matching handler mapping. To avoid security problems, do not create multiple handler mappings that rely on order to match similar requests.
- **Add applicable restrictions to handler mappings.** When adding new handler mappings, make use of the resource type restrictions to restrict the handler mappings only to requests that map to existing physical files or directories in your application. This can help stop malicious requests to resources that do not exist in your application. Additionally, make use of the access restrictions as described in the section titled "Setting Web Site Permissions" later in this chapter.

See Chapter 12 to learn more about creating handler mappings, how preconditions affect them, and using the resource type and access restrictions.

If you are operating a Web server on which third parties can publish application content, be aware that they can modify the handler mappings in any way to control how requests to their applications are processed. They can add new handler mappings to any enabled module, remove any of the existing handler mappings, or map requests to other handlers. If you do not want this to happen, you should consider locking the *system.webServer/handlers* configuration section. For more information on locking down the handler mappings, see the section titled "Locking Down Extensibility" in Chapter 12.

Note Handler mappings that map requests to ISAPI extensions (IsapiModule), CGI programs (CgiModule), and FastCGI programs (FastCgiModule) are further limited by the ISAPI and CGI Restrictions and FastCGI program configuration at the Web server level, which can be set only by the administrator.

Setting Web Site Permissions

Web site permissions are an additional restriction that can be placed on a Web site, application, or URL in configuration to control what requests IIS is allowed to serve. These permissions are implemented at two levels. First, each handler mapping specifies the required permission level by using the *requireAccess* attribute. If the request that matches this handler mapping is made and the required permission is not granted for the requested URL, IIS will reject the request with a 403.X response status code. Second, some IIS components have hard-coded requirements for certain permissions, and they will reject the request if they are processing a request to a URL that does not have this permission.

Note Web site permissions control what functionality is enabled to be used at a particular URL. They do not consider the identity of the requesting user and therefore cannot be used to replace IIS authorization schemes when implementing access control.

Table 14-2 indicates the permission types that can be granted for a particular URL.

Table 14-2 Permission Types Granted for URLs

Permission	Description
None	No permissions are granted.
Read	Read access to files and directories is enabled. In particular, this enables the following: static file handler serving static files, directory listings, and default documents.
Script	Script processing is enabled. In particular, this enables the following: ISAPI extensions, CGI programs, FastCGI programs, and ASP.NET handlers. ISAPI extensions and CGI programs must specify a fixed script processor.
Execute	Running executables is enabled. In particular, this enables the following: ISAPI extensions and CGI programs with no script processor set (that execute the file provided in the request path). If granted, this permission will by default lead to IIS trying to execute EXE files as CGI applications and load DLL files as ISAPI extensions instead of downloading them.
Source	In previous versions of IIS, this permission enables WebDav requests to access the source of script files. No special handling of this permission is present in IIS 7.0.
Write	In previous versions of IIS, this permission enables WebDav requests to write files. No special handling of this permission is present in IIS 7.0.
NoRemoteRead	Prevents remote requests from using the Read permission.
NoRemoteScript	Prevents remote requests from using the Script permission.
NoRemoteExecute	Prevents remote requests from using the Execute permission.
NoRemoteWrite	Prevents remote requests from using the Write permission.

In IIS Manager, you can set the permission granted for a particular Web site, application, or URL by selecting the appropriate node in the tree view and then clicking Handler Mappings. There, you can set the Read, Script, and Execute permissions by clicking Edit Permissions in the Actions pane. Doing this also automatically shows the handlers that require a permission that is not granted as disabled, to let you know that requests to these handlers will be rejected.

You can set the permissions directly in configuration by editing the *accessPolicy* attribute of the *system.web/handlers* configuration section or by using Appcmd or other configuration APIs to do it. For example, to grant only the Read permission to the /files subfolder of the Default Web Site, you can use the following Appcmd syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config "Default Web Site/files"
/section:handlers /accessPolicy:Read
```

Note When you specify a configuration path to apply configuration to a specific Web site or URL, you may get an error indicating that the configuration is locked. This is because most security configuration sections, including all authentication sections, are locked at the Web server level to prevent delegated configuration. You can unlock these sections to allow delegated configuration, or you can persist the configuration to applicationHost.config by using the */commit:apphost* parameter with your Appcmd commands.

Use the following guidelines when setting Web site permissions:

- **Remove unnecessary permissions for URLs that do not require them.** By default, Read and Script permissions are granted. For URLs that do not require the ability to execute dynamic application technologies, remove the Script permission. Do not grant additional permissions such as Execute unless necessary.
- **Keep in mind that applications can configure handler mappings that do not require permissions.** By default, applications can change existing handler mappings or create new handler mappings to not require permissions. Because of this, do not rely on Web site permissions for controlling which handler mappings can or cannot be created by applications that use delegated configuration. The permissions are only guaranteed for built-in IIS features including the static file handler, IsapiModule, CgiModule, and ASP.NET handlers, which hardcode the permission requirements. In other cases, the permissions are guaranteed only if the *system.webServer/handlers* configuration section is locked and prevents changes to the handler mappings set by the Web server administrator. For more information about locking down the handlers configuration section, see the section titled "Locking Down Extensibility" in Chapter 12.

Note Unlike in IIS 6.0, wildcard handler mappings no longer ignore Web site permissions. In IIS 7.0, they require the same level of permissions as they would when mapped with nonwildcard handler mappings. Because of this, configurations in which a wildcard-mapped ISAPI extension is used for URLs that do not allow the Script permission will now be broken and require the Script permission to be granted.

Configuring Minimal Sets of MIME Types

By default, to serve the corresponding physical file to the client, IIS handler mappings are preconfigured to direct all requests not mapped to other modules to the *StaticFileModule* (if the file does not exist, a 404 error response code is returned).

Note In IIS 7.0, the MIME types configured by default have been upgraded to contain many of the new common file extensions.

For security reasons, the *StaticFileModule* will serve only files whose extensions are listed in the MIME type configuration. This behavior is extremely important, because otherwise

applications that contain scripts and other content that is processed by application framework technologies may end up serving these resources directly if the appropriate application framework handler mappings are not installed or become removed. In this situation, the MIME type configuration protects these resources from being served to the client and results in a 404.3 error returned to the client.

Note You can learn about configuring MIME types in the section titled "Enabling New Static File Extensions to Be Served" in Chapter 11.

The default list of MIME types in IIS 7.0 should be safe for most applications. You can further configure the MIME types at the server level—or for a Web site, application, or URL—to mandate which file extensions are servable by the StaticFileModule. By reducing this list to only the extensions of the files known to be safe to serve, you can avoid accidentally serving files that are part of an application and are not meant to be downloaded.

Caution MIME type configuration prevents only unlisted files from being downloaded directly through the StaticFileModule. It does not protect the resources from being accessed through the application, nor does it protect them from being downloaded if they are mapped to custom handlers. To protect application resources that are not meant to be accessed, you should forbid their extensions or use Request Filtering to place the content in a directory that is configured as a hidden segment. For more information, see the section titled "Request Filtering" later in this chapter.

You should use the following guidelines to securely configure the MIME types list:

- **Do not add file extensions to the MIME types configuration that are not meant to be downloaded directly.** This refers to any of the file types that are used by the application, such as ASP, ASPX, or MDB.
- **Configure a minimal set of MIME types for each application.** If possible, configure the MIME types for each application to contain only the minimal set of extensions. This can help prevent accidental serving of new files when they are added to the application. For example, if your application uses XML files to store internal data, you should make sure that your application does not include *xml* in its MIME type configuration even though the .xml extension is listed there by default when IIS is installed.

Configuring Applications for Least Privilege

Next to reducing its surface area, the most effective strategy to reduce the risk of a successful attack on your Web server is to configure your applications to run with the least privilege possible. Doing this minimizes the amount of damage that results if an attacker successfully exploits any known or future vulnerability. Similar to reducing the surface area, this technique

is not limited to blocking specific threats—it works well for any threat that may be present in your application today or that may be found in the future.

The key to reducing the privilege of the application code in the IIS environment is to understand the identity under which the code executes, select the identity with the minimal number of privileges required, and limit the rights of the identity to access server resources. To help achieve least privilege, we will review these techniques:

- Use a low privilege application pool identity
- Set NTFS file system (NTFS) permissions to grant minimal access
- Reduce trust of ASP.NET applications
- Isolate applications

These techniques are discussed next in this section.

Use a Low Privilege Application Pool Identity

The majority of code executed as part of a Web application is executed in the context of the IIS worker process and typically runs under the identity configured for the application pool. Therefore, using a least privilege application pool identity is the primary way to constrain the privileges and rights granted to the application code.

By default, IIS application pools are configured to run using the built-in Network Service account, which has limited rights on the Web server. When each IIS worker process is started, it also automatically receives membership in the IIS_IUSRS group. This group replaces the IIS_WPG group used in IIS 6.0 as the required group identifying all IIS worker processes on the computer. IIS setup may still create the IIS_WPG group for backward-compatibility reasons, in which case IIS_IUSRS will be made a member of this group.

In addition, certain code in your application may execute with the identity of the authenticated user associated with each request. Table 14-3 summarizes the identities that may be used in your application.

Table 14-3 Application Identities

Identity Type	Used When...	Identities
Application pool identity	■ Accessing all files necessary for the execution of the IIS worker process ■ Accessing web.config files ■ Running FastCGI applications (by default) ■ Running ASP.NET applications (by default)	■ *Network Service* by default; otherwise configured application pool identity ■ *IIS_IUSRS* group ■ Application Pool SID (*IIS APPPOOL\<ApppoolName>*)

Table 14-3 Application Identities

Identity Type	Used When...	Identities
Authenticated user	■ Accessing static files ■ Running ISAPI extensions ■ Running CGI programs (by default) ■ Running FastCGI applications (if impersonation is enabled) ■ Running ASP.NET applications (if impersonation is enabled)	■ *IUSR* by default when using anonymous authentication; otherwise configured anonymous user or application pool identity ■ Authenticated user if Windows token authentication methods are used
Virtual directory fixed credentials (when configured)	■ Accessing all application content	■ The configured virtual directory credentials

When using authentication schemes that produce Windows tokens, such as Windows Authentication or Basic Authentication, be aware that when highly privileged users access your application, it will execute with higher privileges than intended. Therefore, it is recommended that you do not allow users that have administrative privileges on the server to access your application. For more information on what identity each application framework executes under, see Chapter 11.

Caution When using authentication schemes that produce Windows identities, your applications may execute with the identity of the authenticated user.

Also, when using anonymous authentication, you may opt for configuring the anonymous user to be the application pool identity, to ensure that all code always executes under the application pool identity. This makes it significantly easier to manage the access rights of the worker process. You can learn about configuring this in the section titled "Anonymous Authentication" later in this chapter.

Note To simplify access management, configure the anonymous authentication user to be the application pool identity.

When selecting the application pool identity for your applications, use the following guidelines to maintain or improve the security of your Web server:

- **Ensure that the application pool identity is not granted sensitive privileges or unnecessary rights to access resources.** Often, in the face of "access denied" errors, administrators tend to grant the application pool identity full or otherwise unnecessary access to resources. This increases the privilege of the worker process and increases the risk of a

serious compromise if the code in the worker process is compromised. Only grant the worker process the minimal access needed for the application to work. If this minimal access involves privileges or rights typically associated with administrative users, you need to re-evaluate your application's design.

- **Do not use highly privileged or administrative identities for IIS application pools.** Never use LocalSystem, Administrator, or any other highly privileged account as an application pool identity. Just say no!
- **Consider using a lower privilege identity.** If your application allows it, consider using a custom low privilege account for the IIS worker process. Unlike IIS 6.0, IIS 7.0 automatically injects the new IIS_IUSRS group into the worker process, eliminating the need for you to make the new identity a member of any group.
- **Separate code with different privilege requirements into different application pools.** If your server has multiple applications that require different levels of privilege (for example, one requires the privilege to write to the Web application, and the other one doesn't), separate them into two different application pools.
- **When using anonymous authentication, configure the anonymous user to be the application pool identity.** This significantly simplifies configuring access rights for your application by making the application code always execute with the application pool identity.
- **Grant minimal access.** When granting access to the application pool identity, grant the minimal access necessary. You can use this in conjunction with separating applications into different application pools to maintain least privilege for your applications. To grant access to a resource for all IIS application pools, grant it to the IIS_IUSRS group. To grant access to a resource for a specific application pool, use the unique application pool identity. Alternatively, use the automatic Application Pool SID that is named IIS APPPOOL\<*ApppoolName*> (the latter does not work for UNC content, only local content). Do not grant access rights to Network Service because it grants access to all services running on the server under the Network Service identity.

Set NTFS Permissions to Grant Minimal Access

By default, all files required for IIS worker processes to function grant access to the IIS_IUSRS group, which ensures that IIS worker processes can function regardless of the selected application pool identity. However, it is up to you to grant access to the application content so that the Web server and the application can successfully access its resources. Additionally, it is up to you to grant access to the additional resources the IIS worker process uses, such as ISAPI extensions, CGI programs, or custom directories configured for logging or failed request tracing.

Table 14-4 indicates the level of access the Web server typically requires for different kinds of resources.

Table 14-4 Access Levels for Web Server Resources

Resource Type	Identity	Required Access
Content (virtual directory physical path and below)	■ Fixed credentials set on the virtual directory (if set) OR ■ Authenticated users ■ IIS worker process identity (application pool identity)	■ Read ■ Write, if your application requires being able to write files in the virtual directory (granting Write is not recommended)
Additional resources IIS features use: CGI programs, ISAPI extensions, native module dynamic-link libraries (DLLs), compression directory, failed request tracing directory, logging directory, and more	■ IIS worker process identity (application pool identity)	■ Read ■ Execute, for CGI programs ■ Write, for compression or logging directories, or whenever the Web server needs to write data

When granting access to content directories, you can use one of the following techniques:

- **Grant access to IIS_IUSRS.** This enables all IIS worker processes to access the content when using the application pool identity, or when using anonymous authentication. However, this does not enable you to isolate multiple application pools. If using a Windows-based authentication scheme, you also will need to grant access to all of the authenticated users that use your application.
- **Grant access to the identity of the application's application pool.** This will enable only the IIS worker processes running in the application pool with the configured identity to access the content. If using anonymous authentication, you additionally need to set the anonymous user to be the application pool identity. If using a Windows-based authentication scheme, you also will need to grant access to all of the authenticated users that use your application. This approach is the basis for application pool isolation. For more information, see the section titled "Isolating Applications" later in this chapter.
- **Configure fixed credentials for the application's virtual directory and grant access to these credentials.** This will prompt the IIS worker process to access the content by using the fixed credentials, regardless of the authenticated user identity. This option is often used when granting access to remote UNC shares to avoid the difficulties of ensuring that authenticated user identities can be delegated to access the remote network share. It can also be an efficient way to manage access to the content for a single identity regardless of the authenticated user (which can be set to the application pool identity when using custom application pool identities). Finally, it can be used to control access to the application when you host multiple applications inside the same application pool.

Note If you are using IIS Manager to administer the application remotely, you will also need to grant Read access to the NT Service\WMSvc account. For more information, see Chapter 8, "Remote Administration."

If you are using an authentication scheme (other than anonymous authentication) that produces Windows identities for authentication users, such as most of the IIS authentication schemes, you will also need to make sure that all authenticated users that require the use of your application have access to its content. This is because the Web server will use the authenticated user identity to access application content. Also, many application frameworks will by default impersonate the authenticated user when executing application code and accessing application resources.

When you need to allow multiple Windows users to use the application, you should add all of these users to a specific group and grant this group access to the application content. Alternatively, when using the fixed credentials model, you do not have to grant access to the authenticated users. Instead, IIS and application code will always impersonate the fixed virtual directory credentials. For more information on setting up the fixed credentials model, see the section titled "Managing Remote Content" in Chapter 9, "Managing Web Sites."

When your content is on a UNC share, you will likely need to use the fixed credentials model because most IIS authentication schemes do not produce Windows tokens that can be used for remote network shares (with the exception of basic authentication and IIS client certificate mapping authentication). Alternatively, you can configure your Web server for Constrained Delegation and Protocol Transition to allow the authenticated user tokens to be used against the remote share. However, using the fixed credentials for virtual directories on UNC shares is significantly easier to configure, so it is recommended over setting up delegation. For more information, see the section titled "UNC Authentication" later in this chapter.

Note Unlike in IIS 6.0, in which the authenticated user having access to the content directory is typically sufficient (except for ASP.NET applications), IIS 7.0 also requires the IIS worker process identity to have access to the content directories before they can read the web.config configuration files. This happens before IIS determines the authenticated user.

Reduce Trust of ASP.NET Applications

In addition to constraining the execution rights of the application by using a low privilege application pool identity, you can further sandbox the .NET parts of your application by using the ASP.NET trust levels. The ASP.NET parts of the application include the ASP.NET applications themselves, as well as managed modules that provide services for any application in ASP.NET Integrated mode applications.

ASP.NET trust levels use the Code Access Security (CAS) infrastructure in the .NET Framework to limit the execution of the application code, by defining a set of code permissions that control what application code can and cannot do. By default, ASP.NET applications and managed modules execute using the Full trust level, which does not limit their execution. In this trust level, the application can perform any action that is allowed by the Windows privileges and resource access rights.

You can reduce the trust level of ASP.NET applications to limit their execution further. This can be an effective way to achieve lower privilege for your application. By default, you have options described in Table 14-5, which are defined by the ASP.NET trust policy files.

Table 14-5 Default Trust Level Options

.NET Trust Level	Execution Limits	Rights
Full (internal)	None	All
High (web_hightrust.config)	None/.NET	Can do anything except call native code
Medium (web_mediumtrust.config)	Application is trusted within its own scope, but should not be able to affect other applications or the rest of the machine	■ Access files in the application root ■ Connect to SQL and OLEDB databases ■ Connect to Web services on the local machine ■ Manipulate threads and execution for its own requests
Low (web_lowtrust.config)	Application is not highly trusted; meant for applications that can use built-in ASP.NET features but do not run custom code	Only read access within application root
Minimal (web_minimaltrust.config)	Application is untrusted; ability to use built-in ASP.NET features is extremely restricted	Minimal permissions for executing code

It is recommended that you run ASP.NET applications by using the Medium trust level. In this trust level, the application is not able to access resources outside of itself and cannot perform operations that can compromise the security of the Web server overall. However, if you do this, you should test the application to make sure that it does not experience any security exceptions due to the lack of required permissions. You may also want to performance test the application to make sure that using the reduced trust level does not negatively impact your application's performance.

Note The Medium trust level is the recommended trust level to constrain the execution of ASP.NET applications and managed modules, and to host multiple applications on a shared Web server.

You can use IIS Manager to configure the trust level used for ASP.NET applications and managed modules by double-clicking .NET Trust Levels. You can do this for the entire Web server—or for a specific application—by selecting that application node prior to using the .NET Trust Levels feature. You can also set the trust level directly by changing the *level* attribute in the *system.web/trust* configuration section.

Caution The *system.web/trust* configuration section is not locked by default, which means that any application can configure its own trust level. If you don't want this, lock the configuration section at the server level.

Isolating Applications

Application pools provide a great way to isolate multiple applications running on the same machine, both in terms of availability and security. This provides the following benefits:

- Failures, instability, and performance degradation experienced in one application do not affect the applications in a different application pool.
- Applications running in a different application pool can restrict access to their resources to that application pool only, preventing other applications from being able to access their resources.

The recommended way to configure applications for isolation is to place each application into a separate application pool. When you do this, IIS 7.0 makes it easy to isolate applications by automatically injecting a unique application pool security identifier, called the application pool SID, into the IIS worker process of each application pool. Each application pool SID is generated based on the application pool name and has the name *IIS APPPOOL\ <Apppool-Name>*. The application pool SID makes it possible to quickly isolate applications by placing NTFS permissions on their content to grant access only to the application pool SID of the application's application pool.

Note You can quickly isolate applications by setting permissions on their content to allow only the Application Pool SID of the corresponding application pool.

To make Application Pool SID–based isolation effective, you need to do the following:

1. Configure anonymous authentication to use the application pool identity.
2. Grant access to the application content for the *IIS APPPOOL\ <ApppoolName>* SID.
3. Do not grant access to the application content to IIS_IUSRS, IIS_WPG or any other application pool identity that may be used by another application pool.

4. Configure separate locations for all temporary and utility directories that IIS and the application use for each application or application pool, and set permissions on them to allow access only for the *IIS APPPOOL\ <ApppoolName>* SID.

Table 14-6 shows some of the common directories that IIS and ASP.NET applications use. The directories must be configured for isolation for each Web site or application and receive the appropriate permissions to enable access only by the associated application pool.

Table 14-6 Common Directories Used by IIS and ASP.NET Applications

Directory	Configured In...
Content directories	Virtual directory physical path
Windows TEMP directory (*%TEMP%* or *%TMP%*): used by Windows components	Set the *loadUserProfile* attribute to *true* in the *processModel* element of each application pool. This causes the TEMP directory to point to *%SystemDrive%\Users\%UserName%\AppData\Local\Temp*.
Web site log file directory	*directory* attribute of the *logFile* element for each site. The default is *%SystemDrive%\Inetpub\Logs\LogFiles*.
Web site Failed Requests Logs directory	*directory* attribute of the *traceFailedRequestsLogging* element of each site. The default is *%SystemDrive%\Inetpub\Logs\FailedReqLogFiles*.
IIS Static Compression directory	Isolated automatically by creating a subdirectory for each application pool and applying ACLs to each directory for the Application Pool SID.
ASP.NET Temp directory: used by ASP.NET compilation system	*tempDirectory* attribute in *system.web/compilation* section for each application. The default is *%SystemRoot%\Microsoft.NET\Framework\<version>\Temporary ASP.NET Files*.
ASP Template Disk Cache directory	Isolated automatically by applying ACLs to each file for the Application Pool SID.

Note The Application Pool SIDs can be used only for isolating local content. If you are using content located on a UNC share, you need to either use a custom application pool identity or configure fixed credentials for each virtual directory. Then you should use that identity to grant access to the network share.

IIS 7.0 provides automatic isolation of the server-level configuration by using configuration isolation. No action is necessary to enable this, because it is done by default. For more information about configuration isolation, see the section titled "Understanding Configuration Isolation" later in this chapter.

Note The server-level configuration in applicationHost.config is isolated automatically using configuration isolation.

However, .NET Framework configuration in the machine.config and root web.config files—as well as the configuration in the distributed web.config files that are part of the Web site's directory structure—are not isolated. To properly isolate the distributed web.config files, set the appropriate permissions on the content directories, as described earlier in this section.

Implementing Access Control

Web applications require the ability to restrict access to their content, to protect sensitive resources, or to authorize access to resources to specific users. IIS 7.0 provides an extensive set of features that you can use to control the access to application content. These features are logically divided into two categories, based on the role they play in the process of determining access to the request resource:

- **Authentication** Authentication features serve to determine the identity of the client making the request, which can be used in determining whether this client should be granted access.
- **Authorization** Authorization features use the authenticated identity on the request or other applicable information to determine whether or not the client should be granted access to the requested resource. Authorization features typically depend on the presence of authentication features to determine the authenticated identity. However, some authorization features determine access based on other aspects of the request or the resource being requested, such as Request Filtering.

IIS 7.0 supports most of the authentication and authorization features available in IIS 6.0, and it introduces several additional features. These features (role services) are listed here in the order in which they apply during the processing of the request:

1. **IP and Domain Restrictions.** Used to restrict access to requests clients make from specific IP address ranges or domain names. The default install does not use this feature.
2. **Request Filtering.** Similar to UrlScan in previous versions of IIS, request filtering is used to restrict access to requests that meet established limits and do not contain known malicious patterns. In addition, Request Filtering is used to restrict access to known application content that is not meant to be served to remote clients. Request filtering is part of the default IIS 7.0 install and is configured to filter requests by default.
3. **Authentication features.** IIS 7.0 offers multiple authentication features that you can use to determine the identity of the client making the request. These include Basic Authentication, Digest Authentication, Windows Authentication (NTLM and Kerberos), and

many others. The Anonymous Authentication feature is part of the default IIS install and is enabled by default.

4. **Authorization features.** IIS 7.0 provides a new URL Authorization feature that you can use to create declarative access control rules in configuration to grant access to specific users or roles. In addition, it continues to support NTFS ACL-based authorization for authentication schemes that yield Windows user identities. IIS uses NTFS ACL-based authorization by default.

Note In IIS 7.0, all of these role services are available as Web server modules that can be individually installed and uninstalled and optionally disabled and enabled for each application. Be careful when removing authentication, authorization, and other access control modules, because you may unintentionally open access to unauthorized users or make your application vulnerable to malicious requests. To review the list of security-sensitive modules that ship with IIS 7.0, and considerations when removing them, see the section titled "Securing Web Server Modules" in Chapter 12.

You should leverage access control features to ensure that only users with the right to access those resources can access them. To do this, you need to configure the right authentication and authorization features for your application.

In addition, you should take advantage of Request Filtering to limit usage of the application as much as is possible, by creating restrictions on content types, URLs, and other request parameters. Doing so enables you to preemptively protect the application from unexpected usage and unknown exploits in the future.

IP and Domain Restrictions

The IP and Domain Restrictions role service enables you to restrict access to your application to clients making requests from a specific IP address range or to clients associated with a specific domain name. This feature is largely unchanged from IIS 6.0.

Note The IP and Domain Restrictions role service is not part of the default IIS install. You can manually install it from the IIS \ Security feature category in Windows Setup on Windows Vista, or from the Security category of the Web Server (IIS) role in Server Manager on Windows Server 2008. See Chapter 12 for more information about installing and enabling modules.

You can use this feature to allow or deny access to a specific range of IP addresses, or to a specific domain name. The IP and Domain Restrictions role service will attempt to match the source IP address of each incoming request to the configured rules, in the order in which rules are specified in configuration. If a matching rule is found, and the rule is denied access to the request, the request will be rejected with a 403.6 HTTP error code. If the rule allows access, the request will continue processing (all additional rules will be ignored).

You can specify any number of allow or deny rules and indicate whether access should be granted or denied if no rules match. The common strategies for using IP Address and Domain Restrictions rules include:

- Denying access by default and creating an Allow rule to grant access only to a specific IPv4 address range, such as the local subnet. You can do this to grant access only to clients on the local network or to a specific remote IP address.
- Allowing access by default and creating a Deny rule to deny access to a specific IP address or IPv4 address range. You can do this to deny access to a specific IP address that you know is malicious.

Caution Allowing access by default and denying access for specific IP address ranges is not a secure technique, because attackers can and often will use different IP addresses to access your application. Also, clients that use IPv6 addresses instead of the IP addresses will not match a Deny rule that uses an IPv4 address range.

If you are looking to restrict access to your application to clients on the local network, you may be able to implement an additional defense measure by specifying that your site binding should listen only on the IP address associated with the private network. For servers that have both private and public IP addresses, this can restrict requests to your site to the private network only. You should use this in conjunction with IP and Domain Restrictions where possible. For more information on creating Web site bindings, see Chapter 9.

Though you can create rules that use a domain name instead of specifying an IP address, we don't recommend that you do so. This is because domain name-based restrictions require a reverse Domain Name System (DNS) lookup of the client IP address for each request, which can have a significant negative performance impact on your server. By default, the feature does not enable the use of domain name–based rules.

To configure the IP and Domain Restrictions rules, you will need to perform the following steps:

1. Use IIS Manager to configure the rules by selecting the Server node, a Site node, or any node under the site in the tree view. Then double-click the IP Address And Domain Restrictions feature, which is shown in Figure 14-4.
2. Use the Add Allow Entry or the Add Deny Entry command in the Actions pane to create allow or deny rules.
3. You can also use the Edit Feature Settings command to configure the default access (allow or deny) and whether or not domain name–based rules are allowed.

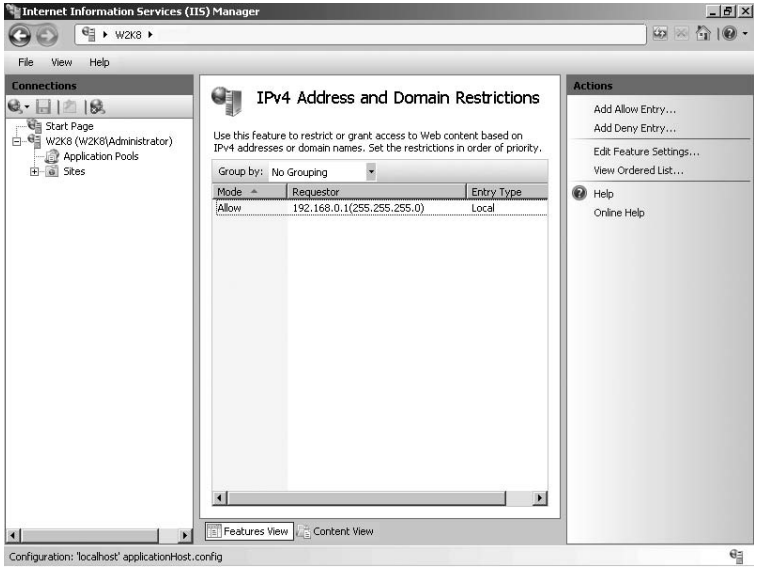


Figure 14-4 Configuring IP and Domain Restrictions using IIS Manager.

Note Although the IP and Domain Restrictions feature enables you to use IPv6 addresses, you cannot configure addresses that use IPv6 rules using IIS Manager. Also, requests that are made over IPv6 connections do not match rules using IPv4 addresses. Likewise, requests made over IPv4 connections do not match rules that specify IPv6 addresses.

Note By default, the *ipSecurity* configuration section is locked at the server level. You can unlock this section by using the *Appcmd Unlock Config* command to specify IP and Domain Restriction rules in web.config files at the site, application, or URL level.

You can also configure the IP and Domain Restrictions configuration by using Appcmd or configuration APIs to edit the *system.webServer/security/ipSecurity* configuration section.

Request Filtering

The Request Filtering feature is an improved version of the UrlScan tool available for previous versions of IIS. The Request Filtering feature enforces limitations on the format of the request URL and its contents to protect the application from possible exploits that may arise from exceeding these limits. With IIS 6.0 and previous versions of IIS, these limits have thwarted the majority of known Web application exploits, such as application-specific buffer overruns resulting from long malicious URLs and query strings. Though the Web server itself, starting with IIS 6.0, has been engineered to not be vulnerable to these attacks, these limits remain

valuable in preventing both known and unknown future exploits for the Web server and applications running on it.

In addition to enforcing request limits, the Request Filtering feature also serves to deny access to certain application resources that are not meant to be served to Web clients, but are located in the servable namespace for an application. These files include the web.config configuration files for IIS and ASP.NET, as well as contents of the /BIN and /App_Code directories for ASP.NET applications.

You can use Request Filtering to do the following:

- **Set request limits** Configure limits on the length and encoding of the URL, the length of the query string, the length of the request POST entity body, allowed request verbs, and maximum lengths of individual request headers.
- **Configure allowed extensions** Configure which file extensions are allowed or rejected, regardless of the selected handler mapping.
- **Configure hidden URL segments** Configure which URL segments are not served, to hide parts of your URL hierarchy.
- **Configure denied URL sequences** Configure which URL patterns are not allowed, possibly to prevent known exploits that use specific URL patterns.

IIS 7.0 depends on request filtering by default to reject requests that may contain malicious payloads and to protect certain content from being served by the Web server. You can also use it to further restrict the input to your application, or to protect additional URLs, directories, and files or file extensions from being served by the Web server.

Caution Request filtering is a critical security component and should not be removed unless it is absolutely clear that it is not needed. Always prefer to relax request filtering limits by setting the configuration rather than uninstalling or removing the Request Filtering module.

The request filtering configuration does not have an associated IIS Manager page, so these settings cannot be set through IIS Manager. To set them, you can configure the *system.webServer/security/requestFiltering* configuration section directly at the command line by using Appcmd.exe or other configuration APIs. The *requestFiltering* section is unlocked by default, so you can set request filtering configuration in web.config files at the site, application, or URL level.

The remainder of this section will illustrate how to modify common request filtering configuration tasks.

Setting Request Limits

You can use Request Filtering to configure even tighter request limits if allowable in your application's usage to further reduce attackers' ability to exploit your application with

malicious input. At other times, you may need to relax the default request limits to allow your application to function correctly, for example, if your application uses long query strings that may exceed the default limit of 2048 characters.

You may need to modify request limits applied by request filtering if any of the default limits interfere with your application usage.

You can use Appcmd to set request filtering limits as follows.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/requestFiltering
[/allowDoubleEscaping:bool] [/allowHighBitCharacters:bool]
[/requestLimits.maxAllowedContentLength:uint] [/requestLimits.maxUrl:uint]
[/requestLimits.maxQueryString:uint]
```

This command uses the parameters in Table 14-7.

Table 14-7 Parameters for Requesting Filtering Limits

Parameter	Description
ConfigurationPath	The configuration path at which to set this configuration.
allowDoubleEscaping	Whether or not request URLs that contain double-encoded characters are allowed. Attackers sometimes use double-encoded URLs to exploit canonicalization vulnerabilities in authorization code. After two normalization attempts, if the URL is not the same as it was after one, the request is rejected with the 404.11 response status code. The default is *false*.
allowHighBitCharacters	Whether or not non-ASCII characters are allowed in the URL. If a high bit character is encountered in the URL, the request is rejected with the 404.12 response status code. The default is *true*.
requestLimits.maxAllowed-ContentLength	The maximum length of the request entity body (in bytes). If this limit is exceeded, the request is rejected with the 404.13 response status code. The default is *30000000* bytes (approximately 28 megabytes).
requestLimits.maxUrl	The maximum length of the request URL's absolute path (in characters). If this limit is exceeded, the request is rejected with the 404.14 response status code. The default is *4096* characters.
requestLimits.maxQueryString	The maximum length of the query string (in characters). If this limit is exceeded, the request is rejected with the 404.15 response status code. The default is *2048* characters.

Note You can also configure the request header length limits for each header by adding header limits in the *headerLimits* collection in the *system.webServer/security/requestFiltering* section. If the request exceeds a configured header limit, it will be rejected with a 404.10 response status code. Additionally, you can configure which verbs are allowed by adding those verbs to the *verbs* collection. If the request specifies a verb that is not allowed, it is rejected with a 404.6 response status code.

Configuring Allowed Extensions

By default, the Web server will process only extensions for which a handler mapping exists. By default, StaticFileModule processes all unmapped extensions. StaticFileModule serves only extensions listed in the *system.webServer/staticContent* configuration section (known as Mimemaps in IIS 6.0 and previous versions of IIS). Therefore, the handler mapping configuration and the static content configuration serves as the two-level mechanism for controlling which extensions can be served for a particular URL.

However, as a defense in depth measure, you may still want to use Request Filtering to deny requests to a particular extension, after making sure that it is not configured in the IIS handler mappings and the static content list. This makes sure that requests to this extension are rejected very early in the request processing pipeline, much before they otherwise would be rejected by the configuration mentioned earlier.

To add a prohibited or explicitly allowed extension by using Appcmd, use the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/requestFiltering
/+fileExtensions.[fileExtension='string',allowed='bool']
```

The fileExtension string is in the format of .extension.

To delete a prohibited extension by using Appcmd, use the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/requestFiltering
/-fileExtensions.[fileExtension='string']
```

These commands use the parameters in Table 14-8.

Table 14-8 Parameters for Deleting Prohibited Extension

Parameter	Description
ConfigurationPath	The configuration path at which to set this configuration.
fileExtension	The extension, in the format of *.extension*, that should be allowed or denied.
allowed	Whether or not the extension is allowed or denied.

You can set this configuration for a particular site, application, or URL by specifying the configuration path with commands shown earlier in this section.

Note Alternatively, you can configure the *fileExtensions* collection to deny all unlisted extensions, explicitly enabling the extensions that are allowed. This may be a more effective practice for reducing surface area than prohibiting specific extensions. However, it requires you to know and maintain the exhaustive list of all allowed extensions necessary for your application. To do this, set the *allowUnlisted* attribute on the *fileExtensions* collection at the desired configuration path to *false*.

Configuring Hidden URL Segments

You may also want to prohibit a URL segment from being servable. ASP.NET uses this technique to prohibit requests to the /BIN, /App_Code, and other special /App_xxx directories that contain ASP.NET application resources that the application is not to serve. You can create your own special directories that contain nonservable content and protect them with URL segments. For example, to prevent content from being served from all directories named data, you can create a hidden segment named data.

Direct from the Source: Protecting ASP.NET Special Directories

The debate on how to protect special directories containing application content not meant to be served directly, such as the ASP.NET /BIN directory, dates back to IIS 5.0 days. The ASP.NET team has gone through multiple attempts at achieving this, starting with the explicit removal of Read permissions from the /BIN directory on IIS 5.0 during ASP.NET application startup, to adding the code that prohibited requests containing the /BIN segment to the ASP.NET ISAPI filter.

During the development of ASP.NET 2.0, I had the opportunity to guide the solution to this problem for the several new directories introduced by ASP.NET 2.0, such as the new App_Code directory. Unfortunately, removing the Read permissions from these directories was no longer an option, because on IIS 6.0, ASP.NET no longer had Write access to the IIS metabase. In the absence of a general Web server feature to protect special directories, we had no better option than to add blocking support for all new directories to the ISAPI filter. Arguably, the most painful part of the project was the decision to ask the community for the preferred naming of these special directories, which resulted in much debate and the changing of the directory names from Application_Code, to App$_Code, and finally to App_Code. At the end, we were so fed up with changing the names that the development manager ordered hats for all of us that were printed with "App$."

IIS 7.0 finally provides a general solution for protecting special directories, a solution that ASP.NET leverages to protect its special directories when installed. Therefore, it no longer relies on the ISAPI filter for this support. Additionally, hidden URL segments provide a general mechanism for anyone to configure protected directories as appropriate for their applications, without writing special code to perform the blocking.

Mike Volodarsky

IIS Core Program Manager, Microsoft

To add a hidden URL segment by using Appcmd, use the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/requestFiltering
/+hiddenSegments.[segment='string']
```

The segment string is the segment to protect.

To delete a hidden URL segment by using Appcmd, use the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/requestFiltering
/-hiddenSegments.[segment='string']
```

Note Unlike file extensions, there is no way to prohibit all segments other than the ones configured. You can only deny specific segments by adding them using the preceding command.

You can target configuration for a particular site, application, or URL by specifying the configuration path with the preceding commands.

Configuring Denied URL Sequences

In some cases, you may want to reject requests that contain specific sequences in the URL, whether or not they are a complete segment. If it is not possible to fix the application itself, this may be an effective way to protect an application from certain URL patterns that are known to cause issues.

To add a denied URL sequence by using Appcmd, use the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/requestFiltering
/+denyURLSequences.[sequence='string']
```

The sequence string is the sequence to reject.

To delete a denied URL sequence by using Appcmd, use the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/requestFiltering
/-denyURLSequences.[sequence='string']
```

Note Unlike file extensions, there is no way to prohibit all URL sequences other than the ones configured. You can only deny specific sequences by adding them using the preceding command.

You can set this configuration for a particular site, application, or URL by specifying the configuration path with the preceding commands.

Authorization

As mentioned earlier, authorization is the second phase in the process of determining whether or not a client has the right to issue a particular request. Authorization refers to determining if the user identity determined during the authentication phase is allowed to access the requested resource.

IIS 7.0 provides several authorization mechanisms that can be leveraged to control access to resources:

- **NTFS ACL-based authorization** By default, IIS 7.0 verifies that the authenticated user identity has the right to access the physical file or folder corresponding to the requested URL. This check is performed only for requests that map to physical files or directories and use authentication methods that produce Windows tokens. This authorization mechanism has multiple usage limitations that are discussed in detail later in this section.
- **URL Authorization** The new IIS 7.0 URL Authorization role service enables applications to create declarative configuration-based rules to determine which authenticated users and/or roles have the right to access the Web site or specific URLs therein. This feature replaces the IIS 6.0 URL Authorization feature, which is no longer supported.
- **ASP.NET URL Authorization** The ASP.NET URL Authorization feature, available since ASP.NET 1.0, is similar to IIS URL Authorization, with a slightly different configuration syntax and rule processing behavior. ASP.NET applications that use this feature today can configure it to control access to all Web site content when they run using ASP.NET integrated mode.

In addition, developers can provide custom authorization features by developing modules by using the IIS 7.0 native module APIs or the ASP.NET APIs for applications using ASP.NET Integrated mode. In fact, you can use most existing ASP.NET authorization modules immediately in applications that are using ASP.NET Integrated mode. This makes it significantly easier to develop custom authorization features that implement business authorization rules and can leverage the powerful ASP.NET membership and role infrastructures. For more information on installing and leveraging custom modules, see Chapter 12.

Caution Exercise extreme caution when configuring or removing authorization modules. If you remove an authorization module that is used to restrict access to the application, parts of the application may become exposed to unauthorized users. See the section titled "Securing Web Server Modules" in Chapter 12 for more information about removing security-sensitive modules.

In the remainder of this section, we will review the authorization features in detail.

NTFS ACL-based Authorization

The IIS 7.0 server engine (rather than a module) automatically performs NTFS ACL-based authorization. During this authorization, the Web server checks that the authenticated user identity has the rights to access the physical file or folder being requested.

Note NTFS ACL-based Authorization is part of the IIS Web server core and therefore is always installed when the Web server is installed. Though it cannot be uninstalled or disabled, you can remove one of the requirements in the following list to configure your application to not use it.

This authorization occurs automatically when all of the following conditions are met:

- The authenticated user identity must have a Windows token. If the request is authenticated using an authentication method that does not provide Windows tokens, for example Forms Authentication, this authorization is not performed.
- The selected handler mapping for the request specifies a resource type of File, Directory, or Either. Some mappings use the resource type of Unspecified to enable requests to virtual URLs that do not have a corresponding physical resource on disk. For these handler mappings, this authorization is not performed.

Note Most ASP.NET handler mappings are marked as Unspecified by default. However, ASP.NET includes additional functionality that ensures that if the URL maps to a physical file or folder, the access check is performed (with the exception of content located in a virtual directory that stores its files on a UNC path).

- The request URL maps to a physical file or folder that exists on disk. If the file or directory does not exist, IIS does not perform the check.
- The virtual directory corresponding to the request being made does not specify fixed access credentials. If the virtual directory specifies fixed credentials, they will be used to access all content for the virtual directory, and therefore IIS does not use the authenticated user to check access.

In addition, for you to successfully use NTFS ACL-based authorization, the following conditions must also be true:

- The physical resources have ACLs configured to properly grant or deny access to each authenticated user. This is typically done by placing all of the allowed users in a group, and granting this group access to the content.
- If the virtual directory corresponding to the request refers to a remote UNC share and does not specify fixed UNC credentials, the authenticated user identity must be able to

delegate to the remote server. This requires basic authentication, or requires Constrained Delegation or Protocol Transition to be configured. For more information, see the section titled "UNC Authentication" later in this chapter.

Because of the aforementioned limitations and the overhead of managing NTFS ACL permissions for multiple users, NTFS ACL-based authorization is not recommended as a generic mechanism for restricting access to IIS resources. Use it only if your application meets the preceding requirements and you would like to use ACLs as an authorization mechanism (for example, if you are sharing static resources for users with domain or local machine accounts, and the resources already have the right permissions configured).

Because this authorization happens automatically for physical resources, you *must* configure all required resources to grant access to the authenticated identities that need to use your application. See the section titled "Set NTFS Permissions to Grant Minimal Access" in this chapter for more information on properly configuring NTFS ACLs for use with NTFS ACL authorization.

URL Authorization

The IIS 7.0 URL Authorization feature is new in IIS 7.0. It provides a way to configure declarative access control rules that grant or deny access to resources based on the authenticated user and its role membership.

Note URL Authentication is not part of the default IIS install. You can manually install it from the Windows Features IIS feature category through Windows Features on Windows Vista or from the Security role service category of the Web Server (IIS) role in Server Manager on Windows Server 2008. See Chapter 12 for more information about installing and enabling modules.

Unlike NTFS ACL-based authorization, URL Authorization has the following advantages:

- It is not tied to authentication schemes that produce Windows identities. It can be used with any authentication schemes, including Forms Authentication, which produces custom authenticated user identities.
- It enables rules to be configured for specific URLs, not underlying files or directories. Therefore, it is not tied to specific resource types and does not require files or directories to exist.
- It stores authorization rules in the configuration, instead of NTFS ACLs. These rules are easier to create and manage, and they can be specified in distributed web.config files that travel with the application when it is deployed or copied between servers.
- It integrates with the ASP.NET Membership and Roles services, which enables custom authentication and role management modules to provide the authenticated users and

their roles. You can use ASP.NET Forms Authentication with Membership and Roles to quickly deploy a data-driven user and roles credential store for your application.

Note The IIS 7.0 URL Authorization feature is new. It is not related to the similarly named IIS 6.0 URL Authorization feature, which is overly complex, difficult to configure, and not widely used. The IIS 6.0 URL Authorization feature is not included with IIS 7.0.

The ASP.NET URL Authorization feature inspired the new URL Authorization feature, which implements similar functionality. However, some key differences exist in how rules are configured and processed. We'll discuss these differences later in this section.

Using URL Authorization to Restrict Access

To use URL Authorization to restrict access to your application, you need to configure one or more URL Authorization rules. These rules can be configured at the Web server level. Alternatively, you can configure them for a specific Web site, application, or URL. This makes it very easy to use URL Authorization to quickly restrict access to any part of your Web site to a specific set of users or roles.

These rules can be one of the following types:

- **Allow** An allow rule grants access to the resource being requested and allows request processing to proceed.
- **Deny** A deny rule denies access to the resource being requested, rejecting the request.

Both types of rules can specify a set of users, roles, and/or verbs that URL Authorization uses to match each rule to the request. As soon as a rule matches, the corresponding action (Allow or Deny) is taken. If the request is denied, URL Authorization will abort request processing with a 401 unauthorized response status code. If no rules matched, the request will also be denied.

Unlike ASP.NET URL Authorization, the deny rules are always processed before allow rules. This means that the relative order between deny and allow rules does not matter. In addition, the order between rules defined by parent configuration levels and the current configuration level does not matter, because all deny rules from all levels are always processed first, followed by all allow rules. Finally, the default behavior when no rules match is to deny the request.

By default, URL Authorization has a single rule configured at the Web server level that provides access to all users. You can restrict access to your Web site or a part of it by creating authorization rules by using the following techniques:

- Remove the default allow rule for all users and create explicit allow rules only for users and roles that should have access to the current URL level. This way, by default, all requests will be denied unless the authenticated user or role matches the configured

allow rule. This is the recommended practice, because it ensures that only the configured users and roles have access to the resource, and it denies access to everyone else.

- Create explicit deny rules for users and roles that should *not* have access to the current URL level. This may be appropriate to prevent access for only the specific users and roles. However, it is not generally a secure practice, because the set of users and roles is typically unbounded. The exception to this rule is the technique of creating a deny all anonymous users rule to restrict access only to authenticated users.

Note When designing the access control rules for your application, prefer to grant access to roles instead of specific users. This makes it easier to manage authorization rules as more users are added.

See the following section titled "Creating URL Authorization Rules" for information on configuring URL Authorization rules.

Creating URL Authorization Rules

You can use IIS Manager to configure URL Authorization rules by selecting the desired node in the tree view and double-clicking Authorization Rules. In the resulting window shown in Figure 14-5, you can see the list of rules currently in effect at the level you selected, which will include both the rules configured at higher configuration levels and the rules configured at the current configuration level.

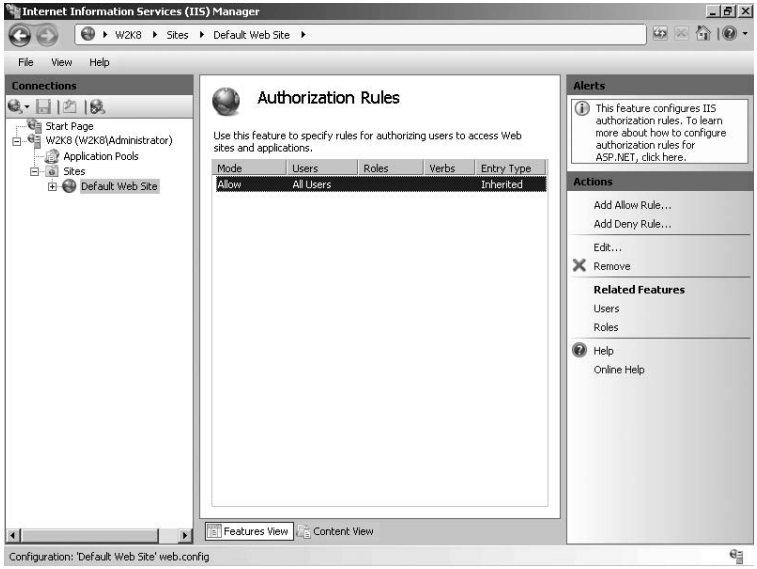


Figure 14-5 Configuring URL authorization rules.

You can remove existing authorization rules (including parent authorization rules that are not locked) by selecting them in the list and clicking Remove in the Actions pane. You can add an allow or deny rule by clicking Add Allow Rule or Add Deny Rule in the Actions pane. Figure 14-6 shows an allow rule that can be used to allow access to all users or specific users or roles.

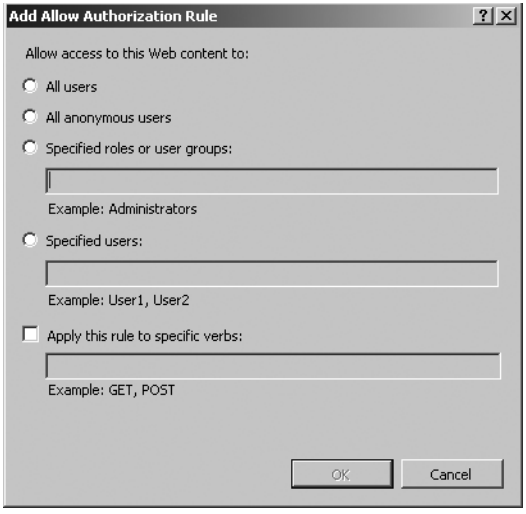


Figure 14-6 Adding an Allow URL authorization rule.

You can also edit URL Authorization rules by configuring them directly in the *system.webServer/security/authorization* configuration section, by using Appcmd from the command line, or by using configuration APIs. This configuration section is unlocked by default to facilitate storing URL authorization rules in your application's web.config files, which enables them to travel with the corresponding application content when deploying the application to another Web server.

To add a URL authorization rule with Appcmd, you can use the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd.exe set config [ConfigurationPath]
/section:system.webServer/security/authorization
"/+[users='string',roles='string',verbs='string',accessType='enum']"
```

To delete a URL authorization rule with Appcmd, you can use the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd.exe set config [ConfigurationPath]
/section:system.webServer/security/authorization
"/-[users='string',roles='string',verbs='string']"
```

The parameters to these commands are shown in Table 14-9.

Table 14-9 Parameters for Adding URL Authorization Rule

Parameter	Description
ConfigurationPath	The configuration path at which this configuration is set.
users	The comma-separated list of user names that this rule allows or denies. Each user name is matched with the user name of the authenticated user set for the request, and the rule matches as soon as a single user matches (or a role matches). For Windows identities that represent domain accounts, use the domain qualified user name format of "domain\user" rather than the fully qualified domain name format of "user@domain.com". Use "*" to refer to all users and "?" to refer to anonymous users. The default is "".
roles	The comma-separated list of roles that this rule allows or denies. Role membership for each role is tested for the authenticated user set for the request, and the rule matches as soon as a single role matches (or user matches). The default is "".
verbs	The comma-separated list of verbs (case-sensitive) that matches the request verb. If specified, one of the verbs must match the request verb for the rule to apply. The default is "".
accessType	Whether the rule should allow or deny access. Accepted values are Allow and Deny. This parameter must be specified to add a rule.

Using ASP.NET Roles with URL Authorization

In applications using the ASP.NET Integrated mode, it is possible to configure the ASP.NET Roles feature to provide application-specific roles for each authenticated user. IIS 7.0 URL Authorization rules can specify access rules that use roles provided by the .NET Roles feature or another ASP.NET module to implement application-specific authorization schemes, much like the original ASP.NET URL Authorization feature. You can learn about setting up the ASP.NET Roles feature at *http://msdn2.microsoft.com/en-us/library/9ab2fxh0.aspx*.

When the .NET Roles feature is enabled, and a role provider is configured for your application, you can begin configuring URL Authorization rules that rely on these roles in your application. To make sure that the roles are available for requests to non-ASP.NET content types, be sure to remove the managedHandler precondition from the RoleManager module. For information about enabling managed modules to run for all requests by removing the managedHandler precondition, see the section titled "Enabling Managed Modules to Run for All Requests" in Chapter 12.

You can also create roles directly using IIS Manager by selecting the application node in the tree view and then clicking.NET Roles. In the resulting page, you can manage existing roles, create new roles using the configured Role provider for the application, and associate application users with roles.

Authentication

Authentication is the process of determining the identity of the user making the request to the Web server. Authorization features can then use this identity to allow or reject the request to specific resources or parts of the application. In some cases, the Web server or the application can impersonate it to access resources. Finally, the application can use the identity to personalize the application experience for the requesting user.

IIS 7.0 includes the following authentication features:

- **Anonymous Authentication** This authentication method provides a configured Windows identity for all anonymous users of the application without the need to provide any client credentials. It is used to allow anonymous (unauthenticated) access.
- **Basic Authentication** This authentication method enables the client to provide the user name and password to the Web server in clear text. Basic Authentication is defined in RFC 2617, and virtually all browsers support it.
- **Digest Authentication** This authentication method is a more secure version of Basic Authentication, and it enables the client to provide user credentials via a hash of the user name and password. Digest Authentication is defined in RFC 2617, and most browsers support it. The implementation used in IIS 7.0 was known as the Advanced Digest Authentication method in IIS 6.0.
- **Windows Authentication** This authentication method supports the NT LAN Manager (NTLM) or Kerberos Windows authentication protocols.
- **Client Certificate Mapping Authentication** This authentication method enables client SSL certificates to be mapped to Windows accounts by using Active Directory directory services.
- **IIS Client Certificate Mapping Authentication** This authentication method enables client SSL certificates to be mapped to Windows accounts via one-to-one or many-to-one mappings stored in IIS configuration.
- **UNC Authentication** Though this is not a true authentication method in the sense that it does not help to establish the identity of the requesting client, IIS 7.0 uses UNC Authentication to establish an identity to access remote content located on a UNC share.

In addition, IIS 7.0 applications using ASP.NET Integrated mode use a unified authentication model between IIS and ASP.NET. This enables existing ASP.NET authentication modules or new managed authentication modules developed with ASP.NET APIs to be used for all content in the application. When ASP.NET is installed, the following authentication methods are also available:

- **Forms Authentication** This ASP.NET authentication method supports forms-based authentication against pluggable credentials stores via the ASP.NET Membership service. For more information on using ASP.NET Forms Authentication to protect all

Web site content, see *http://www.iis.net/articles/view.aspx/IIS7/Extending-IIS7/Getting-Started/How-to-Take-Advantage-of-the-IIS7-Integrated-Pipel*.

The following IIS 6.0 authentication methods are no longer supported:

- **IIS 6.0 Digest Authentication** IIS 7.0 Advanced Digest Authentication method is now provided as the only digest authentication method.
- **.NET Passport Authentication** The Passport support is not included in Windows Server 2008, and therefore this method is also no longer supported.

Developers can also provide custom authentication features developed with the new IIS 7.0 native module API or with ASP.NET APIs for applications using the Integrated mode. In fact, applications running in Integrated mode can use most existing custom ASP.NET authentication modules immediately to provide site-wide authentication. For more information on installing and leveraging custom modules, see Chapter 12.

You can configure one or more authentication methods for your Web site, application, or part thereof to protect it with user-based authorization, enable impersonation for resource access, or allow for application personalization.

Note IIS 7.0 requires that each request is authenticated. Because of this, at least one authentication method must be enabled and be able to provide an authenticated user for each request.

In the remainder of this section, we will review each of the authentication methods.

Anonymous Authentication

Anonymous authentication enables clients to access public areas of your Web site without requiring the client to provide any credentials. Anonymous authentication is the default authentication method enabled in IIS 7.0.

Note Anonymous authentication is part of the default IIS install and is enabled by default. You can manually install or uninstall it by installing or uninstalling the AnonymousAuthnenticationModule module. See Chapter 12 for more information about installing and enabling modules.

Anonymous authentication applies for all requests that do not have an authenticated user identity determined by other authentication methods. It works by setting the authenticated user identity for such requests to be a Windows identity corresponding to the configured anonymous user account.

Caution Be sure to disable anonymous authentication for parts of your Web site that you *do not* want to be accessed by anonymous users. You *must* do this even if you have other authentication methods enabled.

By default, anonymous authentication is configured to use the new built-in IUSR account. It no longer uses the custom IUSR_*ComputerName* account that is used by default with anonymous authentication in IIS 6.0. Because IUSR is a built-in account, it does not have a password that must be periodically changed or synchronized between multiple servers. In addition, because it is built in, the IUSR account has the same SID on all machines. Therefore, ACLs that reference it remain valid when copied from one IIS 7.0 server to another.

When using anonymous authentication, you have the following options:

- **Use the built-in IUSR account.** This is the default.
- **Use a custom account.** You can configure a custom account that should be used for anonymous requests instead of the IUSR account.
- **Use the application pool identity.** You can configure anonymous authentication to use the identity of the IIS worker process (application pool identity) instead of a separate anonymous account.

You can use the application pool identity option to simplify resource access management. This ensures that that resource access is always made under the application pool identity, both when the Web server accesses application resources using the application pool identity and when the Web server or application access resources while impersonating the authenticated user. This way, you only need to manage access rights for a single identity. See the section titled "Set NTFS Permissions to Grant Minimal Access" earlier in this chapter for more information about setting permissions.

You can use IIS Manager to enable or disable anonymous authentication and set the anonymous user options. Select the desired node in the tree view and double-click Authentication. Then, select Anonymous Authentication in the list and use the *Enable*, *Disable*, and *Edit* commands in the Actions pane to configure it.

You can also set anonymous authentication configuration directly; use Appcmd.exe from the command line, or use configuration APIs to configure the *system.webServer/security/anonymousAuthentication* section. You do this with Appcmd by using the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/anonymousAuthentication [/enabled:bool]
[/username:string] [/password:string] [/logonMethod:enum]
```

The parameters of this command are shown in Table 14-10.

Table 14-10 Parameters to Set Anonymous Authentication and Anonymous User Options

Parameter	Description
ConfigurationPath	The configuration path at which to set the specified configuration. If you specify this parameter, you may also need to specify the */commit:apphost* parameter to avoid locking errors when applying configuration to Web site or URL levels.
enabled	Whether to enable or disable anonymous authentication.
username	The user name to use for anonymous authentication. Set to "" to use the application pool identity. Default is IUSR.
password	The password to use when specifying a custom account for anonymous authentication.
logonMethod	The logon method to use for the anonymous user. Allowed values are *Interactive, Batch, Network, ClearText*. Default is *ClearText*. See *http://msdn2.microsoft.com/en-us/library/aa378184.aspx* for more information about logon types.

Basic Authentication

Basic authentication implements the Basic Authentication protocol, a standard HTTP authentication scheme defined in RFC 2617 and supported by most HTTP client software. It enables the client to pass both the user name and the password in clear text, and it uses these credentials to log on locally at the Web server or the Web server's domain. The credentials, therefore, must correspond to a valid local or domain account, and they result in the request being authenticated with a Windows token corresponding to this account.

Note Basic authentication is not part of the default IIS install. You can manually install it from the Security feature category through Windows Features On And Off on Windows Vista. You can also install it from the Security role service category of the Web Server (IIS) role in Server Manager on Windows Server 2008. See Chapter 12 for more information about installing and enabling modules.

Basic authentication is a challenge-based authentication scheme. When a client makes the initial request to a resource that requires authentication, and basic authentication is enabled, the request will be rejected with a 401 unauthorized status that will include a "WWW-Authenticate: basic" response header. If the client supports basic authentication, it will usually prompt the user for credentials and then reissue the request with the credentials included. The basic authentication module will see that credentials are present on the subsequent request and attempt to authenticate the request by logging on with those credentials. The client will typically send these credentials again on every request to the same URL or any URL that is below the URL included in the initial authenticated request.

Caution Just enabling basic authentication does not mean that authentication is required for your application. You must either disable anonymous authentication and/or configure URL authorization rules or NTFS permissions that deny access to the anonymous user.

Basic authentication is not secure because it passes the credentials in clear text, and therefore may enable an attacker to steal them by eavesdropping on the request packets at the network level. This can be mitigated by using SSL to secure the communication channel between the client and the server. If SSL is used to protect all requests that include the credentials, basic authentication may be a secure option. For more information on configuring secure communication with SSL, see the section titled "Securing Communications with Secure Socket Layer (SSL)" later in this chapter.

Caution Basic authentication may enable user credentials to be leaked because it sends them to the Web server in an unencrypted form. When using basic authentication, use SSL to secure the Web site.

Because basic authentication performs the logon locally at the Web server, the resulting Windows token can be used to access resources on a remote server without configuring delegation or Protocol Transition. See the section titled "Understanding Authentication Delegation" later in this chapter for more information.

By default, basic authentication caches the logon token for the corresponding user name and password in the token cache. During this time, the token may be available inside that process. If the worker process is compromised, malicious code can use this token to elevate privileges if the token represents a user with high privileges. If you do not trust the code in the process, you can either disable token caching by uninstalling the token cache module or reduce the amount of time the tokens are cached by setting the *HKLM\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\UserTokenTTL* value to the number of seconds to cache tokens for.

You can use IIS Manager to enable or disable basic authentication and set the logon method options. Select the desired node in the tree view and double-click Authentication. Then, select Basic Authentication from the list and use the *Enable*, *Disable*, and *Edit* commands in the Actions pane to configure it.

You can also set basic authentication configuration directly; use Appcmd.exe from the command line, or use configuration APIs to configure the *system.webServer/security/basicAuthentication* section. You do this with Appcmd by using the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/basicAuthentication [/enabled:bool]
[/realm:string] [/defaultLogonDomain:string] [/logonMethod:enum]
```

The parameters of this command are shown in Table 14-11.

Table 14-11 Parameters for Setting Basic Authentication Configuration Directly

Parameter	Description
ConfigurationPath	The configuration path at which to set the specified configuration. If you specify this parameter, you may also need to specify the *"/commit:apphost"* parameter to avoid locking errors when applying configuration to Web site or URL levels.
enabled	Whether to enable or disable basic authentication.
realm	The basic authentication realm that will be indicated to the client for informational purposes. The Web server does not use the realm during the logon process.
defaultLogonDomain	The domain that will be used by the server to log on using the credentials provided by the client. If the client user name specifies the domain, it will be used instead. If empty, the computer domain is used. The default value is "".
logonMethod	The logon method to use for the logon. Allowed values are *Interactive*, *Batch*, *Network*, and *ClearText*. Default is *ClearText*. See *http://msdn2.microsoft.com/en-us/library/aa378184.aspx* for more information about logon types.

Digest Authentication

The Digest Authentication feature implements the Digest Authentication protocol, a standard HTTP authentication scheme defined in RFC 2617 and supported by some HTTP client software. Unlike basic authentication, the client sends an MD5 hash of the user name and the password to the server so that the real credentials are not sent over the network. The Digest Authentication scheme in IIS 7.0 was known as the Advanced Digest Authentication in IIS 6.0 (IIS 7.0 no longer supports the IIS 6.0 Digest Authentication). If successful, Digest Authentication authenticates the request with a Windows token corresponding to the user's Active Directory account.

Note Digest authentication is not part of the default IIS install. You can manually install it from the Security feature category through Windows Features On And Off on Windows Vista. You can also install it through the Security role service category of the Web Server (IIS) role in Server Manager on Windows Server 2008. See Chapter 12 for more information about installing and enabling modules.

Like basic authentication, digest authentication is a challenge-based authentication scheme. When a client makes the initial request to a resource that requires authentication, and digest authentication is enabled, the request will be rejected with a 401 unauthorized status that includes a "WWW-Authenticate: digest" response header containing additional information required by the Digest Authentication scheme. If the client supports digest authentication, it will usually prompt the user for the credentials and then reissue the request with the hash of the credentials and the nonce information in the challenge. The Digest Authentication

module will see that the hash is present on the subsequent request and attempt to authenticate the hash by comparing it with the hash stored in Active Directory. The client will typically send the hash information again on every request to the same URL or any URL below the URL used in the initial authenticated request.

The Web server and the clients accessing it must meet the following requirements to use IIS 7.0 Digest Authentication:

- Both the Web server and the clients using your application must be members of the same domain, or the client must be a member of a domain trusted by the Web server.
- The clients must use Microsoft Internet Explorer 5 or later.
- The user must have a valid Windows user account stored in Active Directory on the domain controller.
- The domain controller must be using Windows Server 2003 or Windows Server 2008.

Unlike the IIS 6.0 Digest Authentication, the IIS 7.0 Digest Authentication does not require the application pool identity to be LocalSystem. In fact, you should not ever use LocalSystem or any other identity with Administrative privileges on the server as an application pool identity. For more information on configuring least privilege identities for application pools, see the section titled "Configuring Applications for Least Privilege" earlier in this chapter.

Caution Just enabling digest authentication does not mean that authentication is required for your application. You must either disable anonymous authentication and/or configure URL authorization rules or NTFS permissions that deny access to the anonymous user.

Unlike basic authentication, the authenticated token is not suitable for accessing remote resources, and it requires Constrained Delegation or Protocol Transition to be configured to do so. For more information, see the section titled "Understanding Authentication Delegation" later in this chapter.

You can enable or disable digest authentication by using IIS Manager. Select the desired node in the tree view and double-click Authentication. Then, select Digest Authentication from the list and use the *Enable*, *Disable*, and *Edit* commands in the Actions pane to configure it.

You can also set digest authentication configuration directly; use Appcmd.exe from the command line, or use configuration APIs to configure the *system.webServer/security/digestAuthentication* section. You do this with Appcmd by using the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/digestAuthentication [/enabled:bool]
[/realm:string]
```

The parameters of this command are shown in the Table 14-12.

Table 14-12 Parameters for Setting Digest Authentication

Parameter	Description
ConfigurationPath	The configuration path at which to set the specified configuration. If specifying this, you may also need to specify the */commit:apphost* parameter to avoid locking errors when applying configuration to Web site or URL levels.
enabled	Whether to enable or disable digest authentication.
realm	The digest authentication realm that will be used as specified in the RFC 2617.

Windows Authentication

The Windows Authentication scheme enables Windows clients to authenticate with two Windows authentication protocols, NTLM (NT LAN Manager) and Kerberos. Both of these schemes involve a cryptographic exchange between the client and the server to authenticate the client.

Note Unlike Windows Server 2003, Windows Authentication is not part of the default IIS install and is not enabled by default. You can manually install it from the Security feature category through Turn Windows Features On And Off on Windows Vista. You can also install it via the Security role service category of the Web Server (IIS) role in Server Manager on Windows Server 2008. See Chapter 12 for more information about installing and enabling modules. After the module is installed, you have to explicitly enable Windows Authentication for it to be available.

Windows Authentication, similar to other IIS authentication methods, is challenge-based. When a request is rejected with a 401 unauthorized response status code, Windows Authentication will issue a WWW-Authenticate challenge header including one or both of the following authentication scheme names:

- **NTLM** Indicates to the client that it can use the NTLM authentication protocol to authenticate. This is included for older clients that do not support the negotiate wrapper.
- **Negotiate** Indicates to the client that it can use Kerberos *or* NTLM protocols to authenticate. Negotiate is used to allow either Kerberos or NTLM authentication, depending on what is available on the client.

Note Both Kerberos and NTLM authentication methods involve the client making several (typically two to three) requests to the server as part of the authentication handshake. This means that your modules may see multiple requests as part of the authentication process. By default, authentication occurs once per connection, so it does not occur again for subsequent requests using the same connection.

The client then makes the decision to use either Kerberos (if available) or NTLM and initiates a sequence of requests to authenticate using the selected protocol. The choice of protocol is based on whether or not the client is configured to be able to use Kerberos to authenticate with the server, which requires a direct connection to the Key Distribution Center (KDC) on the domain controller as well as direct access to Active Directory. NTLM can be used in a non-domain scenario against local Windows accounts on the server or when the connection to domain services required for Kerberos is unavailable.

Note Windows Authentication is best suited for intranet environments.

Windows Authentication is a reasonable choice for Windows-based intranet environments, but for other environments, keep in mind the following limitations:

- It does not work over HTTP proxies. This is because Kerberos and NTLM are connection-based, and proxies may not keep connections open or may share connections between requests from multiple clients.
- The Kerberos protocol requires both the client and the server to be members of the same domain or two domains with a trust relationship and have a direct connection to Active Directory and the KDC services located on the domain controller.
- The Kerberos protocol requires correct Service Principal Name (SPN) registration in Active Directory for all application pools performing Kerberos authentication.

Configuring Windows Authentication

You can enable or disable Windows Authentication by using IIS Manager. Select the desired node in the tree view and double-click Authentication. Then, select Windows Authentication in the list and use the *Enable*, *Disable*, and *Edit* commands in the Actions pane to configure it.

You can also set digest authentication configuration directly; use Appcmd.exe from the command line, or use configuration APIs to configure the *system.webServer/security/windowsAuthentication* section. You do this with Appcmd by using the following syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/windowsAuthentication [/enabled:bool]
[/authPersistSingleRequest:bool] [/authPersistNonNTLM:bool]
[/useKernelMode:bool] [/useAppPoolCredentials:bool]
```

The parameters of this command are shown in Table 14-13.

Table 14-13 Parameters for Configure Authentication

Parameter	Description
ConfigurationPath	The configuration path at which to set the specified configuration. If you specify this, you may also need to specify the */commit:apphost* parameter to avoid locking errors when applying configuration to Web site or URL levels.
enabled	Whether to enable or disable Windows Authentication.
authPersistSingleRequest	Whether or not to require each new request to reauthenticate. If set to false, the client will perform the authentication handshake only once per connection, and the server will cache the authenticated identity for all subsequent requests. Otherwise, each request will require the authentication handshake. Default is false.
authPersistNonNTLM	Whether to require each new request to reauthenticate when using Kerberos. If set to false, the client will perform the authentication handshake only once per connection, and the server will cache the authenticated identity for all subsequent requests. Otherwise, each request will require the authentication handshake. Default is false.
useKernelMode	Whether to perform Windows Authentication in the kernel. The default is true.
useAppPoolCredentials	Whether to use the application pool identity instead of LocalSystem when performing kernel Windows Authentication. This is needed when you are using a domain account as the application pool identity to enable Kerberos authentication on a Web farm. The default is false.

In addition, you can also control whether the server uses NTLM or Negotiate protocols. To do this, you can edit the *providers* collection in the *system.webServer/security/windowsAuthentication* configuration section. By default, this collection contains both NTLM and Negotiate protocol providers. You can force the server to use only NTLM by removing the Negotiate protocol provider. However, you cannot force the server to use only Kerberos in this configuration, because the negotiate wrapper enables the client to use either NTLM or Kerberos. There is no way to tell the client that only Kerberos is supported.

You can, however, configure the NTLM authentication level by using the Local Security Policy console and modifying the Security Settings\Local Policies\Security Options\Network Security: LAN Manager Authentication Level option, as shown in Figure 14-7. The default setting is *Send NTLMv2 Response Only*, which enables the server to accept all levels. You can set this setting to *Send NTLMv2 Response Only. Refuse LM & NTLM* for maximum security while allowing clients that do not have the ability to use Kerberos to use the NTLMv2 scheme.

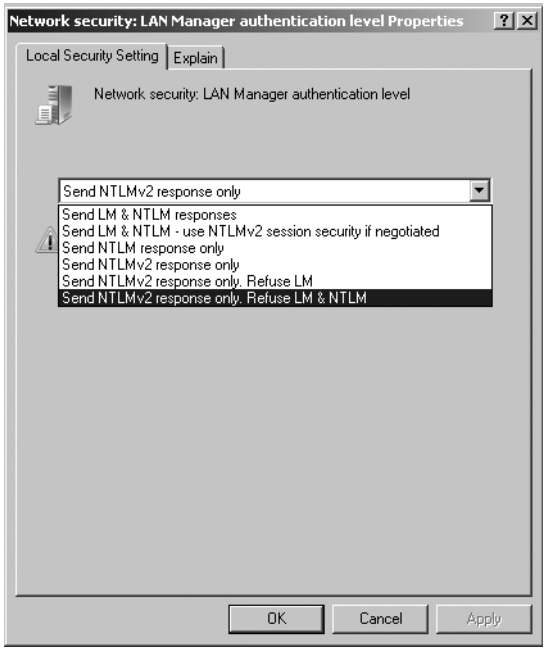


Figure 14-7 Configure NTLM Authentication Level.

In IIS 6.0, to use the Kerberos authentication protocol, you have to use the Setspn.exe command line tool to register Service Principal Names (SPNs) in Active Directory for the NetBIOS and the Fully Qualified Domain Name (FDQN) names for each application pool account. Additionally, you could have only one application pool account registered for each SPN, preventing multiple application pools with different identities from using Kerberos authentication.

In IIS 7.0, kernel-based Windows Authentication (enabled by default) offers improved functionality. Because HTTP.sys performs the authentication process in the kernel, it is done under the LocalSystem account regardless of the application pool identity. This results in the following improvements:

- It should no longer be necessary to configure separate SPNs, because Kerberos will use the default NetBIOS SPN entry created automatically when the Web server computer is joined to the domain.
- Application pool identity can be changed without the need to reregister the SPN with the new account. The application pool account no longer needs to be a domain account.
- Multiple application pools can use Kerberos authentication.

These changes make it significantly easier to deploy and use the Kerberos protocol with IIS.

Note You need to use the application pool identity and register SPNs for Kerberos authentication when you are using it on a Web farm.

However, if you are using IIS on a Web farm and require the Kerberos protocol, you will need to disable the use of the LocalSystem identity for Kerberos authentication by setting the *useAppPoolCredentials* attribute in the *system.webServer/security/authentication/windowsAuthentication* configuration section to *true*. In addition, you will need to use a domain account as an identity for the application pool. You will also be required to use Setspn.exe to register the Web site host name using this domain account under which the application pools are configured to run in Active Directory. For more information about registering SPNs for Kerberos with Setspn.exe, see *http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/523ae943-5e6a-4200-9103-9808baa00157.mspx?mfr=true*.

Client Certificate Mapping Authentication

Client Certificate Mapping Authentication enables clients to authenticate with the Web server by presenting client certificates over Secure Socket Layer (SSL) connections.

Note Certificate-based authentication enables clients to use client certificates to authenticate with the Web server. It is not required to enable secure communication between the client and the server.

The Client Certificate Mapping Authentication uses the Directory Services Mapper (DS Mapper) service in Active Directory to map client certificates provided by the user to domain accounts. IIS also provides a custom certificate mapping feature, the IIS Client Certificate Mapping Authentication, which allows for more flexible mapping of client certificates to Windows accounts. See the section titled "IIS Client Certificate Mapping Authentication" later in this chapter for more information.

Note Client Certificate Mapping Authentication is not part of the default IIS install and is not enabled by default. You can manually install it from the Security feature category through Turn Windows Features On And Off on Windows Vista. You can also install it via the Security role service category of the Web Server (IIS) role in Server Manager on Windows Server 2008. See Chapter 12 for more information about installing and enabling modules. After the module is installed, you have to explicitly enable Client Certificate Mapping Authentication for it to be available.

To use Client Certificate Mapping Authentication, you need to meet the following requirements:

- The Web server must be a member of a Windows domain.
- You must issue client certificates to your users by using a Certificate Authority (CA) trusted by the Web server.
- You must map each client certificate to a valid domain account in Active Directory.

Note You do not need to use Client Certificate Mapping Authentication to require clients to present client certificates. You can configure the server to always require client certificates to access the server, but use another authentication scheme to authenticate the client. To do this, see the section titled "Client Certificates" later in this chapter.

To enable Client Certificate Mapping Authentication on the Web server, you need to perform the following steps (after installing the Certificate Mapping Authentication module).

1. **Enable Client Certificate Mapping Authentication.** You can do this in IIS Manager by clicking the server node, double-clicking Authentication, selecting Active Directory Client Certificate Authentication, and clicking Enable in the Actions pane. Note that this can only be done at the server level when using IIS Manager, although you can enable Client Certificate Mapping Authentication for a specific URL through configuration.
2. **Configure SSL on each Web site using this authentication method.** Certificate authentication is possible only if the Web site is being accessed over an SSL connection and therefore requires an SSL binding to be configured for the Web site. See the section titled "Configuring SSL" later in this chapter for more details.
3. **Enable DS Mapper for each Web site SSL binding.** IIS Manager does this automatically for each Web site when the Client Certificate Mapping Authentication is enabled and you add an SSL binding for the Web site. To do this manually, use the *Netsh.exe* command with the following syntax: ***netsh http add sslcert IP Address:Port dsmapperusage=enable***, where *IP Address* and *Port* are the IP address and port of the corresponding binding.
4. **Configure each Web site using this authentication method to accept client certificates** (and possibly require them). This ensures that the server accepts client certificates when provided by the client and can also configure the server to require the client to present a certificate to proceed with the request. See the section titled "Client Certificates" later in this chapter for more details.

You can also enable Client Certificate Mapping Authentication by editing the *system.webServer/security/authentication/clientCertificateMappingAuthentication* configuration section directly or by using Appcmd or other configuration APIs. You can enable this authentication method by using the following Appcmd syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config /section:
system.webServer/security/authentication/
clientCertificateMappingAuthentication /enabled:true
```

The *enabled* attribute specifies whether or not the Client Certificate Mapping Authentication is enabled. You can enable this method for a specific URL. However, do note that the decision to use the Directory Services Mapper to map certificates to Windows domain accounts is dependent on each Web site binding having been configured to use the HTTP.sys DS Mapper setting.

You can read more about configuring SSL and configuring the server to accept client certificates in the section titled "Client Certificates" later in this chapter.

IIS Client Certificate Mapping Authentication

IIS Client Certificate Mapping Authentication enables clients to authenticate with the Web server by presenting client certificates over Secure Socket Layer (SSL) connections.

Note Certificate-based authentication enables clients to use client certificates to authenticate with the Web server. It is not required to enable secure communication between the client and the server using SSL.

The IIS Client Certificate Mapping Authentication provides a more flexible mechanism for authenticating clients based on client certificates than does the Active Directory–based Client Certificate Mapping Authentication. Instead of relying on the Directory Services Mapper (DS Mapper) service to map client certificates to Windows accounts, it uses the configuration to perform the mapping. As such, it also does not require the user accounts to be domain accounts and does not require Active Directory to operate.

Note IIS Client Certificate Mapping Authentication is not part of the default IIS install and is not enabled by default. You can manually install it from the Security feature category through Turn Windows Features On And Off on Windows Vista. You can also install it via the Security role service category of the Web Server (IIS) role in Server Manager on Windows Server 2008. See Chapter 12 for more information about installing and enabling modules. After the module is installed, you have to explicitly enable IIS Client Certificate Mapping Authentication for it to be available.

The IIS Client Certificate Mapping Authentication feature supports the following mapping types:

- **One-to-one mapping** Map a single client certificate to a specific Windows account. The server will use an exact copy of the client certificate to perform the match and therefore must possess a copy of each client certificate.
- **Many-to-one mapping** Map client certificates to a Windows account by matching wildcard expressions involving specific certificate fields, such as *issuer* or *subject*. This does not require a copy of the client certificate.

To use IIS Client Certificate Mapping Authentication, you need to meet the following requirements:

- You cannot use Active Directory–based Client Certificate Mapping Authentication on any of the sites for which you enable IIS Client Certificate Mapping Authentication.
- You must have the passwords for all Windows accounts used to map certificates. Unlike Client Certificate Mapping Authentication, which relies on Active Directory to generate

the Windows token for the account, you will need to specify both the user name and password for each account being mapped so that IIS can generate the token.

- To use one-to-one mappings, you must have an exact copy of each client certificate being mapped. If you provide the certificates to users, you will have this copy. Otherwise, you will need each user to provide you with an exported copy of the certificate. When using many-to-one mappings, you do not need copies of the certificates.

Note You do not need to use IIS Client Certificate Mapping Authentication to require clients to present client certificates. You can configure the server to always require client certificates to access the server and then use another authentication scheme to authenticate the client. For more about how to do this, see the section titled "Client Certificates" later in this chapter.

To enable IIS Client Certificate Mapping Authentication for a specific Web site or URL, you need to perform the following steps (after installing the IIS Certificate Mapping Authentication module):

1. **Enable IIS Client Certificate Mapping Authentication.** This option is not available in IIS Manager. To do this, you will need to edit the *system.webServer/security/authentication/iisClientCertificateMappingAuthentication* section directly. Alternatively, you can use Appcmd or another configuration API. You can enable IIS Client Certificate Mapping Authentication with Appcmd by using the following syntax (see details on configuring this configuration section later in this section).

   ```
   %systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
   /section:system.webServer/security/authentication/
   iisClientCertificateMappingAuthentication /enabled:true /commit:apphost
   ```

2. **Configure SSL on each Web site using this authentication method.** Certificate authentication is possible only if the Web site is being accessed over an SSL connection. Therefore, it requires an SSL binding to be configured for the Web site. See the section titled "Configuring SSL" for more details.
3. **Configure each Web site using this authentication method to accept client certificates** (and possibly require them). Doing so ensures that the server accepts client certificates when the clients provide them. Doing so can also configure the server to require the client to present a certificate to proceed with the request. See the section titled "Client Certificates" for more details.
4. **Configure the required one-to-one or many-to-one mappings.** Create the mappings to map certificates to Windows accounts.

Note Although you can enable the IIS Client Certificate Mapping Authentication feature for a specific URL, the mapping configuration can only be set at the server or Web site level, and it is ignored if it is set at a lower configuration level.

You can read more about configuring SSL and configuring the server to accept client certificates in the section titled "Client Certificates" later in this chapter. We will describe the process for configuring certificate mappings for IIS Client Certificate Mapping Authentication next in the sections titled "Creating One-to-One Certificate Mappings" and "Creating Many-to-One Certificate Mappings."

Creating One-to-One Certificate Mappings

You can use one-to-one certificate mappings as part of a strong authentication and authorization scheme to control access to application resources based on the exact identity of the client. It can be used instead of a user name and password authentication scheme that requires the user to supply credentials. To use one-to-one mappings, you need to have the exact copy of each certificate.

IIS Manager does not provide support for configuring one-to-one mappings. You can configure them by using the Appcmd command line tool. You can also do it by editing the *system.webServer/security/authentication/iisClientCertificateMappingAuthentication* configuration section directly or with other configuration APIs. You can add a one-to-one mapping by using the following Appcmd syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [SiteName]
/section:system.webServer/security/authentication/
iisClientCertificateMappingAuthentication /+oneToOneMappings
[certificate='string',enabled='bool',username='string',password='string']
```

You can remove a one-to-one mapping by using the following Appcmd syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [SiteName]
/section:system.webServer/security/authentication
/iisClientCertificateMappingAuthentication /-oneToOneMappings
[certificate='string']
```

These commands use the parameters in Table 14-14.

Table 14-14 Parameters for Creating Certificate Mappings

Parameter	Description
SiteName	The site name of the Web site for which to set these settings. If omitted, this parameter sets them for the entire Web server. If you specify these settings for a configuration path deeper than the Web site root, these settings will not take effect.
certificate	The exact text of the certificate (not the certificate hash).
enabled	Whether or not this mapping is enabled.
userName	The user name for the account to which the certificate maps.
password	The password for the account to which the certificate maps. This value is stored in the encrypted form.

You can obtain the exact text of the certificate from an exported certificate file (containing unencrypted certificate information) or by dumping the certificate from the local or domain certificate store. To do the latter, you can use the following command.

```
certuil –encode –f CertName cert.cer
```

CertName is the friendly name of the certificate. You can view the certificate store and obtain the friendly name of the installed certificates with the following command.

```
certuil –viewstore StoreName
```

StoreName is the name of the certificate store. Use *MY* for the personal certificate store.

Note You must specify the exact base64 encoded certificate contents for the one-to-one mapping, with the training line feed removed. Do not use the certificate hash. If you do not specify the certificate correctly, you will get a 401.1 status error when making requests to the Web site. This error will show the 0x8009310b HRESULT, indicating that IIS failed to load the certificate from the mapping entry.

Creating Many-to-One Certificate Mappings

Many-to-one mappings, unlike one-to-one mappings, are not typically used to authenticate specific users. Instead, you can use them to authenticate a group of users by matching fields in their certificates to a single Windows account. Therefore, authorization based on the authenticated user produced by a many-to-one mapping is similar to role- or group-based authorization, with the authenticated user representing a group to which multiple users belong. For example, you can match all certificates issued by a specific organization to that organization's account. As such, many-to-one mappings may be less appropriate for user-based personalization or access control than one-to-one mappings, depending on your authorization strategy.

Note One-to-one mappings are always processed before many-to-one mappings.

Many-to-one mappings do not require the server to have the exact certificate for each user. Instead, you simply configure wildcard rules based on one or more fields in the certificate that map all certificates with matching fields to a Windows account.

IIS Manager does not provide support for configuring many-to-one mappings. You can configure them by using the Appcmd command line, too. You can also edit the *system.webServer/security/authentication/iisClientCertificateMappingAuthentication* configuration section directly or with other configuration APIs. You can add a one-to-one mapping by using the following Appcmd syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [SiteName]
/section:system.webServer/security/authentication/
iisClientCertificateMappingAuthentication /+manyToOneMappings
[name='string',enabled='bool',permissionMode='enum',
username='string',password='string',description='string']
```

Then, you have to add one more matching rule to the mapping by using the following Appcmd syntax, specifying the name of the mapping created in the command shown previously.

```
%systemroot%\system32\inetsrv\Appcmd set config [SiteName]
/section:system.webServer/security/authentication/
iisClientCertificateMappingAuthentication /+manyToOneMappings
[name='string'].rules.[certificateField='enum',
certificateSubField='string',matchCriteria='string',
compareCaseSensitive='bool']
```

You can delete a mapping by using the following Appcmd syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [SiteName]
/section:system.webServer/security/authentication/
iisClientCertificateMappingAuthentication /-manyToOneMappings
[name='string']
```

These commands use the parameters in Table 14-15.

Table 14-15 Parameters for Creating Certificate Mappings

Parameter	Description
SiteName	The site name of the Web site for which to set these settings. If omitted, sets them for the entire Web server. If you specify these settings for a configuration path deeper than the Web site root, these settings will not take effect.
name	The name of the mapping; can also be used to add rules to it or delete it.
enabled	Whether or not this mapping is enabled.
permissionMode	Whether to allow or deny access to the user who is given this mapping.
userName	The user name for the account to which the certificate maps.
password	The password for the account to which the certificate maps. This value is stored in the encrypted form.
description	The friendly description of the mapping.
certificateField	The certificate field to match in the current rule. Common fields are Issuer and Subject. For more information, get the details about the contents of the certificate from the CA.
certificateSubField	The certificate subfield to match in the current rule. For more information on the subfields, get the details about the contents of the certificate from the CA.
matchCriteria	The match criteria. This can include * and ? wildcard matching characters.
compareCaseSensitive	Whether or not the comparison should be case-sensitive.

UNC Authentication

The Web server core uses UNC authentication to establish an identity for accessing remote application content inside virtual directories that reside on a UNC share. It is not a true authentication method in the sense that it does not itself support an authentication scheme for establishing the identity of the client. Rather, it is a mechanism for using the authenticated user that has been established through other authentication mechanisms—and in some cases a fixed identity set in configuration—to determine which identity should be used for remote content access.

IIS uses UNC authentication whenever a request is made to a resource that resides in a virtual directory whose physical path is located on a UNC share (whether or not the UNC share is on the local computer). During UNC authentication, the Web server determines the identity to be used for accessing remote content as follows:

1. **Uses the virtual directory's fixed credentials.** In IIS 7.0, any virtual directory can specify fixed credentials that IIS uses for all accesses to that location. This replaces the *UNCUserName* and *UNCPassword* properties in IIS 6.0 that were used only when the virtual directory referred to a UNC share.
2. **Otherwise, uses the authenticated user if available.** If the virtual directory does not specify fixed credentials, use the authenticated user if it has already been determined by an authentication method. This is referred to as pass-through authentication. You cannot use this to access web.config files, because this access occurs before IIS determines the authenticated user.
3. **Otherwise, uses process identity.** If IIS has not yet determined the authenticated user, it will use the identity of the IIS worker process. The Web server uses this option to access web.config files (if virtual directory credentials are not configured), because configuration is read prior to the authentication stage.

Note IIS 7.0 cannot use pass-through authentication to access web.config files located on the remote UNC share. Because of this, the virtual directory must specify fixed credentials, or the application pool identity must have Read access to the remote UNC share.

By default, IIS cannot access remote UNC content. This is because the default anonymous user IUSR is a local built-in account that does not have network privileges. Additionally, because IIS is required to access web.config by using the IIS worker process identity, it has a similar problem because the Network Service account also does not typically have the right to access remote resources. Therefore, you typically have two options for configuring UNC authentication to allow proper access of remote content:

- **Use pass-through authentication** Pass-through authentication requires both the application pool identity and all allowed authenticated user identities to have access to

the remote UNC share. Additionally, it requires the use of an authentication scheme that is capable of delegating the user identity to a remote computer or configuring Constrained Delegation and/or Protocol Transition to enable this for other authentication schemes.

- **Use virtual directory fixed credentials** This is the recommended approach, because it requires you to grant access to the share for a single identity. Also, it does not have the requirement of ensuring that the authentication scheme can delegate its identities to the remote UNC share, because the fixed identity is always used to access the content. However, the fixed credential model does not enable the use of NTFS authorization and auditing for authenticated users accessing the share, because the access is always made under the specified credentials and not the authenticated user identity. For more information on setting up fixed credentials for virtual directories, see the section titled "Managing Remote Content" in Chapter 9.

Note It is highly recommended that you use the fixed credential model to configure access to remote UNC shares. Use this in all cases when you do not rely on NTFS ACL-based access control or auditing of remote content for your authenticated users.

If you do choose to use pass-through authentication, you will need to take the following steps:

1. Use a custom application pool identity that has access to the UNC share.
2. If using anonymous authentication, configure the anonymous user to be the application pool identity. Alternatively, configure a custom anonymous user that has access to the UNC share.
3. If you are using other authentication methods that produce Windows identities, ensure that these methods can delegate identities to the UNC share. Then, ensure that all authenticated users have access to the UNC share.

For more information about ensuring that your authentication scheme supports delegating authenticated user identities to remote resources, see the following section titled "Understanding Authentication Delegation."

Understanding Authentication Delegation

Many IIS authentication methods produce Windows identities that can be impersonated for the purpose of accessing resources. When the resources being accessed reside on a remote machine, the authenticated user identity needs to be transmitted to the remote machine for authentication with the remote service. This process is referred to as delegation. It occurs when IIS attempts to access files located on remote UNC shares, or when the application impersonates the authenticated user identity to connect to a remote server such as SQL Server.

Most IIS authentication methods do not produce authenticated identities that are suitable for delegation. This means that when IIS is configured to use these authentication methods, IIS and the application may fail to access resources located on remote machines when impersonating the authenticated identity.

Note In general, the rule for remembering which authentication methods can be delegated is to remember which authentication methods perform their logon locally on the Web server. For example, any authentication scheme in which the user name and password are available on the Web server—such as Basic Authentication, IIS Client Certificate Mapping Authentication, or Anonymous Authentication—use the Web server to log on and therefore can delegate authenticated identities.

To ensure that your application has access to its backed resources located on remote servers, you generally have three options:

- Use an authentication method that supports delegation (see Table 14-16).
- Use fixed virtual directory credentials to create an authentication identity that can be delegated to and can be impersonated instead of the authenticated user. For more information, see the section titled "UNC Authentication" earlier in this chapter.
- Configure Constrained Delegation and Protocol Transition to upgrade the authenticated identity to an authenticated identity you can delegate to using the Kerberos protocol.

Table 14-16 lists the built-in IIS authentication schemes and the required configuration to enable delegation of authenticated identities.

Table 14-16 Built-In IIS Schemes and Required Configuration to Enable Delegation of Authenticated Identities

Authentication	Configuration
Anonymous	Delegates when using a custom anonymous user or when using a custom application pool identity as the anonymous user (1 hop)
Basic	Delegates by default (1 hop)
Windows (Kerberos)	Requires Constrained Delegation
Windows (NTLM)	Requires Constrained Delegation and Protocol Transition
Digest	Requires Constrained Delegation and Protocol Transition
Client certificate mapping	Requires Constrained Delegation and Protocol Transition
IIS Client certificate mapping	Delegates by default (1 hop)

Most of the authentication schemes that do not perform the logon locally on the machine require Constrained Delegation and Protocol Transition to be able to delegate the authenticated identity to a remote machine. Constrained Delegation refers to the ability of a service to

use a user identity obtained using the Kerberos protocol to access remote resources. Protocol Transition, used in conjunction with Constrained Delegation, enables other authentication schemes to obtain a Kerberos identity to be used with Constrained Delegation to access remote resources. To learn more about setting up Constrained Delegation and Protocol Transition, see *http://technet2.microsoft.com/WindowsServer/en/library/c312ba01-318f-46ca-990e-a597f3c294eb1033.mspx*.

Securing Communications with Secure Socket Layer (SSL)

By default, all communication between the Web server and the client occurs over a clear-text connection, which has the potential to expose the information included in the requests and responses to an attacker able to listen to the communication at the network layer. This includes packet sniffing at a local network, or compromising a router or a proxy server that is located on the path between the client and the Web server. This can result in the unintended disclosure of the response information, which may contain sensitive information, client credentials that are sent as part of some authentication methods (such as basic authentication or forms-based authentication methods), cookies, and more. The attacker can sometimes successfully use this information to misrepresent the client by providing these credentials to the Web server in a replay attack.

To prevent this from happening, you can use the Secure Socket Layer (SSL) or the newer Transport Layer Security (TLS) protocols to secure the communication between the client and server. TLS is a widely accepted standard that most browser technologies implement. In the rest of this section, we will refer to both of these protocols collectively as SSL for ease of reference.

In addition to securing the communication between the client and the Web server, SSL serves to confirm the identity of the Web server to the client. This process is widely used on the Internet today to ensure that the client is dealing with the entity that the Web site claims to represent. IIS can also use it to establish the identity of the client, if the client has an acceptable certificate. Client certificates are discussed later in the section titled "Client Certificates" later in this chapter.

Configuring SSL

To configure SSL, you must perform the following steps:

1. **Obtain a server certificate from a trusted Certificate Authority.** The Certificate Authority (CA) must be a trusted root CA for all of the clients that connect to the Web site that uses this certificate. For intranet sites, this may be a domain CA provided by the Active Directory Certificate Services. For Internet sites, this is usually a CA that is trusted by most client browsers by default. You can obtain the certificate by making a certificate request using the Server Certificates feature in IIS Manager. Alternatively, you can use a self-issued (or self-signed) certificate if you control both the Web server and the clients, and if you intend to use this certificate for testing purposes.

2. **Create a secure binding by using the HTTPS protocol and port 443 (or another port), and specify the server certificate for each Web site.** You can do this by creating a binding in IIS Manager, or by adding a binding programmatically and then using the *netsh http add sslcert ipport*=IPAddress:*443 certstorename=MY certhash*=hash *appid*=GUID command to associate the certificate with the binding. You can obtain the certificate hash from the Certificates console by viewing the certificate details and copying the value of the *Thumbprint* property.

Note Unlike IIS 6.0, where certificate association information is stored in the metabase, and the Web Publishing Service (W3SVC) is responsible for associating the site bindings with certificates when it is started, IIS 7.0 stores the certificate information directly in the HTTP.sys configuration. You can manipulate these associations by using IIS Manager or by using the *netsh http add sslcert* command.

To be accepted by the clients, the server certificate must contain Common Name (CN) entries for all of the host headers that the Web site used. This needs to be done when the certificate is requested.

Note It is possible to have multiple SSL Web sites that use unique server certificates if each Web site uses a separate IP address or port for its HTTPS binding. As in IIS 6.0, IIS 7.0 does not support having multiple Web sites with their own server certificates if the HTTPS bindings for those Web sites are on the same IP address/port and differentiate only by using host headers. This is because the host header information is not available when the SSL negotiation occurs. Because of this, the only way to have multiple Web sites on the same IP address that use host headers is to configure all of those Web sites to use the same SSL certificate with a wildcard CN. For more information, see *http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/596b9108-b1a7-494d-885d-f8941b07554c.mspx?mfr=true*.

For more information on configuring site bindings, see Chapter 9. You can also read more about requesting certificates in the IIS 7.0 online documentation at *http://technet2.microsoft.com/windowsserver2008/en/library/d780d806-e8a8-4bc5-8d7a-9f045d1f3e221033.mspx?mfr=true*

Requiring SSL

To ensure that the communication between your Web server and clients is protected, you may choose to require that clients request your Web site content over secure connections. This is an effective way to protect clients' authentication credentials or sensitive cookies issued by the Web site over unsecure connections.

Caution If your Web site enables mixed SSL usage, such as by allowing the Web site to be accessed over both SSL and unsecure connections, or by allowing portions of your Web site to be accessed over unsecure connections, be aware that requests made over these connections may leak sensitive information. For example, if your Web site uses Forms authentication to authenticate users, uses cookie-based session state, or stores sensitive information about the user in cookies, your clients may leak these cookies when making requests over unsecure connections. Therefore, always prefer to protect your entire Web site with SSL by requiring SSL for the entire Web site's URL namespace. Also, configure your cookies to include the *secure bit* to make sure the browser will not attempt to send them over unencrypted connections.

You can require SSL in IIS Manager by selecting the Web server, Web site, or another node corresponding to the URL for which you'd like to require SSL. Then double-click SSL Settings. In this feature, select the Require SSL check box to mandate SSL, as shown in Figure 14-8. You also have the option of selecting the Require 128-Bit SSL check box.

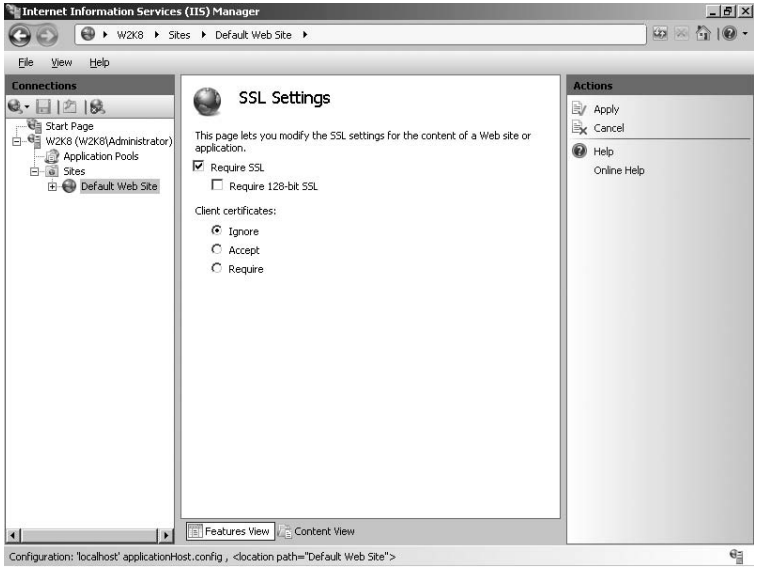


Figure 14-8 Configuring SSL settings using IIS Manager.

Alternatively, you can require SSL by editing the *system.webServer/security/access* section directly by using Appcmd or another configuration API. For example, you can set this configuration using the following Appcmd syntax.

```
%systemroot%\system32\inetsrv\Appcmd set config [ConfigurationPath]
/section:system.webServer/security/access /sslFlags:enum
```

This command has the parameters presented in Table 14-17.

Table 14-17 Parameters for Configuring SSL Settings

Parameter	Description
ConfigurationPath	The configuration path for which to apply this configuration. If you specify this, you may also need to specify the */commit:apphost* parameter to avoid locking errors when you apply configuration at Web site or URL levels.
sslFlags	A comma-separated list of one or more of the following values: *None, Ssl, Ssl128, SslNegotiateCert,SslRequireCert*. Set this to *Ssl* to require SSL, and *Ssl,Ssl128* to require 128-bit SSL. For a description of *SslNegotateCert* and *SslRequireCert*, see the following section titled "Client Certificates."

Client Certificates

Though SSL is typically used to confirm the identity of the Web server to the client, it can also be used to confirm the client's identity to the Web server if the client has certificates issued by a CA that the Web server trusts. Client certificates can be used as part of a strong two-factor authentication scheme that requires both a user name/password as well as a physical authentication method to provide the client certificate, such as a Smart Card. Or, it can be used as a single authentication method with one of the client certificate mapping authentication methods that IIS supports.

To use client certificates, you must meet the following requirements:

- The Web site must be configured to use SSL and have a valid server certificate.
- The client must have a client certificate issued by a CA that the Web server trusts.

When a client makes a connection that uses SSL, the Web server negotiates the client certificates (if configured to do) by indicating the list of trusted CAs on the server, causing the client to respond with the list of certificates that are available on the client and that are issued by those CAs. The server then validates the certificates, including checking their expiration times and making sure that they are not listed on the Certificate Revocation List (CRL) on the Web server.

IIS supports multiple levels of using client certificates:

1. **Negotiate certificates.** This requests the client to provide a client certificate when the request is made, but it does not require the certificate. If the client provides it, the server validates the certificate, and the certificate is made available to the application. To do this, set the *sslFlags* attribute of the *system.webServer/security/access* configuration section to include *SslNegotiateCert,* as described in the section titled "Requiring SSL" earlier in this chapter.
2. **Require certificates.** This requires that the client provide a client certificate when the request is made. If the certificate is not provided, the request is rejected with a 403.7 – Client Certificate Required error. If the certificate is not successfully validated by the

server, the request will be rejected with a 403.16 – Client Certificate Is Untrusted Or Invalid error. It could also be rejected with a 403.17 – Client Certificate Has Expired Or Is Not Yet Valid error. To require certificates, set the *sslFlags* attribute of the *system.webServer/security/access* configuration section to include *SslNegotiateCert,SslRequireCert,* as described in the section titled "Requiring SSL" earlier in this chapter. You can require certificates to implement a strong two-factor authentication scheme. Alternatively, you can require certificates in conjunction with a client certificate mapping authentication scheme as the primary authentication scheme for your Web site. For more information, see the sections titled "Client Certificate Mapping Authentication" and "IIS Client Certificate Mapping Authentication" earlier in this chapter.

3. **Authenticate users with client certificates.** IIS can also be configured to authenticate clients based on the client certificates. To learn more about using client certificate authentication, see the sections titled "Client Certificate Mapping Authentication" and "IIS Client Certificate Mapping Authentication" earlier in this chapter.

Securing Configuration

Previous versions of IIS have used a centralized configuration store known as the metabase. IIS 7.0 abandons the metabase in favor of a new configuration system based on a hierarchy of XML configuration files, in order to provide for simpler deployment and more flexible management of the Web server.

Note You can learn more about the new IIS 7.0 configuration system in Chapter 4, "Understanding the Configuration System."

In this section, we will take a look at the files that comprise the IIS 7.0 configuration hierarchy and how they are accessed. We will also review security best practices for limiting access to these files to ensure that the configuration contained therein is secure against unauthorized information disclosure and tampering. In addition, we'll look at isolating the configuration between application pools that are using the new configuration isolation support.

Because the configuration is stored in plain text XML files, some of which may be taken off the server during deployment or otherwise exposed, it is sometimes necessary to protect the information stored therein from being discovered. The configuration system provides built-in encryption support to protect secrets stored in configuration files against disclosure even if an attacker is able to access the file. Later in this section, we'll take a look at best practices for storing secrets in configuration files and using encryption to protect them.

The configuration file hierarchy goes beyond centralized configuration and includes distributed web.config configuration files access that can be delegated to the site or application owner. This enables sites and applications to contain required configuration as part of their content for single-step deployment that does not require administrative rights on the server. It

also enables site and application owners to manage their applications remotely without having administrative rights. In this section, we will review the best practices for securely configuring configuration delegation.

Note For more information about enabling and securing remote delegated management, see Chapter 8.

Restricting Access to Configuration

IIS 7.0 configuration is stored in a hierarchy of configuration files, including both server-level configuration files and distributed web.config files that may be delegated to site and application owners. Because these files store configurations by using plain text XML, anyone with the ability to access them can read and/or tamper with a server configuration without using any additional tools or APIs. Therefore, the NTFS access permissions placed on these files determine who can access server configuration and what they can do with it.

To properly secure configuration files, it is important to understand the files that comprise the hierarchy and how they are accessed. This can help define and maintain the proper access strategy in your environment.

Note You can learn about the configuration file hierarchy, the locations of each file, and their role in configuring the server in Chapter 4.

The files that constitute the IIS 7.0 configuration hierarchy, and their default access permissions, are listed in Table 14-18.

Table 14-18 Default Access Permissions for Configuration Files

File	Description	Default Permissions
Framework machine.config	Machine-level .NET Framework configuration	BUILTIN\IIS_IUSRS:(RX) BUILTIN\Users:(RX)
Framework root web.config	Machine-level configuration for ASP.NET applications	BUILTIN\IIS_IUSRS:(RX) BUILTIN\Users:(RX)
applicationHost.config	IIS machine-level configuration	NT SERVICE\WMSvc:(R) NT SERVICE\WMSvc:(R)
<*AppoolName*>.config	Auto-generated version of applicationHost.config for each application pool	IIS APPPOOL\<*ApppoolName*>:(R)
Distributed web.config (wwwroot)	Delegated configuration files in the Web site directory structure	BUILTIN\IIS_IUSRS:(RX) BUILTIN\Users:(RX)

Table 14-18 Default Access Permissions for Configuration Files

File	Description	Default Permissions
administration.config	Machine-level configuration file for IIS Manager	NT SERVICE\WMSvc:(R)
redirection.config	Machine-level configuration file for configuring remote location of applicationHost.config	NT SERVICE\WMSvc:(R)

Note The default permissions for all entries in Table 14-18 also contain permissions granting full rights to NT AUTHORITY\SYSTEM and BUILTIN\Administrators. These were removed from this table for clarity.

Looking at the default permissions lets you see that:

- The server-level configuration files, including Framework machine.config, Framework root web.config, applicationHost.config, administration.config, and redirection.config are writable only by the System and members of the Administrators group.
- All members of the Users group and the IIS_IUSRS group Framework can read machine.config and root web.config files. Unlike IIS server-level configuration files, any user on the machine—as well as any application running inside the IIS worker processes—can read the configuration in these files. This is due to the fact that these files are used to configure the behavior of .NET Framework components in any .NET application that runs on the machine.
- The IIS server-level configuration files, applicationHost.config, redirection.config, and administration.config, are only readable by the system, members of the administrators group, and the Web Management Service (NT Service\WMSvc). Unlike .NET Framework configuration files, they *cannot* be read by non-administrative users or even IIS worker processes. IIS worker processes receive a subset of configuration in the applicationHost.config file from the automatically generated *ApppoolName*.config files for each application pool.
- The Windows Process Activation Service (WAS) automatically generates the *Apppool-Name*.config files for each application pool, which are readable only by the IIS worker processes in the corresponding application pool. This is the basis of configuration isolation explained in the section titled "Understanding Configuration Isolation" later in this chapter.
- The distributed web.config files located in the site directory structure are by default readable by members of the Users group. These files typically must also grant access to the IIS_IUSRS group to allow access to the IIS worker process (IIS setup automatically grants this for the default Web site root located in *%SystemDrive%*\Inetpub\Wwwroot).

Setting Permissions on Configuration Files

The configuration files in the IIS hierarchy have restrictive permissions configured by default and should typically not be changed (with the exception of distributed web.config files that are part of your site directory structure). Changes to the permissions on these files may cause these files to become more vulnerable to unauthorized access. Keep the following in mind to maintain the security of these files:

- Never grant non-administrative identities (with the exception of NT SERVICE\WMSvc) access to applicationHost.config, redirection.config, and administration.config (either Read or Write). This includes Network Service, IIS_IUSRS, IUSR, or any custom identity used by IIS application pools. IIS worker processes are not meant to access any of these files directly. See the following section titled "Understanding Configuration Isolation" for information on how IIS worker processes get the configuration from applicationHost.config.
- Never share out applicationHost.config, redirection.config, and administration.config on the network. When using Shared Configuration, prefer to export applicationHost.config to another location (see the section titled "Setting Permissions for Shared Configuration" later in this chapter).
- Keep in mind that all users can read .NET Framework machine.config and root web.config files by default. Do not store sensitive information in these files if it should be for administrator eyes only. Encrypt sensitive information that should be read by the IIS worker processes only and not by other users on the machine.

The only exception to this rule is the distributed web.config files that are part of your Web site's directory structure. It is up to you to ACL these files correctly to prevent unauthorized access to their contents. You should follow the standard guidance for setting permissions for your Web site content provided in the section titled "Setting NTFS Permissions to Grant Minimal Access" earlier in this chapter, including using application pool isolation to properly restrict access to the application pool to which the application belongs and setting required permissions to allow remote delegated administration through IIS Manager.

Understanding Configuration Isolation

As mentioned earlier, IIS worker processes do not have Read access to applicationHost.config. How, then, are they able to read any of the configuration set in this file?

The answer lies in the configuration isolation feature provided by IIS 7.0, which is always on by default. Instead of enabling IIS worker processes to read applicationHost.config directly when reading the configuration file hierarchy, IIS generates filtered copies of this file and uses these copies as a replacement of applicationHost.config when configuration is read inside the IIS worker process.

The reason for doing this is to prevent IIS worker processes from application pool A to be able to read configuration information in applicationHost.config that is intended for application pool B. Because applicationHost.config may contain sensitive information, such as the user name and password for custom application pool identities, as well as user name and password for virtual directories, allowing all application pools to access applicationHost.config would break application pool isolation.

WAS is responsible for generating the temporary application pool configuration files that each IIS worker process uses as a replacement of applicationHost.config. These files are placed by default in the *%SystemDrive%*\Inetpub\Temp\Apppools directory and are named *AppPoolName.config*. As mentioned earlier, these files are configured to allow access only to the IIS worker processes in the corresponding application pool, by using the IIS APPPOOL*AppPoolName* Application Pool SID.

Note This process occurs automatically each time applicationHost.config is changed and therefore does not require any manual action from the administrator outside of normal configuration procedures.

Each application pool configuration file contains the configuration in applicationHost.config, with the following information removed:

- All application pool definitions in the *system.applicationHost/applicationPools* configuration section. Only WAS is required to read this configuration section.
- Any Web site definitions in the *system.applicationHost/sites* configuration section for sites that do not have applications in the current application pool.
- Any configuration in location tags for specific Web sites, applications, or URLs that do not reside inside the applications in the current application pool.

Caution All application definitions (and their virtual directory definitions, possibly containing user name and password credentials) for any site that has at least one application in the current application pool will be present in the application pool configuration file. To disable this behavior and include only the application definitions for applications in the application pool, set the *IsolationWholeSiteInclude* DWORD value to 0 in the HKLM\System\CurrentControlSet\Services\WAS\Parameters key and perform an IISRESET. This may break applications in sites with applications in multiple application pools when they attempt to map physical paths for URLs in other applications.

Keep in mind that global configuration settings set in the applicationHost.config (without using location tags to apply them to specific Web sites, applications, or URLs) are not filtered. Each application pool configuration file will contain all of these settings.

Configuration isolation is a key part of the application pool isolation strategy in IIS 7.0. It is enabled by default to provide configuration isolation for server-level configuration in applicationHost.config. For strategies on achieving proper application pool isolation, see the section titled "Isolating Applications" earlier in this chapter.

Caution Configuration stored in .NET Framework machine.config and root web.config files is not isolated. Only configuration stored in applicationHost.config is isolated.

Setting Permissions for Shared Configuration

IIS 7.0 supports sharing server configuration in the applicationHost.config configuration file between multiple Web servers on a Web farm. Using a shared configuration requires exporting the configuration from the source Web server, placing it on a network share, and configuring the share so that all member Web servers have access to it. The process of doing this is explained in Chapter 4.

To prevent unauthorized access to the shared configuration, follow these guidelines:

- Do not grant Write access to the identity that the Web server uses to access the shared applicationHost.config. This identity should have only Read access.
- Use a separate identity to publish applicationHost.config to the share. Do not use this identity for configuring access to the shared configuration on the Web servers.
- Use a strong password when exporting the encryption keys for use with shared configuration.
- Maintain restricted access to the share containing the shared configuration and encryption keys. If this share is compromised, an attacker will be able to read and write any IIS configuration for your Web servers, redirect traffic from your Web site to malicious sources, and in some cases gain control of all Web servers by loading arbitrary code into IIS worker processes. Consider protecting this share with firewall rules and IPsec policies to allow only the member Web servers to connect.

Warning Maintain restricted access to the share containing shared configuration. Malicious access to this share can cause complete Web server compromise.

For more information on setting up shared configuration, see Chapter 4.

Securing Sensitive Configuration

The information in the configuration files in the IIS 7.0 configuration hierarchy is protected by the restricted permissions specified by the NTFS ACLs on each file. These permissions should

prevent unauthorized users from being able to access these files. For more information on maintaining secure permissions on the configuration files, see the section titled "Restricting Access to Configuration" earlier in this chapter.

However, this alone may not provide a sufficient level of protection for especially sensitive information stored in configuration files, such as user names and passwords of custom application pool identities. It is essential to prevent this information from being discovered even if an attacker manages to compromise the local Web server and gain access to the configuration file containing the information. In addition, if someone copies the configuration file off the server for archival or transport reasons, an attacker should not be able to read the secrets stored in the configuration file. To ensure this, IIS 7.0 provides the ability to encrypt specific information stored in configuration.

Using Configuration Encryption to Store Configuration Secrets

IIS 7.0 configuration encryption works by encrypting the contents of configuration attributes for which encryption is enabled before storing their values in the configuration file. Therefore, even if someone obtains access to the file, they cannot read the contents of the attribute without decrypting it first.

Whether or not configuration encryption is used for each attribute is determined by the attribute's definition in the schema of the containing configuration section. You can find more information about how encryption is configured in the configuration section schema in the section titled "Protecting Sensitive Configuration Data" in Chapter 13, "Managing Configuration and User Interface Extensions." The schema also serves as a mechanism to select the encryption provider used to encrypt the data for this attribute (you can learn more about available encryption providers in the following section titled "Selecting Encryption Providers").

When any configuration tool or API writes the value of each encrypted attribute, the configuration system will automatically encrypt it using the configured encryption provider before persisting it to the configuration file on disk. If you inspect the resulting configuration file, you will see the encrypted value, as shown in the following code example for the password attribute in the application pool definition inside the *system.applicationHost/applicationPools* configuration section.

```
<applicationPools>
    <add name="MyAppPool">
        <processModel identityType="SpecificUser" userName="TestUser"
password="[enc:IISWASOnlyAesProvider:N8mr4dLU6PnMW5xlmCWg6914cKePgeU0fTbxew
ZppiwyTLmBQh0mZnFywQO78pQY:enc]" />
    </add>
</applicationPools>
```

The configuration system decrypts the attribute automatically when it is accessed, provided that the caller of the configuration system has the rights to use the encryption provider used to perform the encryption. Therefore, the decryption and encryption process is completely

transparent to the administrator, while ensuring that the resulting configuration is not stored in plain text.

Selecting Encryption Providers

IIS provides several encryption providers that can be used to encrypt configuration, in addition to several encryption providers provided by the .NET Framework. One of these providers is used for each configuration attribute that is marked for encryption. The providers are listed in Table 14-19.

Table 14-19 Configuration Encryption Providers

Provider	Use	Encryption Key Access
RsaProtectedConfiguration Provider	Encrypting .NET Framework configuration sections using exportable RSA encryption	RSA machine key: SYSTEM and administrators only; grant access using Aspnet_regiis.exe -pa.
DataProtectionConfiguration Provider	Encrypting .NET Framework configuration sections using machine-local Data Protection API encryption	By default, everyone on the Web server; optionally user-based key
IISWASOnlyRsaProvider	Encrypting IIS configuration sections read by WAS using exportable RSA encryption	RSA machine key: SYSTEM and administrators only
IISWASOnlyAesProvider	Encrypting IIS configuration sections read by WAS using AES encryption	Session key encrypted with RSA machine key: SYSTEM and administrators only
AesProvider	Encrypting IIS configuration sections read by the IIS worker process using AES encryption	Session key encrypted with RSA machine key: IIS_IUSRS, NT Service\WMSvc

.NET Framework creates both the RsaProtectedConfigurationProvider and the DataProtectionConfigurationProvider providers. These providers are primarily used to encrypt .NET configuration for ASP.NET applications using the Aspnet_regiis.exe tool. For more information on using the .NET Framework configuration encryption, see *http://msdn2.microsoft.com/en-us/library/53tyfkaw.aspx*.

Note You cannot use IIS configuration encryption to encrypt .NET configuration sections. Likewise, you cannot use .NET configuration encryption to encrypt the contents of IIS configuration sections with Aspnet_regiis.exe. If you attempt to read .NET configuration sections encrypted with .NET configuration encryption by using IIS configuration APIs, you will receive an error, because IIS does not support section-level encryption used by the .NET Framework configuration system.

You can use the IIS encryption providers—IISWASOnlyRsaProvider, IISWASOnlyAesProvider, and AesProvider—to encrypt IIS configuration sections.

IISWASOnlyRsaProvider and IISWASOnlyAesProvider are both used to encrypt configuration sections that WAS reads, such as the *system.applicationHost/applicationPools* section, and do not allow IIS worker processes to decrypt the configuration. The IISWASOnlyAesProvider provides better performance because it uses AES encryption using an RSA encrypted session key, instead of using full RSA encryption, and is used by default. The session key itself is encrypted using the RSA key container used by IISWASOnlyAesProvider, so it has the same access requirements. Configuration attributes encrypted using these providers can only be decrypted by SYSTEM and members of the Administrators group.

The *AesProvider* provider is an AES provider that uses a session key encrypted using an RSA key container that has permissions for the IIS_IUSRS group, therefore allowing IIS worker processes to encrypt and decrypt configuration encrypted with this provider. This is the provider used by default by all IIS configuration sections that are read by IIS worker processes. It is also the provider you should consider using to protect your custom configuration sections. Configuration attributes encrypted using this provider can be decrypted by any IIS application.

Note The IIS configuration system does not support pluggable encryption providers unlike the .NET configuration system. However, you can configure new instances of the IIS configuration provider types to use different key containers for encryption purposes.

You can also create additional instances of the RSA and AES providers by creating new entries in the *configProtectedData* configuration section, and configure them to use new RSA key containers. You can create new RSA key containers by using the Aspnet_regiis.exe −pc command, as described at *http://msdn2.microsoft.com/en-us/library/2w117ede.aspx*.

You can then manipulate the permissions on the RSA key to determine who can use it to encrypt and decrypt configuration by using the *Aspnet_regiis.exe −pa* and *Aspnet_regiis −pr* commands.

Keep in mind the following guidelines when using encryption:

- Configuration encrypted with IISWASOnlyAesProvider can only be decrypted by members of the Administrators group. This provider is only used to encrypt configuration read exclusively by WAS.
- Configuration encrypted using *AesProvider* can be decrypted by any IIS application. It can be used to protect configuration from being disclosed outside of the Web server, but it is not a way to protect configuration from applications running on the Web server. It also does not protect configuration used by one application pool from another application pool (although this protection may be afforded by proper NTFS permissions configured for application pool isolation).

- If you require to encrypt configuration for each application pool as an additional isolation measure, you should create separate RSA keys for each application pool identity and ACL them for that application pool using the Application Pool SIDs or custom application pool identities. Then you can create a provider for each application pool, using the corresponding RSA keys, and encrypt the configuration for each application pool using the corresponding provider.
- In order to share a configuration on a Web farm or deploy an application with a encrypted configuration to another server, you must share encryption keys and provider definitions between the original server on which encryption was performed and the target server. When exporting encryption key containers, be sure to use a strong password and protect these keys from being accessed by unauthorized users. You can learn more about exporting encryption keys here: *http://msdn2.microsoft.com/en-us/library/2w117ede.aspx*. In addition, you can export encryption keys by using IIS Manager when setting up shared configuration. For more information, see Chapter 4.

Caution Changing the permissions on the RSA key containers may lead to compromise of the encryption keys and therefore may expose your encrypted configuration. Do not change the default permissions on the built-in IIS RSA key containers.

Limitations of Storing Secrets in Configuration

When you store secrets in configuration, the secret is protected by both the NTFS permissions on the configuration file (see the section titled "Restricting Access to Configuration" earlier in this chapter) and configuration encryption, if configured. However, you should be aware of the following limitations that may impact the security of your secret:

- NTFS permissions provide the basic level of protection for secrets in configuration files. However, when configuration files are archived, copied off the machine, or sent over a network, this protection is lost. Always use encryption to provide protection for secrets in these cases.
- Any code in the IIS worker process can decrypt any encrypted configuration data that the IIS worker process has access to. By default, any IIS worker processes can decrypt any configuration data encrypted using the default IIS encryption provider (*AesProvider*).
- Encryption is only as secure as the key that is used to perform the encryption. Therefore, be sure that only the users authorized to perform decryption have access to the key container used to perform the encryption and make sure that this key container is not compromised if it is exported off the machine.

Limiting Access to Configuration from Managed Code in Partial Trust Environments

When accessing IIS configuration from native code, the permissions set on the configuration files are the basis for determining whether or not access to the configuration is granted. Native modules and other code running in the IIS worker process can therefore read any of the configuration in the configuration file hierarchy that is not hidden by application pool isolation or encrypted with encryption keys that the IIS worker process does not have access to.

However, when managed code modules access configuration using the *Microsoft.Web.Administration* API, their ability to read some configuration sections can be further constrained through Code Access Security (CAS) policy configured for the application. This is similar to how CAS is used to prevent managed code applications from performing other actions that the hosting process may otherwise be allowed to perform, such as accessing files or opening network connections.

You can leverage this mechanism to prevent ASP.NET applications running in partial trust from being able to access information from certain configuration sections. This is done by setting the *requirePermission* attribute on the section declaration to *true*. When this is done, only ASP.NET applications and managed modules running with Full Trust can read the contents of these configuration sections. For more information on setting the *requirePermission* attribute as part of the section declaration process, see the section titled "Declaring Configuration Sections" in Chapter 13.

Note The *requirePermission* attribute only prevents the application from using the configuration APIs to read the configuration section when in partial trust. The application can still access the file directly, if the CAS policy and file permissions allow it. Because of this, *requirePermission* is only effective at preventing medium or below trust applications from reading the contents of configuration sections specified outside of the application's directory structure, such as in applicationHost.config. The application can still open the distributed web.config files in its directory structure by using file IO APIs directly.

By default, no IIS configuration sections are declared with *requirePermission* set to *true*, so the contents of IIS configuration sections can be read by partial trust applications. So, this technique is more applicable to new configuration sections being declared.

For more information on using ASP.NET trust levels to constrain the execution of managed modules and ASP.NET applications, see the section titled "Reduce Trust of ASP.NET Applications" earlier in this chapter.

Controlling Configuration Delegation

One of the key management scenarios that the IIS 7.0 configuration system has in mind is configuration delegation. Configuration delegation refers to the ability of the Web site or

application owner to specify the required IIS configuration for their application without being an administrator on the Web server computer. To allow this, the IIS 7.0 configuration file hierarchy supports specifying configuration in distributed web.config files, which can be located anywhere in the Web site's directory structure to override the configuration specified at the server level. This also allows Web sites and applications to become portable by including all of the configuration files necessary alongside their content, so they can be deployed by simply being copied to the Web server.

If configuration delegation was an all or nothing approach, it likely wouldn't work, because most Web server administrators would not want to allow the Web site or application to be able to override all of the configuration set at the server level, especially for configuration sections that affect security, reliability, and performance of the Web server. Therefore, the IIS 7.0 configuration system provides an extensive set of controls that server administrators can use to determine which configuration sections, and further yet, which specific configuration attributes, can be overridden at the Web site or application level. If you manage a Web server that allows others to publish application content, you will likely need to review the configuration allowed for delegation, and in some cases lock or unlock specific configuration for delegation.

Further, IIS 7.0 also provides the infrastructure for Web site and application administrators to manage their configuration remotely through IIS Manager, without having administrative privileges on the Web server computer. Again, as a server administrator, you have fine-grained control over who can manage the Web sites and applications on your computer remotely, and what management features they can use. You can learn more about configuring remote management permissions in Chapter 8.

Controlling Which Configuration Is Delegated

The configuration section is the basic unit of configuration delegation. By default, each configuration section is marked to initially allow or deny delegation when it is first declared, by specifying the *overrideModeDefault* attribute in the section declaration (this is typically determined by the developer based on whether the section is considered sensitive and should not be modifiable by non-Administrators by default). If the section is marked as not delegated, any attempt to specify the configuration for this section at any lower level in the configuration hierarchy will lead to a configuration error when this section is accessed.

Note You can learn more about declaring configuration sections and the *overrideModeDefault* attribute in the section titled "Declaring Configuration Sections" in Chapter 13.

By default, all IIS configuration sections are declared in applicationHost.config. Each section declaration specifies whether or not this section is available for delegation, based on the Microsoft IIS team's criteria for whether or not the configuration section is sensitive. This criteria includes considerations of whether the configuration section can be used to weaken

the security, reduce reliability, or significantly impact the performance of the Web server overall, or allow the Web site or application to access information outside of its boundaries.

Note You can also manage the delegation of .NET configuration sections using the IIS administration stack. Both IIS and .NET configuration use the same mechanism for controlling delegation, including section-level locking and fine-grained configuration locking. For more information, see Chapter 4.

The default delegation of IIS configuration sections is shown in Table 14-20.

Table 14-20 Default Delegation of IIS Configuration Sections

Section	Default State	Reason
system.applicationHost		
applicationPools	n/a	Section can be specified only in applicationHost.config
configHistory	n/a	Section can be specified only in applicationHost.config
customMetadata	n/a	Section can be specified only in applicationHost.config
listenerAdapters	n/a	Section can be specified only in applicationHost.config
log	n/a	Section can be specified only in applicationHost.config
sites	n/a	Section can be specified only in applicationHost.config
webLimits	n/a	Section can be specified only in applicationHost.config
system.webServer		
asp	Deny	Contains security, performance, and reliability sensitive settings for ASP applications
caching	Allow	
cgi	Deny	Security sensitive: createProcessAsUser
defaultDocument	Allow	
directoryBrowse	Allow	
fastCgi	n/a	Section can be specified only in applicationHost.config
globalModules	n/a	Section can be specified only in applicationHost.config

Table 14-20 Default Delegation of IIS Configuration Sections

Section	Default State	Reason
handlers	Deny; Allow when .NET Extensibility is installed	For compatibility with IIS 6.0; section is effectively unlocked as soon as .NET Extensibility is installed; see "Locking Down Extensibility" section in Chapter 12
httpCompression	n/a	Section can only be specified in applicationHost.config
httpErrors	Deny	Security sensitive: ability to specify error pages outside of the application
httpLogging	Deny	Security sensitive: turning off logging can create repudiation issues
httpProtocol	Allow	
httpRedirect	Allow	
httpTracing	Deny	Performance sensitive: list of ETW URLs to trace
isapiFilters	n/a	Section can be specified only in applicationHost.config
modules	Deny; Allow when .NET Extensibility is installed	For compatibility with IIS 6.0; section is effectively unlocked as soon as .NET Extensibility is installed; see "Locking Down Extensibility" section in Chapter 12
odbcLogging	Deny	Security sensitive: configuring logging to external database
serverRuntime	Deny	Security, performance, and reliability affecting settings for the core Web server engine
serverSideInclude	Deny	Security sensitive: enabling server-side include can allow the application to access content outside of its boundaries
staticContent	Allow	
urlCompression	Allow	
validation	Allow	
system.webServer/security		
access	Deny	Security sensitive: configure SSL requirements
applicationDependencies	n/a	Section can be specified only in applicationHost.config
authorization	Allow	

Table 14-20 Default Delegation of IIS Configuration Sections

Section	Default State	Reason
ipSecurity	Deny	Security sensitive: determine who can access the application
isapiCgiRestriction	n/a	Section can only be specified in applicationHost.config
requestFiltering	Allow	Caution: delegated, but application can end up removing basic protection configured at server level and lessen its security
system.webServer/security/ authentication		
anonymousAuthentication	Deny	Security sensitive: enable or disable authentication method
basicAuthentication	Deny	Security sensitive: enable or disable authentication method
clientCertificateMapping Authentication	Deny	Security sensitive: enable or disable authentication method
digestAuthentication	Deny	Security sensitive: enable or disable authentication method
iisClientCertificate Mapping-Authentication	Deny	Security sensitive: enable or disable authentication method
windowsAuthentication	Deny	Security sensitive: enable or disable authentication method
system.webServer/ tracing		
traceFailedRequests	Allow	
traceProviderDefinitions	n/a	Section can only be specified in applicationHost.config

The default delegation state for IIS configuration sections is just that—a default—and may not work for everyone. If you allow third parties to publish Web site or application configuration on the server, you will need to review the impacts of allowing each section to be delegated and strike a balance between application requirements for delegation and the need to protect the Web server from unintended or malicious configuration changes. Then, you can lock or unlock configuration sections to allow them for delegation or even use fine-grained configuration locking to allow section delegation but lock specific configuration attributes, elements, or collection entries.

Note For information about how to lock and unlock sections, and use fine-grained configuration locking, see the "Delegating Configuration" and "Granular Configuration Locking" sections in Chapter 4.

When determining which configuration should be delegated, keep the following guidelines in mind:

- Err on the side of leaving configuration sections locked at the server level and unlock specific sections as needed by the application. You can also unlock specific sections for specific Web sites or applications only and leave them locked for others. This is an effective method to avoid unexpected configuration changes at the application level even if you do not delegate configuration to other parties.
- When unlocking a specific section, you can still lock parts of it that contain sensitive configuration or configuration you do not want to be changed. Use fine-grained configuration locking to lock the attributes, elements, or collection elements that you don't want changed while allowing other parts of the configuration section to be delegated.

Summary

IIS 7.0, much like its predecessor, comes with secure defaults that minimize the risk of exploits against the Web server. As you deploy your applications to the Web server and change configuration, you should familiarize yourself with the configuration and features to make sure that you do not introduce any threats to the Web server. In this chapter, you have reviewed the security changes and new security features in IIS 7.0 that can help you maintain the security of the Web server.

Unfortunately, history has shown that most Web server exploits are directed at the application running on the Web server rather than at the Web server itself. Applications are often tested less rigorously then the Web server features and are often designed with less understanding of the threat vectors that exist for Web-facing applications. Because of this, it is important to perform rigorous threat modeling and security testing at the application layer to minimize application vulnerabilities.

In addition, it is important to take an approach to security that does not depend on specific application threat vectors. IIS 7.0 makes it possible to apply such an approach, by reducing the surface area of the Web server and running the application components with least privilege possible. Together, these two techniques can both minimize the risk of any known or future exploit and reduce the damage if such an exploit does occur. By using the best practices in this chapter, you can successfully apply these techniques to your application to minimize the risk of a security compromise of your Web server.

Finally, be aware that a Web server does not function in a vacuum. It depends on a variety of Windows subsystems for its security and relies on the security of the network and other services around it. Be sure to consider the security of the network overall and related services when designing a secure Web farm.

Additional Resources

- Chapter 4, "Understanding the Configuration System," for information about configuring IIS7.
- Chapter 11, "Hosting Application Development Frameworks," for information about the execution privileges of application frameworks.
- Chapter 12, "Managing Web Server Modules" for information about managing and securing web server modules.
- Improving Web Application Security: Threats and Countermeasures: *http://msdn2.microsoft.com/en-us/library/ms994921.aspx*.
- *http://www.iis.net*.
- *http://www.mvolo.com* for frequent blog coverage of IIS 7.0, including security information.

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

Q

R

S

T

U

X