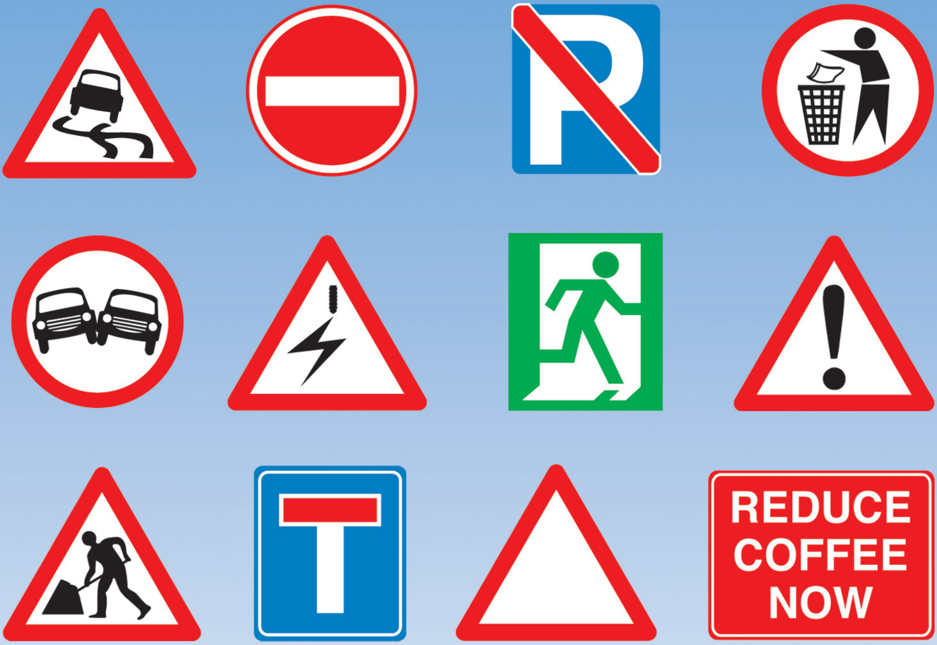
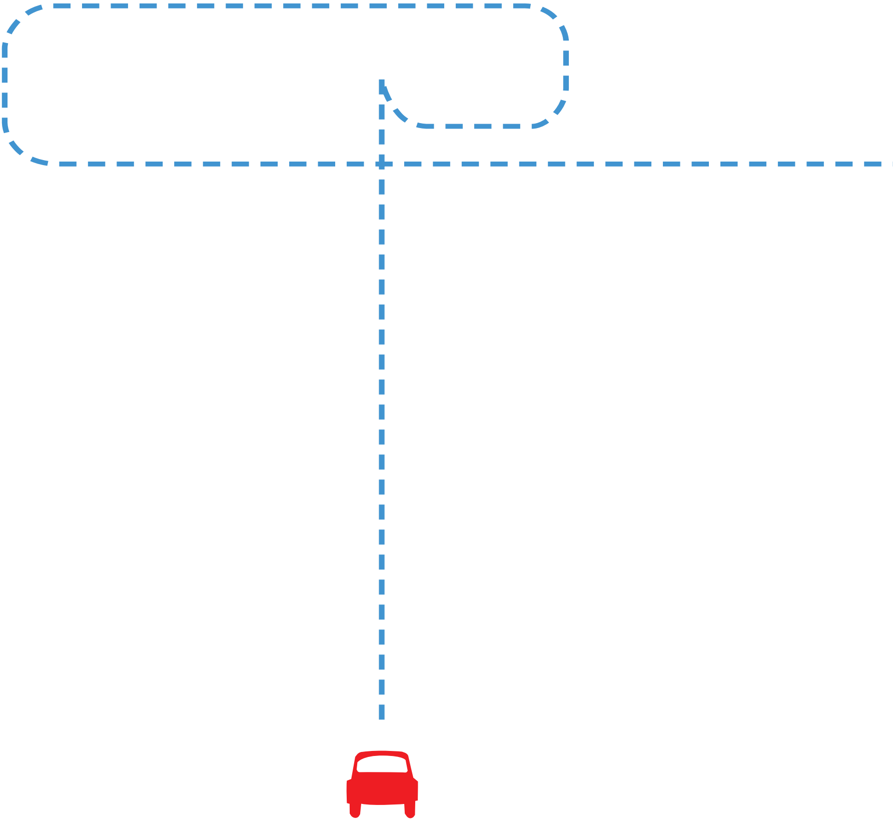
THE DEVELOPER HIGHWAY CODE



The drive for safer coding

Microsoft | Application Security

THE DEVELOPER HIGHWAY CODE



Introduction

Welcome to the Highway Code: **The drive for safer coding!** To build software that meets your security objectives, you must integrate security activities into your software development lifecycle. This handbook is a quick reference for developers that summarises the key security engineering activities that should be an integral part of software development processes. These security engineering activities have been developed by Microsoft *patterns & practices* to build on, refine and extend core lifecycle activities with a set of security-specific activities. This handbook provides a snapshot view of the steps necessary to perform each activity, references for additional reading about each activity, and a comprehensive set of security checklists that you can use as job aids while developing your software.

Audience

This handbook provides security activity guidance, checklists and question lists for application architects and software developers who want to improve the security of the applications that they develop. Software developers are the primary audience, but the security engineering activities that this handbook summarises are designed to be used by team members from many different disciplines, including business analysts, architects, developers, testers, security analysts and administrators. The handbook is task-based and is centered on key security activities that you should perform at the various stages of the application lifecycle. The question lists and checklists in Part II of the handbook are job aids and quick reference sheets that software developers should use when designing and implementing solutions.

Contents

Part I: Securing Engineering

This part presents an overview of key security engineering activities that should be an integral part of your application development lifecycle.

Cont...

Part I

Securing Engineering

This part presents an overview of key security engineering activities that should be an integral part of your application development lifecycle.

Module 1: Integrating Security into the Lifecycle
Module 2: Security Objectives
Module 3: Web Application Security Design Guidelines
Module 4: Threat Modelling
Module 5: Security Architecture and Design Review
Module 6: Security Code Review
Module 7: Security Deployment Review

Module 1: Integrating Security into the Lifecycle

Summary

To meet your application security objectives, you must integrate security into your application development lifecycle. You can do so by including specific security-related activities in your current software engineering processes. These activities include identifying security objectives, applying secure design guidelines, patterns & principles, creating threat models, conducting architecture and design reviews for security, performing regular code reviews for security, testing for security and conducting deployment reviews to ensure secure configuration.

For more information, see http://msdn.com/securityengineering

Lifecycle Integration

	Core	Security
Planning		
Requirements and Analysis	Functional Requirements Non Functional Requirements Technology Requirements	**Security Objectives**
Architecture and Design	Design Guidelines Architecture and Design Review	**Security Design Guidelines** **Threat Modelling** **Security Architecture and Design Review**
Development	Unit Tests Code Review Daily Builds	**Security Code Review**
Testing	Integration Testing System Testing	**Security Testing**
Deployment	Deployment Review	**Security Deployment Review**
Maintenance		

Security Activities

Activity	Description
Security Objectives	Define security objectives and requirements at the start of the lifecycle. Security objectives are goals and constraints that affect the confidentiality, integrity and availability of your data and application.
Security Design Guidelines	Use proven design practices, patterns and principles during design. Organise design patterns and practices into common categories to focus on those areas where security mistakes are most often made.
Threat Modelling	Use threat modelling to understand and identify the threats and vulnerabilities relevant to your application.
Security Architecture and Design Review	Analyse the architecture and design of your application from a security perspective. Examine deployment, infrastructure and the approach taken across each tier of your application.
Security Code Review	Inspect code to identify security vulnerabilities. Perform the activity continuously during the development and test phases.
Security Testing	Use a risk-based approach and use the output from the threat modelling activity to help establish the scope of your testing activities and define your test plans.
Security Deployment Review	Review deployment configuration to ensure that weak or inappropriate configuration settings do not introduce security vulnerabilities.

Conflict error

Module 2: Security Objectives

Summary

Security objectives and requirements should be defined early in the application development process. Security objectives are goals and constraints that affect the confidentiality, integrity and availability of your data and application. If you do not know what the objectives are for your application then it is difficult to be successful with any other security activity. You use security objectives to filter the set of design guidelines that are applicable, guide threat modelling activities, determine the scope and guide the process of architecture and design reviews, help set code review objectives, guide security test planning and execution, and guide deployment reviews.

Types of Security Objectives

Security objectives are unique for each application. However, a set of common categories of objectives that you can use to jump-start the identification process is shown in the following table.

Objective Category	Questions to Ask
Tangible Assets to Protect	• Are there user accounts and passwords to protect? • Is there confidential user information (such as credit card numbers) that needs to be protected? • Is there sensitive intellectual property that needs to be protected? • Can this system be used as a conduit to access other corporate assets that need to be protected?
Intangible Assets to Protect	• Are there corporate values that could be compromised by an attack on this system? • Is there potential for an attack that may be embarrassing, although not otherwise damaging?
Compliance Requirements	• Are there corporate security policies that must be adhered to? • Is there security legislation you must comply with? • Is there privacy legislation you must comply with? • Are there standards you must adhere to? • Are there constraints forced upon you by your deployment environment?
Quality of Service Requirements	• Are there specific availability requirements you must meet? • Are there specific performance requirements you must meet?

Module 3: Web Application Security Design Guidelines

Summary

Web applications present a complex set of security issues for architects, designers and developers. The most secure and hack-resilient Web applications are those that have been built from the ground up with security in mind. In addition to applying sound architectural and design practices, incorporate deployment considerations and corporate security policies during the early design phases. Failure to do so can result in applications that cannot be deployed on an existing infrastructure without compromising security.

For more information, see http://msdn.microsoft.com/library/en-us/dnnetsec/html/THCMCh04.asp

Web Application Security Frame

Security design guidelines can be organised using a pattern-based information model that defines a set of security related categories specifically for the application type you are designing. These categories represent the areas where security mistakes are most often made.

The following table identifies the key questions you need to consider while designing your Web application. These are organised by the Web application security frame.

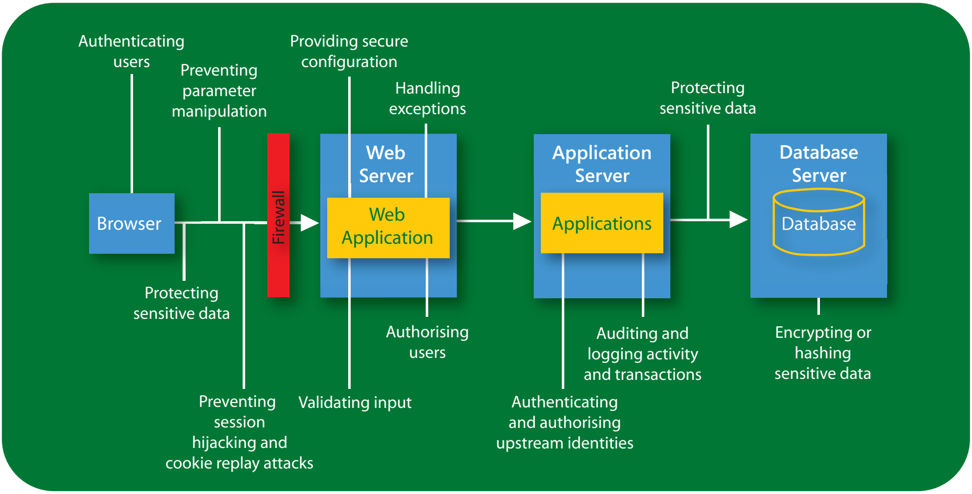
Category	Description
Input / Data Validation	How do you know that the input your application receives is valid and safe? Input validation refers to how your application filters, scrubs or rejects input before additional processing.
Authentication	Who are you? Authentication is the process where an entity proves the identity of another entity, typically through credentials, such as a username and password.
Authorisation	What can you do? Authorisation is how your application provides access controls for resources and operations.

Cont...

Category	Description
Configuration Management	Who does your application run as? Which databases does it connect to? How is your application administered? How are these settings secured? Configuration management refers to how your application handles these operational issues.
Sensitive Data	How does your application handle sensitive data? Sensitive data refers to how your application handles any data that must be protected either in memory, over the network or in persistent stores.
Session Management	How does your application handle and protect user sessions? A session refers to a series of related interactions between a user and your Web application.
Cryptography	How are you keeping secrets (confidentiality)? How are you tamper-proofing your data or libraries (integrity)? How are you providing seeds for random values that must be cryptographically strong? Cryptography refers to how your application enforces confidentiality and integrity.
Exception Management	When a method call in your application fails, what does your application do? How much do you reveal? Do you return friendly error information to end users? Do you pass valuable exception information back to the caller? Does your application fail gracefully?
Auditing and Logging	Who did what and when? Auditing and logging refers to how your application records security-related events.

Server connection error

Top Web Application Security Issues

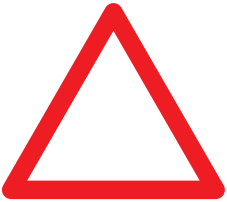

Authenticating users
Preventing parameter manipulation
Providing secure configuration
Handling exceptions
Protecting sensitive data
Browser
Firewall
Web Server
Web Application
Application Server
Applications
Database Server
Database
Protecting sensitive data
Authorising users
Auditing and logging activity and transactions
Encrypting or hashing sensitive data
Preventing session hijacking and cookie replay attacks
Validating input
Authenticating and authorising upstream identities

Security Design Guidelines Summary

Category	Guidelines
Input / Data Validation	• Do not trust input including form fields, cookies, query strings, HTTP headers • Consider centralised input validation • Do not rely on client-side validation • Be careful with canonicalisation issues • Constrain, reject and sanitise input • Validate for type, length, format and range
Authentication	• Partition site by anonymous, identified and authenticated area • Use strong passwords • Support password expiry periods and account disablement • Do not store credentials (use one-way hashes with salt) • Encrypt communication channels to protect authentication tokens • Pass forms authentication cookies only over HTTPS connections
Authorisation	• Use least privileged accounts • Consider authorisation granularity • Enforce separation of privileges • Restrict user access to system-level resources

Cont...

Category	Guidelines
Configuration Management	• Use least privileged process and service accounts • Do not store credentials in plaintext • Use strong authentication and authorisation on administration interfaces • Do not use the LSA • Secure the communication channel for remote administration • Avoid storing sensitive data in the Web space
Sensitive Data	• Avoid storing secrets • Encrypt sensitive data over the wire • Secure the communication channel • Provide strong access controls on sensitive data stores • Do not store sensitive data in persistent cookies • Do not pass sensitive data using the HTTP-GET protocol
Session Management	• Limit the session lifetime • Secure the channel • Encrypt the contents of authentication cookies • Protect session state from unauthorised access
Cryptography	• Do not develop your own cryptography; use tried and tested platform features • Keep unencrypted data close to the algorithm • Use the right algorithm and key size • Avoid key management where possible (use DPAPI) • Cycle your keys periodically • Store keys in a restricted location
Exception Management	• Use structured exception handling • Do not reveal sensitive application implementation details • Do not log private data such as passwords • Consider a centralised exception management framework
Auditing and Logging	• Identify malicious behaviour • Know what good traffic looks like • Audit and log activity through all of the application tiers • Restrict access to log files • Back up and regularly analyse log files



Missing character

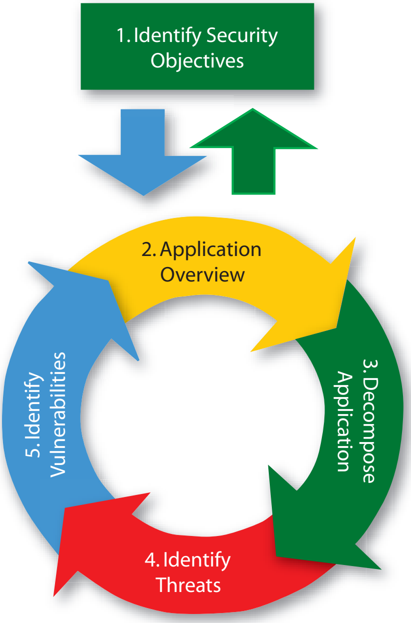
Module 4: Threat Modelling

Summary

Threat modelling is an engineering technique you can use to help you identify threats, attacks, vulnerabilities, and countermeasures relevant to your application. Use threat modelling to shape your application design to meet your security objectives, help make trade-offs during key engineering decisions, reduce risk of security issues arising during development and operations.

For more information, see http://msdn.com/threatmodelling.

Threat Modelling Activity



Threat Modelling Steps

The five threat modelling steps are:

- **Step 1: Identify security objectives.** Clear objectives help you to focus the threat modelling activity and determine how much effort to spend on subsequent steps.
- **Step 2: Create an application overview.** Itemizing your application's important characteristics and actors helps you to identify relevant threats during step 4.
- **Step 3: Decompose your application.** A detailed understanding of the mechanics of your application makes it easier for you to uncover more relevant and more detailed threats.
- **Step 4: Identify threats.** Use details from steps 2 and 3 to identify threats relevant to your application scenario and context.
- **Step 5: Identify vulnerabilities.** Review the layers of your application to identify weaknesses related to your threats. Use vulnerability categories to help you focus on those areas where mistakes are most often made.

Threat Modelling Explained

Threat modelling is an engineering discipline you can use to help you identify threats, attacks, vulnerabilities and countermeasures that are relevant to your particular application and its environment. The threat modelling activity helps you to:

- **Identify your security objectives.**
- **Identify relevant threats.**
- **Identify relevant vulnerabilities and countermeasures.**

It also helps you to address questions like these:
How do you know whether your security efforts are being directed in the right areas? How do you know that the most appropriate security technique is being used? How do you know which threats you should care most about? How do you measure security? Just how secure is your application?

To answer these types of questions you need a formal, repeatable and structured approach to application security and this is what threat modelling offers you. Without threat modelling, application security tends to be applied in an ad-hoc, haphazard manner (often using a 'scattergun' approach) and you can have little idea about the threats that your security work has addressed and which threats you might have failed to consider altogether.

Threat modelling forces you to start by considering the overall security objectives for your application. These are usually derived from business requirements. You then use threat modelling to shape your application design to meet your security objectives. It helps you to make the right trade-offs during key engineering decisions and helps to ensure that you focus your security efforts in the right areas. Threat modelling is fundamentally about reducing the risk of security issues arising during development and operations of your application.

The *patterns & practices* approach to threat modelling has been designed with developers (and not security experts) in mind. The set of resources on MSDN should help you to speed up your learning curve so that you can become productive with threat modelling very quickly. It makes use of a pattern-based information model that you can use to identify the patterns of repeatable problems and solutions and organise them into categories. This information model is referred to as the *application security frame*. The threat modelling resources on MSDN include a cheat sheet **(http://msdn2.microsoft.com/en-us/library/ms978518.aspx)** that provides groups of vulnerabilities, threats, attacks and countermeasures organised by the security frame. The threat lists are a great way to help kick-start the threat identification process.

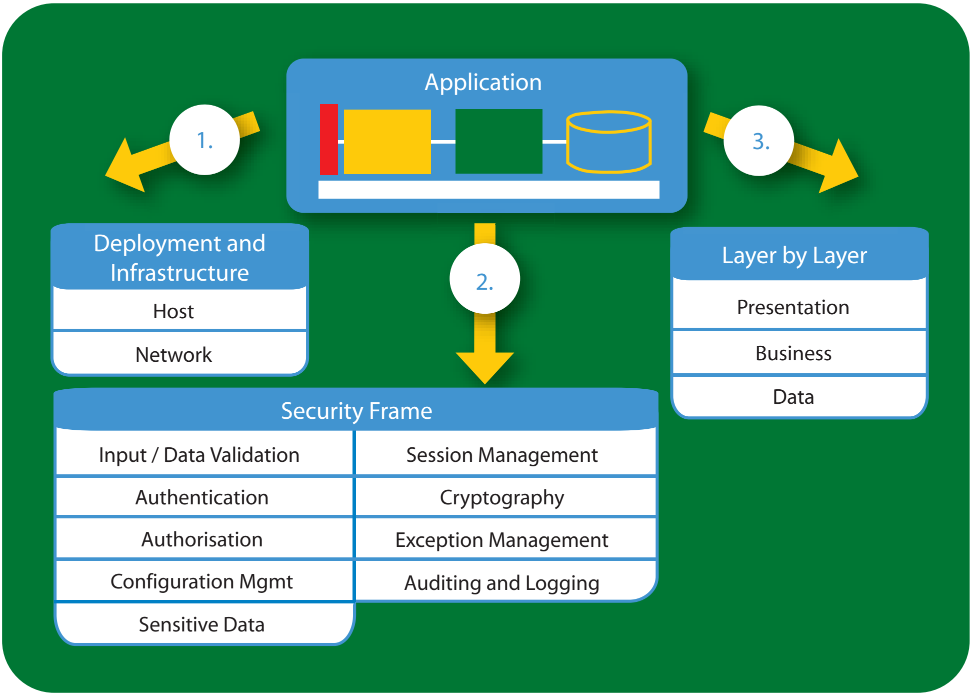
Module 5: Security Architecture and Design Review

Summary

The architecture and design review process analyses the architecture and design from a security perspective. If you have just completed the design, the design documentation can help you with this process. Regardless of how comprehensive your design documentation is, you must be able to decompose your application and be able to identify key items, including trust boundaries, data flow, entry points and privileged code. You must also know the physical deployment configuration of your application. Pay attention to the design approaches you have adopted for those areas that most commonly exhibit vulnerabilities.

For more information,
see http://msdn.microsoft.com/library/en-us/dnnetsec/html/THCMCh05.asp

Security Architecture and Design Review Overview



Techniques

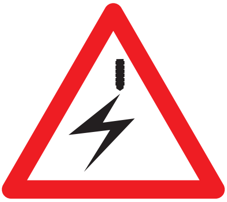
Use a question driven approach to expose the highest risk design decisions and use the security frame to dive into areas that reveal common mistakes. The following techniques can help to guide your approach when reviewing the architecture and design of your application:

1. **Deployment and infrastructure.** Review the design of your application in relation to the target deployment environment and the associated security policies. Consider the restrictions imposed by the underlying infrastructure-layer security.

2. **Security frame.** Review the approach to critical areas in your application, including authentication, authorisation, input/data validation, exception management and other areas. Use the application vulnerability categories, defined by the security frame, as a roadmap and to make sure that you do not miss any key areas during the review.

3. **Layer by layer analysis.** Walk through the logical layers of your application and examine the security of ASP.NET Web pages and controls, Web services, serviced components, Microsoft .NET Remoting, data access code and others.

Use checklists to help you perform architecture and design reviews while evaluating the security of your applications. The checklist should help you explore the high level design and architecture decisions that have been made for your application. You should evolve your checklists to include custom checks based on the unique aspects of your application's architecture.



Lightning coder

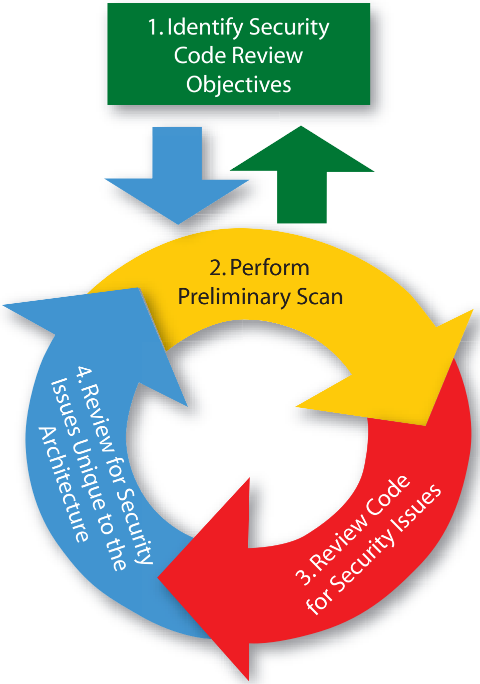
Module 6: Security Code Review

Summary

Security code review is an effective mechanism for uncovering security bugs before testing or deployment begins. Performing code reviews helps you reduce the number of implementation errors in an application before it is deployed to a test team or to a customer. While design bugs are the most expensive to fix, implementation bugs are the most common.

For more information, see
http://msdn.microsoft.com/library/en-us/dnpag2/html/SecurityCodeReviewIndex.asp

Security Code Review Activity


1. Identify Security Code Review Objectives
2. Perform Preliminary Scan
3. Review Code for Security Issues
4. Review for Security Issues Unique to the Architecture

Code Review Steps

The four code review steps are:

- **Step 1: Identify security code review objectives.** Establish goals and constraints for the review.
- **Step 2: Perform a preliminary scan.** Use static analysis to find an initial set of bugs and improve your understanding of where bugs are most likely to be discovered during further review.
- **Step 3: Review the code for security issues.** Review the code thoroughly to find security vulnerabilities that are common to many applications. You can use the results of step 2 to focus your analysis.
- **Step 4: Review for security issues unique to the architecture.** Complete a final analysis that focuses on bugs that relate to the unique architecture of your application. This step is most important if you have implemented a custom security mechanism or any feature designed specifically to mitigate a known security threat.

Question Lists

Using a question-driven approach can help with the review process. Question lists to help you review .NET code and ASP.NET Web application code are provided in Part II, Checklists and Question Lists on page 22.

12 hours' coding
– need a coffee

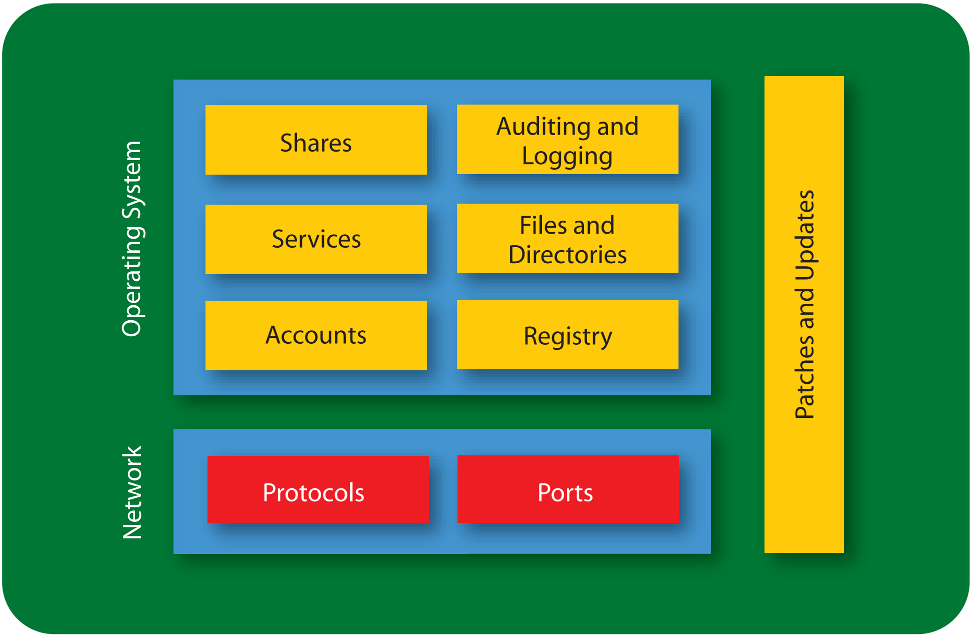
Module 7: Security Deployment Review

Summary

A security deployment review is an activity that can be used to ensure that configuration and deployment problems are discovered before they can result in an application vulnerability. Even the most securely designed and implemented application can be compromised by an error during deployment, leaving it open to attack. When you review your security deployment, you can organise the precautions you must take and the settings you must configure into categories. By using configuration categories, you can systematically review the entire application, or pick a particular category and complete specific steps.

For more information, see http://msdn.microsoft.com/library/en-us/dnnetsec/html/THCMCh22.asp

Server Security Categories



Server Security Categories Explained

Category	Practices
Patches and Updates	Patching and updating your server software is a critical first step.
Accounts	Enforce strong password policies. Audit your accounts and remove any you do not need. Configure accounts with least privilege.
Auditing and Logging	Use auditing and alerts to detect logon failures for identifying intruders, attacks in progress, and evidence of attacks that have occurred. Configure auditing for your server.
Files and Directories	Secure all files and directories with restricted permissions that only allow access to necessary accounts. Use auditing to allow you to detect when suspicious or unauthorised activity occurs.
Ports	Services that run on the server listen to specific ports so that they can respond to incoming requests. Audit the ports on your server regularly to ensure that a service that is not secured or that is unnecessary is not active on your server.
Protocols	Avoid using protocols that are inherently insecure. If you cannot avoid using these protocols, take the appropriate measures to provide secure authentication and communication.
Registry	Protect access to the registry. Apply restricted Microsoft Windows® access control lists (ACLs) and block remote registry administration.
Services	If the service is necessary, secure and maintain the service. Consider monitoring any service to ensure availability. If your service software is not secure, but you need the service, try to find a secure alternative.
Shares	Remove all unnecessary file shares. Secure any remaining shares with restricted permissions.

Warning – unstable coding ahead

Techniques

Use the following techniques when conducting a deployment review:

1. **Use server security categories.** Use server security categories to help make security deployment reviews systematic and repeatable.

2. **Break down your deployment review.** You can use the categories to break down your application deployment for further analysis and to help identify vulnerabilities.

3. **Review systematically.** By using categories, you can systematically go through the deployment review process from start to finish or pick a particular category for further analysis.

Part II

Checklists and Question Lists

This part includes question lists and checklists to help you design, build and deploy software that meets your security objectives.

.NET Framework 1.1 Checklists

- Checklist: Web Application Architecture and Design
- Security Checklist: .NET Framework 1.1
- Security Checklist: ADO.NET 1.1
- Security Checklist: ASP.NET 1.1
- Security Checklist: Enterprise Services (.NET Framework 1.1)
- Security Checklist: Remoting (.NET Framework 1.1)
- Security Checklist: Web Services (.NET Framework 1.1)
- Security Checklist: Network Security
- Security Checklist: Web Server (IIS 5.1)
- Security Checklist: Database Server (SQL Server 2000)

.NET Framework 2.0 Checklists

- Security Checklist: ASP.NET version 2.0
- Security Checklist: .NET Framework version 2.0
- Security Checklist: ADO.NET 2.0

.NET Framework 3.0 Checklists

- Security Checklist: Windows Communication Foundation

Native Code Checklists

- Security Checklist: Native Code Security

Question Lists for Conducting Security Code Reviews

- .NET Framework 2.0 Question List
- ASP.NET 2.0 Question List

Checklist:

Web Application Architecture and Design

Checklist: Web Application Architecture and Design

How To Use This Module

This checklist is a companion to Chapter 4 'Design Guidlines for Secure Web Applications' and Chapter 5, 'Architecture and Design Review for Security' from 'Improving Web Application Security: Threats and Countermeasures' on MSDN® at **http://msdn.com/secnet** - use it to help you perform architecture and design reviews to evaluate the security of your Web applications and to implement the design guidelines in Chapter 4.

This checklist should evolve based on the experience you gain from performing reviews. You might also want to perform custom checks that are based on a specific aspect of your architecture or design to ensure that your development environment is catered for by the design.

Deployment and Infrastructure Considerations

Check	Description
▢	The design identifies, understands and accommodates the company security policy.
▢	Restrictions imposed by infrastructure security (including available services, protocols and firewall restrictions) are identified.
▢	The design recognises and accommodates restrictions imposed by hosting environments (including application isolation requirements).
▢	The target environment code-access-security trust level is known.
▢	The design identifies the deployment infrastructure requirements and the deployment configuration of the application.
▢	Domain structures, remote application servers and database servers are identified.
▢	The design identifies clustering requirements.
▢	The design identifies the application configuration maintenance points (such as what needs to be configured and what tools are available for an IDC admin).
▢	Secure communication features provided by the platform and the application are known.
▢	The design addresses Web farm considerations (including session state management, machine specific encryption keys, Secure Sockets Layer (SSL), certificate deployment issues and roaming profiles).
▢	The design identifies the Certificate Authority (CA) to be used by the site to support SSL.
▢	The design addresses the required scalability and performance criteria.

Application Architecture and Design Considerations

Input Validation

Check	Description
□	All entry points and trust boundaries are identified by the design.
□	Input validation is applied whenever input is received from outside the current trust boundary.
□	The design assumes that user input is malicious.
□	Centralised input validation is used where appropriate.
□	The input validation strategy that the application adopted is modular and consistent.
□	The validation approach is to constrain, reject and then sanitise input. (Looking for known, valid and safe input is much easier than looking for known malicious or dangerous input.)
□	Data is validated for type, length, format and range.
□	The design addresses potential canonicalisation issues.
□	Input filenames and file paths are avoided where possible.
□	The design addresses potential SQL injection issues.
□	The design addresses potential cross-site scripting issues.
□	The design does not rely on client-side validation.
□	The design applies defence in-depth to the input validation strategy by providing input validation across tiers.
□	Output that contains input is encoded using HtmlEncode and UrlEncode.

Authentication

Check	Description
▢	Application trust boundaries are identified by the design.
▢	The design identifies the identities that are used to access resources across the trust boundaries.
▢	The design partitions the Web site into public and restricted areas using separate folders.
▢	The design identifies service account requirements.
▢	The design identifies secure storage of credentials that are accepted from users.
▢	The design identifies the mechanisms to protect the credentials over the wire (SSL, IPSec, encryption and so on).
▢	Account management policies are taken into consideration by the design.
▢	The design ensures that minimum error information is returned in the event of authentication failure.
▢	The identity that is used to authenticate with the database is identified by the design.
▢	If SQL authentication is used, credentials are adequately secured over the wire (SSL or IPSec) and in storage (DPAPI).
▢	The design adopts a policy of using least-privileged accounts.
▢	Password digests (with salt) are stored in the user store for verification.
▢	Strong passwords are used.
▢	Authentication tickets (cookies) are not transmitted over non-encrypted connections.

Conflict error

Authorisation

Check	Description
○	The role design offers sufficient separation of privileges (the design considers authorisation granularity).
○	Multiple gatekeepers are used for defence in-depth.
○	The application's login is restricted in the database to access-specific stored procedures.
○	The application's login does not have permissions to access tables directly.
○	Access to system level resources is restricted.
○	The design identifies code access security requirements. Privileged resources and privileged operations are identified.
○	All identities that are used by the application are identified and the resources accessed by each identity are known.

Configuration Management

Check	Description
○	Administration interfaces are secured (strong authentication and authorisation is used).
○	Remote administration channels are secured.
○	Configuration stores are secured.
○	Configuration secrets are not held in plaintext in configuration files.
○	Administrator privileges are separated based on roles (for example, site content developer or system administrator).
○	Least-privileged process accounts and service accounts are used.

Sensitive Data

Check	Description
□	Secrets are not stored unless necessary. (Alternative methods have been explored at design time.)
□	Secrets are not stored in code.
□	Database connections, passwords, keys or other secrets are not stored in plaintext.
□	The design identifies the methodology to store secrets securely. (Appropriate algorithms and key sizes are used for encryption. It is preferable that DPAPI is used to store configuration data to avoid key management.)
□	Sensitive data is not logged in clear text by the application.
□	The design identifies protection mechanisms for sensitive data that is sent over the network.
□	Sensitive data is not stored in persistent cookies.
□	Sensitive data is not transmitted with the GET protocol.

Session Management

Check	Description
□	SSL is used to protect authentication cookies.
□	The contents of authentication cookies is encrypted.
□	Session lifetime is limited.
□	Session state is protected from unauthorised access.
□	Session identifiers are not passed in query strings.

Cryptography

Check	Description
□	Platform-level cryptography is used and it has no custom implementations.
□	The design identifies the correct cryptographic algorithm (and key size) for the application's data encryption requirements.
□	The methodology to secure the encryption keys is identified.
□	The design identifies the key recycle policy for the application.
□	Encryption keys are secured.
□	DPAPI is used where possible to avoid key management issues.
□	Keys are periodically recycled.

Parameter Manipulation

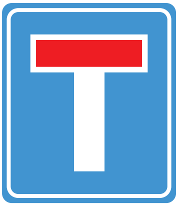
Check	Description
□	All input parameters are validated (including form fields, query strings, cookies and HTTP headers).
□	Cookies with sensitive data are encrypted.
□	Sensitive data is not passed in query strings or form fields.
□	HTTP header information is not relied on to make security decisions.
□	View state is protected using MACs.

Exception Management

Check	Description
□	The design outlines a standardised approach to structured exception handling across the application.
□	Application exception handling minimises the information disclosure in case of an exception.
□	The design identifies generic error messages that are returned to the client.
□	Application errors are logged to the error log.
□	Private data (for example, passwords) is not logged.

Auditing and Logging

Check	Description
□	The design identifies the level of auditing and logging necessary for the application and identifies the key parameters to be logged and audited.
□	The design considers how to flow caller identity across multiple tiers (at the operating system or application level) for auditing.
□	The design identifies the storage, security and analysis of the application log files.



Server connection error

Security Checklist:

.NET Framework 1.1

Security Checklist: .NET Framework 1.1

How To Use This Module

This checklist is a companion to Chapter 7, 'Building Secure Assemblies' and Chapter 8, 'Code Access Security in Practice' from 'Improving Web Application Security: Threats and Countermeasures' on MSDN at **http://msdn.com/secnet** – use it to help you implement a security review for managed code in your ASP.NET 1.1 Web application, or as a quick evaluation snapshot of the corresponding chapters.

Use this checklist as you develop your managed code. You should expand and evolve this security checklist by adding managed code practices that you discover during software development.

General Code Review Guidelines

Check	Description
□	Potential threats are clearly documented. (Threats are dependent upon the specific scenario and assembly type.)
□	Code is developed based on .NET Framework coding guidelines and secure coding guidelines at **http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp**
□	The FXCop analysis tool is run on assemblies and security warnings are addressed.

Managed Code Review Guidelines

Assembly-Level Checks

Check	Description
□	Assemblies have a strong name. (Dynamically generated ASP.NET Web page assemblies cannot currently have a strong name.)
□	You have considered delay signing as a way to protect and restrict the private key that is used in the strong name and signing process.
□	Assemblies include declarative security attributes (with **SecurityAction.RequestMinimum**) to specify minimum permission requirements.
□	Highly privileged assemblies are separated from lower privileged assemblies. If the assembly is to be used in a partial-trust environment (for example, it is called from a partial-trust Web application), then privileged code is sandboxed in a separate assembly.

Class-Level Checks

Check	Description
□	Class and member visibility is restricted. The most restrictive access modifier is used (private where possible).
□	Non-base classes are sealed if they contain security secrets (like passwords) accessible through protected APIs or if they contain many virtual members that cannot be sealed and the type is not really designed for third-party extensibility.
□	Input from outside the current trust boundary is validated. Input data is constrained and validated for type, length, format and range.
□	Code implements declarative checks where virtual internal methods are used.
□	Access to public classes and methods are restricted with principal permission demands (where appropriate).
□	Fields are private. When necessary, field values are exposed by using read/write or read-only public properties.
□	Read-only properties are used where possible.
□	Types returned from methods that are not designed to be created independently contain private default constructors.
□	Unsealed public types do not have internal virtual members.
□	Use of event handlers is thoroughly reviewed.
□	Static constructors are private.

Cryptography

Check	Description
□	Code uses platform-provided cryptography and does not use custom implementations.
□	Random keys are generated by using **RNGCryptoServiceProvider** (and not the Random class).
□	**PasswordDeriveBytes** is used for password-based encryption.
□	DPAPI is used to encrypt configuration secrets to avoid the key management issue.
□	The appropriate key sizes are used for the chosen algorithm, or, if they are not, the reasons are identified and understood.
□	Keys are not held in code.
□	Access to persisted keys is restricted.
□	Keys are cycled periodically.
□	Exported private keys are protected.

Secrets

Check	Description
□	Secrets are not hard coded.
□	Plaintext secrets are not stored in configuration files.
□	Plaintext secrets are not stored in memory for extended periods of time.

Missing character

Exception Management

Check	Description
□	Code uses exception handling. You catch only the exceptions that you know about.
□	Exception details are logged on the server to assist in diagnosing problems.
□	The information that is returned to the end user is limited and safe.
□	Code that uses exception filters is not sensitive to filter execution sequence (filter runs before finally block).
□	Code fails early to avoid unnecessary processing that consumes resources.
□	Exception conditions do not allow a user to bypass security checks to run privileged code.

Delegates

Check	Description
□	Delegates are not accepted from untrusted sources.
□	If code does accept a delegate from untrusted code, it constrains the delegate before calling it by using security permissions with **SecurityAction.PermitOnly**.
□	Permissions are not asserted before calling a delegate.

Serialisation

Check	Description
□	Serialisation is restricted to privileged code.
□	Sensitive data is not serialised.
□	Field data from serialised data streams is validated.
□	**ISerializable.GetObjectData** implementation is protected with an identity permission demand in scenarios where you want to restrict which code can serialise the object.

Threading

Check	Description
□	Results of security checks are not cached.
□	Impersonation tokens are considered when new threads are created (any existing thread token is not passed to the new thread).
□	Threads are synchronised in static class constructors for multi-threaded application code. Object implementation code is designed and built to be thread safe.
□	Threads are synchronised in static class constructors.

Reflection

Check	Description
□	Caller cannot influence dynamically generated code (for example, by passing assembly and type names as input arguments).
□	Code demands permission for user authorisation where assemblies are loaded dynamically.

Late home again = unhappy girlfriend

Unmanaged Code Access

Check	Description
□	Input and output strings that are passed between managed and unmanaged code are constrained and validated.
□	Array bounds are checked.
□	File path lengths are checked and do not exceed MAX_PATH.
□	Unmanaged code is compiled with the /GS switch.
□	Use of 'dangerous' APIs by unmanaged code is closely inspected. These include LogonUser, RevertToSelf, CreateThread, Network APIs and Sockets APIs.
□	Naming conventions (safe, native, unsafe) are applied to unmanaged APIs.
□	Assemblies that call unmanaged code specify unmanaged permission requirements using declarative security **(SecurityAction.RequestMinimum)**.
□	Unmanaged API calls are sandboxed and isolated in a wrapper assembly.
□	Use of **SuppressUnmanagedCodeSecurityAttribute** is thoroughly reviewed and additional security checks are implemented.
□	Types are not annotated with **SuppressUnmanagedCodeSecurityAttribute**. (This attribute is used on specific P/Invoke method declarations instead.)
□	Calling code is appropriately authorised using a full stack walk Demand (using either a .NET Framework permission or custom permission). Unmanaged types or handles are never exposed to partially trusted code.
□	Pointers are private fields.
□	Methods that use **IntPtr** fields in a type that has a finaliser, call **GC.KeepAlive(object)**.

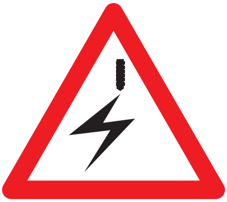
Resource Access Considerations

File I/O

Check	Description
□	No security decisions are made based on filenames.
□	Input file paths and filenames are well formed.
□	Environment variables are not used to construct file paths.
□	File access is constrained to the context of the application (by using a restricted FileIOPermission).
□	Assembly file I/O requirements are specified using declarative security attributes (with **SecurityAction.RequestMinimum**).

Event Log

Check	Description
□	Event log access code is constrained using **EventLogPermission**. This particularly applies if your event logging code could be called by untrusted callers.
□	Event sources are created at installation time (or the account used to run the code that writes to the event log must be allowed to create event sources by configuring an appropriate ACL in the registry).
□	Security-sensitive data, such as passwords, is not written to the event log.



Lightning coder

Registry

Check	Description
□	Sensitive data, such as database connection strings or credentials, is encrypted prior to storage in the registry.
□	Keys are restricted. If a key beneath HKEY_CURRENT_MACHINE is used, the key is configured with a restricted ACL. Alternatively, HKEY_CURRENT_USER is used.
□	Registry access is constrained by using **RegistryPermission**. This applies especially if your registry access code could be called by untrusted callers.

Environment Variables

Check	Description
□	Code that accesses environment variables is restricted with **EnvironmentPermission**. This applies especially if your code can be called by untrusted code.
□	Environment permission requirements are declared by using declarative security attributes with **SecurityAction.RequestMinimum**.

Code Access Security Considerations

If an entry is preceded by an asterisk (*), it indicates that the checks are performed by the FXCop analysis tool.

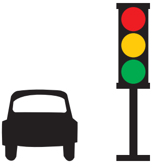
For more information about FXCop security checks, see www.gotdotnet.com/team/libraries/FxCopRules/SecurityRules.aspx

Check	Description
▢	Assemblies marked with **AllowPartiallyTrustedCallersAttribute (APTCA)** do not expose objects from non-APTCA assemblies.
▢	Code that only supports full-trust callers is strong named or explicitly demands the full-trust permission set.
▢	All uses of **Assert** are thoroughly reviewed.
▢	All calls to **Assert** are matched with a corresponding call to **RevertAssert**.
▢	*The Assert window is as small as possible.
▢	*Asserts are proceded with a full permission demand.
▢	*Use of **Deny** or **PermitOnly** is thoroughly reviewed.
▢	All uses of **LinkDemand** are thoroughly reviewed. (Why is a LinkDemand and not a full Demand used?)
▢	LinkDemands within Interface declarations are matched by LinkDemands on the method implementation.
▢	*Unsecured members do not call members protected by a LinkDemand.
▢	Permissions are not demanded for resources accessed through the .NET Framework classes.
▢	Access to custom resources (through unmanaged code) is protected with custom code access permissions.
▢	Access to cached data is protected with appropriate permission demands.
▢	If LinkDemands are used on structures, the structures contain explicitly defined constructors.

12 hours' coding – need a coffee

Cont...

Check	Description
□	*Methods which override other methods that are protected with LinkDemands also issue the same LinkDemand.
□	*LinkDemands on types are not used to protect access to fields inside those types.
□	*Partially trusted methods call only other partially trusted methods.
□	*Partially trusted types extend only other partially trusted types.
□	*Members that call late bound members have declarative security checks.
□	*Method-level declarative security does not mistakenly override class-level security checks.
□	**Use of the following 'potentially dangerous' permissions is thoroughly reviewed:** SecurityPermission Unmanaged Code SkipVerification ControlEvidence ControlPolicy SerialisationFormatter ControlPrincipal ControlThread ReflectionPermission MemberAccess
□	Code identity permission demands are used to authorise calling code in scenarios where you know in advance the range of possible callers (for example, you want to limit calling code to a specific application).
□	Permission demands of the .NET Framework are not duplicated.
□	Inheritance is restricted with **SecurityAction.InheritanceDemand** in scenarios where you want to limit which code can derive from your code.



Security Checklist:

ADO.NET 1.1

Security Checklist: ADO.NET 1.1

How To Use This Module

This checklist is a companion to Chapter 14, 'Building Secure Data Access' and Chapter 16, 'Securing Your Database Server' from 'Improving Web Application Security: Threats and Countermeasures' on MSDN at **http://msdn.com/secnet** – use it to help you build secure data access, or as a quick evaluation snapshot of the corresponding chapters. This checklist should evolve with secure data access practices that you discover during software development.

SQL Injection Checks

Check	Description
▢	Input passed to data access methods that originate outside the current trust boundary is constrained. Sanitisation of input is only used as a defence in-depth measure.
▢	Stored procedures that accept parameters are used by data access code. If stored procedures are not used, type safe SQL parameters are used to construct SQL commands.
▢	Least-privileged accounts are used to connect to the database.

Authentication

Check	Description
▢	Windows authentication is used to connect to the database.
▢	Strong passwords are used and enforced.
▢	If SQL Server authentication is used, the credentials are secured over the network by using IPSec or SSL, or by installing a database server certificate.
▢	If SQL Server authentication is used, connection strings are encrypted by using DPAPI and are stored in a secure location.
▢	Application connects using a least-privileged account. The **sa** account or other privileged accounts that are members of the **sysadmin** or **db_owner** roles are not used for application logins.

Authorisation

Check	Description
□	Calling users are restricted using declarative or imperative principal permission checks (normally performed by business logic).
□	Calling code is restricted using identity permission demands in scenarios where you know and want to limit the calling code.
□	Application login is restricted in the database and can only execute selected stored procedures. Application's login has no direct table access.

Configuration Management

Check	Description
□	Windows authentication is used to avoid credential management.
□	Connection strings are encrypted and encrypted data is stored securely, for example, in a restricted registry key.
□	OLE DB connection strings do not contain Persist Security Info="true" or "yes".
□	UDL files are secured with restricted ACLs.

Sensitive Data

Check	Description
□	Sensitive data is encrypted in the database using strong symmetric encryption (for example, 3DES).
□	Symmetric encryption keys are backed up and encrypted with DPAPI and stored in a restricted registry key.
□	Sensitive data is secured over the network by using SSL or IPSec.
□	Passwords are not stored in custom user store databases. Password hashes are stored with salt values instead.

Exception Management

Check	Description
□	ADO.NET exceptions are trapped and logged.
□	Database connections and other limited resources are released in case of exception or completion of operation.
□	ASP.NET is configured with a generic error page using the **<customErrors>** element.

Deployment Considerations

Check	Description
□	Firewall restrictions ensure that only the SQL Server listening port is available on the database server.
□	A method for maintaining encrypted database connection strings is defined.
□	The application is configured to use a least-privileged database login.
□	SQL Server auditing is configured. Failed login attempts are logged at minimum.
□	Data privacy and integrity over the network is provided with IPSec or SSL.

Security Checklist:
ASP.NET 1.1

Security Checklist: ASP.NET 1.1

How To Use This Module

This checklist is a companion to Chapter 10, 'Building Secure ASP.NET Pages and Controls' Chapter 19, 'Securing Your ASP.NET Application and Web Services' and Chapter 20, 'Hosting Multiple Web Applications' from 'Improving Web Application Security: Threats and Countermeasures' on MSDN at **http://msdn.com/secnet** – use it to help you secure an ASP.NET application and also as a snapshot of the corresponding chapters. You should expand and evolve this security checklist by adding additional practices that you discover during software development.

Design Considerations

Check	Description
□	Security decisions should not rely on client-side validations; they are made on the server side.
□	The Web site is partitioned into public access areas and restricted areas that require authentication access. Navigation between these areas should not flow sensitive credentials information.
□	The identities used to access remote resources from ASP.NET Web applications are clearly identified.
□	Mechanisms have been identified to secure credentials, authentication tickets, and other sensitive information over network and in persistent stores.
□	A secure approach to exception management is identified. The application fails securely in the event of exceptions.
□	The site has granular authorisation checks for pages and directories.
□	Web controls, user controls and resource access code are all partitioned in their own assemblies for granular security.

Application Categories Considerations

Input Validation

Check	Description
▢	User input is validated for type, length, format and range. Input is checked for known valid and safe data and then for malicious, dangerous data.
▢	String form field input is validated using regular expressions (for example, by the **RegularExpressionValidator** control).
▢	Regular HTML controls, query strings, cookies and other forms of input are validated using the Regex class and/or your custom validation code.
▢	The **RequiredFieldValidator** control is used where data must be entered.
▢	Range checks in server controls are checked by **RangeValidator** controls.
▢	Free form input is sanitised to clean malicious data.
▢	Input filenames are well formed and are verifiably valid within the application context.
▢	Output that includes input is encoded with HtmlEncode and UrlEncode.
▢	**MapPath** restricts cross-application mapping where appropriate.
▢	Character encoding is set by the server (ISO-8859-1 is recommended).
▢	The ASP.NET version 1.1 validateRequest option is enabled.
▢	URLScan is installed on the Web server.
▢	The **HttpOnly** cookie option is used for defence in-depth to help prevent cross-site scripting (this applies to Internet Explorer 6.1 or later).
▢	SQL parameters are used in data access code to validate length and type of data and to help prevent SQL injection.

Conflict error

Authentication

Check	Description
□	Site is partitioned to restricted areas and public areas.
□	Absolute URLs are used for navigation where the site is partitioned with secure and non-secure folders.
□	Secure Sockets Layer (SSL) is used to protect credentials and authentication cookies.
□	The **slidingExpiration** attribute is set to "false" and limited authentication cookie time-outs are used where the cookie is not protected by using SSL.
□	The forms authentication cookie is restricted to HTTPS connections by using the **requireSSL** attribute or the **Secure** cookie property.
□	The authentication cookie is encrypted and integrity checked (protection="All").
□	Authentication cookies are not persisted.
□	Application cookies have unique path/name combinations.
□	Personalisation cookies are separate from authentication cookies.
□	Passwords are not stored directly in the user store; password digests with salt are stored instead.
□	The impersonation credentials (if using a fixed identity) are encrypted in the configuration file by using **Aspnet_setreg.exe**.
□	Strong password policies are implemented for authentication.
□	The **<credentials>** element is not used inside **<forms>** element for forms authentication (use it for testing only).

Authorisation

Check	Description
□	URL authorisation is used for page and directory access control.
□	File authorisation is used with Windows authentication.
□	Principal permission demands are used to secure access to classes and members.
□	Explicit role checks are used if fine-grained authorisation is required.

Configuration Management

Check	Description
□	Configuration file retrieval is blocked by using **HttpForbiddenHandler**.
□	A least-privileged account is used to run ASP.NET.
□	Custom account credentials (if used) are encrypted on the **<processModel>** element by using **Aspnet_setreg.exe**.
□	To enforce machine-wide policy, Web.config settings are locked by using **allowOveride="false"** in **Machine.config**.

Missing probe

Sensitive Data

Check	Description
□	SSL is used to protect sensitive data on the wire.
□	Sensitive data is not passed across pages; it is maintained using server-side state management.
□	Sensitive data is not stored in cookies, hidden form fields or query strings.
□	Output caching for pages that contain sensitive data is turned off.
□	Plaintext passwords are avoided in Web.config and Machine.config files. (**Aspnet_setreg.exe** is used to encrypt credentials.)

Session Management

Check	Description
□	The session cookie is protected using SSL on all pages that require authenticated access.
□	The session state service is disabled if not used.
□	The session state service (if used) runs using a least-privileged account.
□	Windows authentication is used to connect to Microsoft SQL Server state database.
□	Access to state data in the SQL Server is restricted.
□	Connection strings are encrypted by using **Aspnet_setreg.exe**.
□	The communication channel to state store is encrypted (IPSec or SSL).

Parameter Manipulation

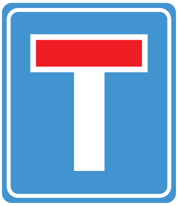
Check	Description
□	View state is protected using message authentication codes (MACs).
□	Query strings with server secrets are hashed.
□	All input parameters are validated.
□	**Page.ViewStateUserKey** is used to counter one-click attacks.

Exception Management

Check	Description
□	Structured exception handling is used.
□	Exception details are logged on the server.
□	Generic error pages with harmless messages are returned to the client.
□	Page-level or application-level error handlers are implemented.
□	The application distinguishes between errors and exception conditions.

Auditing and Logging

Check	Description
□	The ASP.NET process is configured to allow new event sources to be created at runtime, or application event sources to be created at installation time.



Server connection error

Configuration File Settings

Check	Description
□	**<trace/>** Tracing is not enabled on the production servers. `<trace enabled="false">`
□	**<globalization>** Request and response encoding is appropriately configured.
□	**<httpRuntime>** **maxRequestLength** is configured to prevent users from uploading very large files (optional).
□	**<compilation>** Debug compiles are not enabled on the production servers by setting `debug="false"<compilation debug="false" . . ./>`

□	**<pages>**
	If the application does not use view state, **enableViewState** is set to "false".
	`<pages enableViewState="false" . . ./>`
	
	If the application uses view state, **enableViewState** is set to "true" and **enableViewStateMac** is set to "true" to detect view state tampering.
	
	`<pages enableViewState="true" enableViewStateMac="true" />`

□	**<customErrors>**
	Custom error pages are returned to the client and detailed exception details are prevented from being returned by setting **mode="On"**.
	`<customErrors mode="On" />`
	
	A generic error page is specified by the defaultRedirect attribute.
	`<customErrors mode="On" defaultRedirect="/apperrorpage.htm" />`

□	**<authentication>**
	The authentication mode is appropriately configured to support application requirements. To enforce the use of a specific authentication type, a **<location>** element with **allowOverride="false"** is used.
	`<location path="" allowOverride="false">`
	`   <system.web>`
	`   <authentication mode="Windows" />`
	`   </system.web>`
	`</location>`

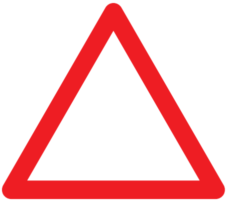
Cont...

Check	Description

□	**<forms>**
	The Web site is partitioned for public and restricted access.
	The Forms authentication configuration is secure:
	`<forms  loginUrl="Restricted\login.aspx"`
	`        protection="All"`
	`        requireSSL="true"`
	`        timeout="10"`
	`        name="AppNameCookie"`
	`        path="/FormsAuth"`
	`        slidingExpiration="true"  />`
	
	The authentication cookie is encrypted and integrity checked **(protection)**.
	SSL is required for authentication cookie **(requireSSL)**.
	Sliding expiration is set to false if SSL is not used **(slidingExpiration)**.
	The session lifetime is restricted **(time-out)**.
	Cookie names and paths are unique (**name** and **path**).
	The **<credentials>** element is not used.

□	**<identity>**
	Impersonation identities (if used) are encrypted in the registry by using **Aspnet_setreg.exe**:
	`<identity impersonate="true"`
	`   userName="registry:HKLM\SOFTWARE\YourApp\`
	`identity\ASPNET_SETREG,userName"`
	`   password="registry:HKLM\SOFTWARE\YourApp\`
	`identity\ASPNET_SETREG,password"/>`

□	**<authorisation>** Correct format of role names is verified.



Missing character

Cont...

Check	Description

□	<machineKey>
	If multiple ASP.NET Web applications are deployed on the same Web server, the “IsolateApps” setting is used to ensure that a separate key is generated for each Web application.
	`<machineKey validationKey="AutoGenerate,IsolateApps"`
	`    decryptionKey="AutoGenerate,IsolateApps"`
	`    validation="SHA1" />`
	If the ASP. NET Web application is running in a Web farm, specific machine keys are used and these keys are copied across all servers in the farm.
	If the view state is enabled, the **validation** attribute is set to "SHA1".
	The **validation** attribute is set to "3DES" if the Forms authentication cookie is to be encrypted for the application.

□	<sessionState> If **mode="StateServer"**, then credentials are stored in an encrypted form in the registry by using Aspnet_setreg.exe. If **mode="SQLServer"**, then Windows authentication is used to connect to the state store database and credentials are stored in an encrypted form in the registry by using Aspnet_setreg.exe.

□	<httpHandlers>
	Unused file types are mapped to HttpForbiddenHandler to prevent files from being retrieved over HTTP. For example:
	`<add verb="*" path="*.rem"`
	`    type="System.Web.HttpForbiddenHandler"/>`

Cont...

Check	Description

□	**<processModel>**
	A least-privileged account like ASPNET is used to run the ASP.NET process.
	`<processModel userName="Machine" password="AutoGenerate"`
	
	The system account is not used to run the ASP.NET process.
	The Act as part of the operating system privilege is not granted to the process account.
	Credentials for custom accounts are encrypted by using Aspnet_setreg.exe
	`<processModel`
	`    userName="registry:HKLM\SOFTWARE\MY_SECURE_APP\`
	`    processmodel\ASPNET_SETREG,userName"`
	`    password="registry:HKLM\SOFTWARE\MY_SECURE_APP\`
	`    processmodel\ASPNET_SETREG,password" . . ./>`
	
	If the application uses Enterprise Services, **comAuthenticationLevel** and **comImpersonationLevel** are configured appropriately.
	Call level authentication is set at minimum to ensure that all method calls can be authenticated by the remote application.
	PktPrivacy is used to encrypt and tamper-proof the data across the wire in the absence of infrastructure channel security (IPSec).
	PktIntegrity is used for tamper-proofing with no encryption (eavesdroppers with network monitors can see your data).

□	**<webServices>** Unused protocols are disabled. Automatic generation of Web Services Description Language (WSDL) is disabled (optional).

Late home again = unhappy girlfriend

Web Farm Considerations

Check	Description
▢	**Session state**. To avoid server affinity, the ASP.NET session state is maintained out of process in the ASP.NET SQL Server state database or in the out-of-process state service that runs on a remote machine.
▢	**Encryption and verification**. The keys used to encrypt and verify Forms authentication cookies and view state are the same across all servers in a Web farm.
▢	**DPAPI**. DPAPI cannot be used with the machine key to encrypt common data that needs to be accessed by all servers in the farm. To encrypt shared data on a remote server, use an alternative implementation, such as 3DES.

Hosting Multiple Applications

Check	Description

▢	Applications have distinct machine keys.
	Use **IsolateApps** on **<machineKey**> or use per application **<machineKey>** elements.
	`<machineKey validationKey="AutoGenerate,IsolateApps"`
	`   decryptionKey="AutoGenerate,IsolateApps" . . . />`

▢	Unique path/name combinations for Forms authentication cookies are enabled for each application.
▢	Multiple processes (IIS 6.0 application pools) are used for application isolation on Microsoft Windows Server® 2003.
▢	Multiple anonymous user accounts (and impersonation) are used for application isolation on Windows 2000.
▢	Common machine keys are enabled on all servers in a Web farm.
▢	Separate machine keys for each application are used when hosting multiple applications on a single server.
▢	Code access security trust levels are used for process isolation and to restrict access to system resources (requires .NET Framework version 1.1).

ACLs and Permissions

Check	Description
□	Temporary ASP.NET files %windir%\Microsoft.NET\Framework\{version}Temporary ASP.NET Files ASP.NET process account and impersonated identities: Full Control
□	Temporary directory (%temp%) ASP.NET process account: Full Control
□	.NET Framework directory %windir%\Microsoft.NET\Framework\{version} ASP.NET process account and impersonated identities: Read and Execute List Folder Contents
□	.NET Framework configuration directory %windir%\Microsoft.NET\Framework\{version}\CONFIG ASP.NET process account and impersonated Identities: Read and Execute List Folder Contents Read
□	Web site root C:\inetpub\wwwroot or the path that the default Web site points to ASP.NET process account: Full Control
□	System root directory %windir%\system32 ASP.NET process account: Read

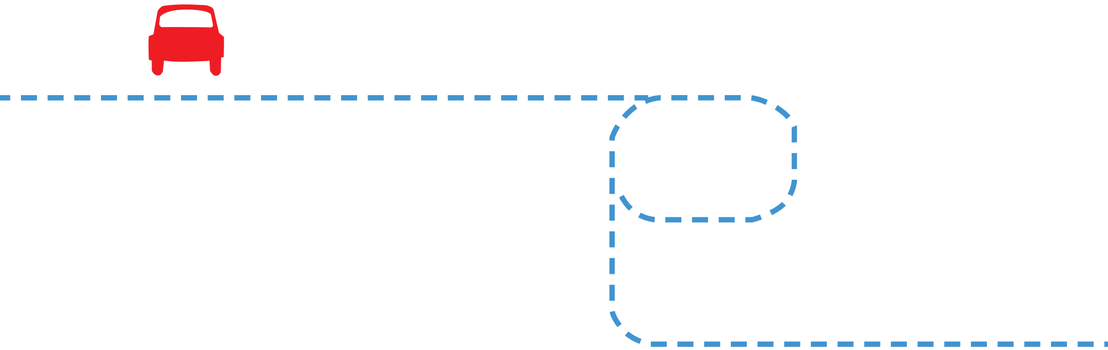
Cont...

Check	Description
□	Global assembly cache `%windir%\assembly` Process account and impersonated identities: Read

□	Content directory
	`C:\inetpub\wwwroot\YourWebApp`
	Process account:
	Read and Execute
	List Folder Contents
	Read
	Note: With .NET Framework version 1.0, all parent directories from the content directory to the file system root directory also require the above permissions. Parent directories include:
	`C:\`
	`C:\inetpub\`
	`C:\inetpub\wwwroot\`

Application Bin Directory

Check	Description
□	IIS Web permissions are configured. Bin directory does not have Read, Write or Directory browsing permissions. Execute permissions are set to None.
□	Authentication settings are removed (so that all access is denied).



Security Checklist:

Enterprise Services
(.NET Framework 1.1)

Security Checklist: Enterprise Services (.NET Framework 1.1)

How To Use This Module

This checklist is a companion to Chapter 11, 'Building Secure Serviced Components' and Chapter 17, 'Securing Your Application Server' from 'Improving Web Application Security: Threats and Countermeasures' at **http://msdn.com/secnet** – use it to help you secure Enterprise Services and the server it runs on, or as a quick evaluation snapshot of the corresponding chapters.

Developer Checks

Use the following checks if you build serviced components.

Authentication

Check	Description

□	Call-level authentication is used at minimum to prevent anonymous access.
	Serviced component assemblies include:
	`[assembly: ApplicationAccessControl`
	`(Authentication = AuthenticationOption.Call)]`

Authorisation

Check	Description
□	Role-based security is enabled. Serviced component assemblies include: `[assembly: ApplicationAccessControl(true)]`

□	Component-level access checks are enabled to support component-level, interface-level and method-level role checks. Serviced component assemblies include:
	`[assembly: ApplicationAccessControl(AccessChecksLevel=`
	`AccessChecksLevelOption.ApplicationComponent)]`

□	Component-level access checks are enforced for all serviced components. Classes are annotated with: `[ComponentAccessControl(true)]`
□	To support method-level security, the `[SecurityMethod]` attribute is used on classes or method implementations, or the `[SecurityRole]` attribute is used on method implementations.

Configuration Management

Check	Description
□	Server applications are configured to run with least-privileged accounts.
□	Server applications only run using the interactive user account during development.
□	Object constructor strings do not contain plaintext secrets.

Sensitive Data

Check	Description

□	In the absence of IPSec encryption, RPC encryption is used to secure sensitive data over the network in the absence of an IPSec infrastructure. Serviced component assemblies that use RPC encryption include:
	`[assembly: ApplicationAccessControl`
	`   (Authentication = AuthenticationOption.Privacy)]`

Auditing and Logging

Check	Description
□	User transactions are logged to an event log. The audit record includes original caller identity from **SecurityCallContext.OriginalCaller**.

12 hours' coding
– need a coffee

Deployment Considerations

Check	Description
□	Port ranges are defined if you use dynamic port range allocation OR static endpoint mapping is configured.
□	Secrets are not stored in object constructor strings. Secrets such as database connection strings are encrypted prior to storage.
□	The server application run-as account is configured as a least-privileged account.

Impersonation

Check	Description
□	The impersonation level is configured correctly. For ASP.NET clients, the impersonation level is configured in Machine.config on the **<processModel>** element. For Enterprise Services client applications, the level is configured in the COM+ catalog.

□	Serviced component assemblies define the required impersonation level by using the **ApplicationAccessControl** attribute as shown below:
	`[assembly: ApplicationAccessControl`
	`   (ImpersonationLevel=ImpersonationLevelOption.Identify)]`

Administrator Checklist

Check	Description
□	Latest COM+ updates and patches are installed.
□	Object constructor strings do not contain plaintext secrets.
□	COM+ administration components are restricted.
□	Impersonation level that is set for the application is correct.
□	Server applications are configured to run with a least-privileged account. Server applications do not run using the identity of the interactively logged on user.
□	DTC service is disabled if it is not required.

Security Checklist:
Remoting (.NET Framework 1.1)

Security Checklist: Remoting (.NET Framework 1.1)

How To Use This Module

This checklist is a companion to Chapter 13, 'Building Secure Remoted Components' from 'Improving Web Application Security: Threats and Countermeasures' at **http://msdn.com/secnet** – use it to help you build secure components that use the Microsoft .NET remoting technology and as a snapshot of the corresponding chapter.

Design Considerations

Check	Description
▢	Remote components are not exposed to the Internet.
▢	The ASP.NET host and **HttpChannel** are used to take advantage of Internet Information Services (IIS) and ASP.NET security features.
▢	**TcpChannel** (if used) is only used in trusted server scenarios.
▢	**TcpChannel** (if used) is used in conjunction with custom authentication and authorisation solutions.

Input Validation

Check	Description
▢	**MarshalByRefObj** objects from clients are not accepted without validating the source of the object.
▢	The risk of serialisation attacks are mitigated by setting the **typeFilterLevel** attribute programmatically or in the application's Web.config file.
▢	All field items that are retrieved from serialised data streams are validated as they are created on the server side.

Authentication

Check	Description
□	Anonymous authentication is disabled in IIS.
□	ASP.NET is configured for Windows authentication.
□	Client credentials are configured at the client through the proxy object.
□	Authentication connection sharing is used to improve performance.
□	Clients are forced to authenticate on each call (**unsafeAuthenticatedConnectionSharing** is set to "false").
□	**connectionGroupName** is specified to prevent unwanted reuse of authentication connections.
□	Plaintext credentials are not passed over the network.
□	IPrincipal objects passed from the client are not trusted.

Authorisation

Check	Description
□	IPSec is used for machine-level access control.
□	File authorisation is enabled for user access control.
□	Users are authorised with principal-based role checks.
□	Where appropriate, access to remote resources is restricted by setting **rejectRemoteRequest** attribute to "true".

Warning – unstable coding ahead

Configuration Management

Check	Description
□	Configuration files are locked down and secured for both the client and the server.
□	Generic error messages are sent to the client by setting the mode attribute of the **<customErrors>** element to "On".

Sensitive Data

Check	Description
□	Exchange of sensitive application data is secured by using SSL, IPSec or a custom encryption sink.

Exception Management

Check	Description
□	Structured exception handling is used.
□	Exception details are logged (not including private data, such as passwords).
□	Generic error pages with standard, user-friendly messages are returned to the client.

Auditing and Logging

Check	Description
□	If ASP.NET is used as the host, IIS auditing features are enabled.
□	If required, a custom channel sink is used to perform logging on the client and the server.

Security Checklist:

Web Services (.NET Framework 1.1)

Security Checklist: Web Services (.NET Framework 1.1)

How To Use This Module

This checklist is a companion to Chapter 12, 'Building Secure Web Services' from 'Improving Web Application Security: Threats and Countermeasures' at **http://msdn.com/secnet** – use it to help you build and secure your ASMX Web services and also as a snapshot of the corresponding chapter.

Design Considerations

Check	Description
▢	The authentication strategy has been identified.
▢	Privacy and integrity requirements of SOAP messages have been considered.
▢	Identities that are used for resource access have been identified.
▢	Implications of code access security trust levels have been considered.

Development Considerations

Input Validation

Check	Description
▢	Input to Web methods is constrained and validated for type, length, format and range.
▢	Input data sanitisation is only performed in addition to constraining input data.
▢	XML input data is validated based on an agreed schema.

Authentication

Check	Description
▢	Web services that support restricted operations or provide sensitive data support authentication.
▢	If plaintext credentials are passed in SOAP headers, SOAP messages are only passed over encrypted communication channels, for example, using SSL.
▢	Basic authentication is only used over an encrypted communication channel.
▢	Authentication mechanisms that use SOAP headers are based on Web Services Security (WSS) using the Web Services Enhancements (WSE).

Authorisation

Check	Description
▢	Web services that support restricted operations or provide sensitive data support authorisation.
▢	Where appropriate, access to Web service is restricted using URL authorisation or file authorisation if Windows authentication is used.
▢	Where appropriate, access to publicly accessible Web methods is restricted using declarative principle permission demands.

Conflict error

Sensitive Data

Check	Description
□	Sensitive data in Web service SOAP messages is encrypted using XML encryption OR messages are only passed over encrypted communication channels (for example, using SSL).

Parameter Manipulation

Check	Description
□	If parameter manipulation is a concern (particularly where messages are routed through multiple intermediary nodes across multiple network links), messages are digitally signed to ensure that they cannot be tampered with.

Exception Management

Check	Description
□	Structured exception handling is used when implementing Web services.
□	Exception details are logged (except for private data, such as passwords).
□	**SoapExceptions** are thrown and returned to the client using the standard **<Fault>** SOAP element.
□	If application-level exception handling is required a custom SOAP extension is used.

Auditing and Logging

Check	Description
□	The Web service logs transactions and key operations.

Proxy Considerations

Check	Description
□	The endpoint address in Web Services Description Language (WSDL) is checked for validity.
□	The URL **Behavior** property of the Web reference is set to dynamic for added flexibility.

Administration Considerations

Check	Description
□	Unnecessary Web service protocols, including HTTP GET and HTTP POST, are disabled.
□	The documentation protocol is disabled if you do not want to support the dynamic generation of WSDL.
□	The Web service runs using a least-privileged process account (configured through the **<processModel>** element in Machine.config). Custom accounts are encrypted by using **Aspnet_setref.exe**.
□	Tracing is disabled with: `<trace enabled="false" />`
□	Debug compilations are disabled with: `<compilation debug="false" explicit="true" defaultLanguage="vb">`

Security Checklist:

Network Security

Security Checklist: Network Security

How To Use This Module

This checklist is a companion to Chapter 15, ‘Securing Your Network’ from ‘Improving Web Application Security: Threats and Countermeasures’ on MSDN at **http://msdn.com/secnet** – use it to help secure your network or as a quick evaluation snapshot of the corresponding chapters. This checklist should evolve as you discover steps that help implement your secure network.

Router Considerations

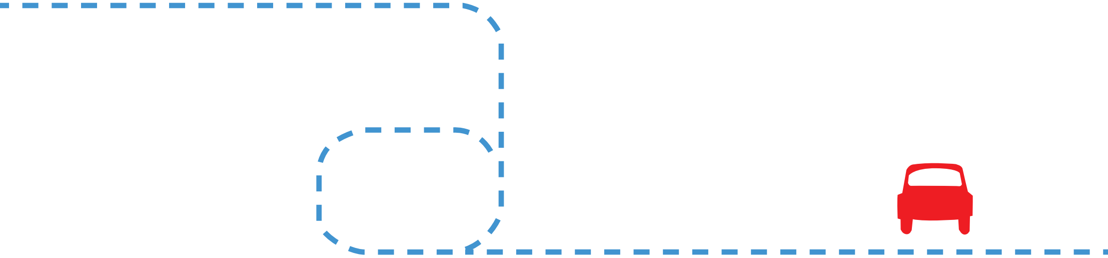
Check	Description
▢	Latest patches and updates are installed.
▢	You subscribed to router vendor's security notification service.
▢	Known vulnerable ports are blocked.
▢	Ingress and egress filtering is enabled. Incoming and outgoing packets are confirmed as coming from public or internal networks.
▢	ICMP traffic is screened from the internal network.
▢	Administration interfaces to the router are enumerated and secured.
▢	Web-facing administration is disabled.
▢	Directed broadcast traffic is not received or forwarded.
▢	Unused services are disabled (for example, TFTP).
▢	Strong passwords are used.
▢	Logging is enabled and audited for unusual traffic or patterns.
▢	Large ping packets are screened.
▢	Routing Information Protocol (RIP) packets, if used, are blocked at the outermost router.

Firewall Considerations

Check	Description
□	Latest patches and updates are installed.
□	Effective filters are in place to prevent malicious traffic from entering the perimeter.
□	Unused ports are blocked by default.
□	Unused protocols are blocked by default.
□	IPsec is configured for encrypted communication within the perimeter network.
□	Intrusion detection is enabled at the firewall.

Switch Considerations

Check	Description
□	Latest patches and updates are installed.
□	Administrative interfaces are enumerated and secured.
□	Unused administrative interfaces are disabled.
□	Unused services are disabled.
□	Available services are secured.



Security Checklist:

Web Server (IIS 5.1)

Security Checklist: Web Server (IIS 5.1)

How To Use This Module

This checklist is a companion to Chapter 16, 'Securing Your Web Server' from 'Improving Web Application Security: Threats and Countermeasures' on MSDN at **http://msdn.com/secnet** – use it to help implement a secure Web server running IIS 5.1 and ASP.NET version 1.1, or as a quick evaluation snapshot of the corresponding chapter.

Patches and Updates

Check	Description
□	MBSA is run on a regular interval to check for latest operating system and components updates. For more information, see **www.microsoft.com/technet/treeview/default.asp?url= /technet/security/tools/Tools/mbsahome.asp**
□	The latest updates and patches are applied for Windows, IIS server and the .NET Framework. (These are tested on development servers prior to deployment on the production servers.)
□	Subscribe to the Microsoft Security Notification Service at **www.microsoft.com/technet/ treeview/default.asp?url=/technet/security/bulletin/notify.asp**

IISLockdown

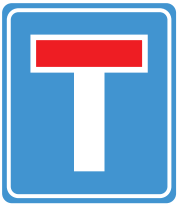
Check	Description
□	IISLockdown has been run on the server.
□	URLScan is installed and configured.

Services

Check	Description
□	Unnecessary Windows services are disabled.
□	Services are running with least-privileged accounts.
□	FTP, SMTP and NNTP services are disabled if they are not required.
□	Telnet service is disabled.
□	ASP.NET state service is disabled and is not used by your applications.

Protocols

Check	Description
□	WebDAV is disabled if not used by the application OR it is secured if it is required. For more information, see Microsoft Knowledge Base article 323470, 'How To: Create a Secure WebDAV Publishing Directory'.
□	TCP/IP stack is hardened.
□	NetBIOS and SMB are disabled (closes ports 137, 138, 139 and 445).



Accounts

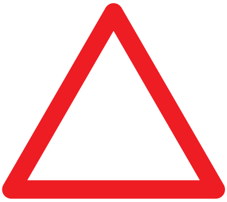
Check	Description
□	Unused accounts are removed from the server.
□	Windows Guest account is disabled.
□	Administrator account is renamed and has a strong password.
□	IUSR_MACHINE account is disabled if it is not used by the application.
□	If your applications require anonymous access, a custom least-privileged anonymous account is created.
□	The anonymous account does not have write access to Web content directories and cannot execute command-line tools.
□	ASP.NET process account is configured for least privilege. (This only applies if you are not using the default ASPNET account, which is a least-privileged account.)
□	Strong account and password policies are enforced for the server.
□	Remote logons are restricted. (The "Access this computer from the network" user-right is removed from the Everyone group.)
□	Accounts are not shared among administrators.
□	Null sessions (anonymous logons) are disabled.
□	Approval is required for account delegation.
□	Users and administrators do not share accounts.
□	No more than two accounts exist in the Administrators group.
□	Administrators are required to log on locally OR the remote administration solution is secure.

Files and Directories

Check	Description
□	Files and directories are contained on NTFS volumes.
□	Web site content is located on a non-system NTFS volume.
□	Log files are located on a non-system NTFS volume and not on the same volume where the Web site content resides.
□	The Everyone group is restricted (no access to \WINNT\system32 or Web directories).
□	Web site root directory has deny write ACE for anonymous Internet accounts.
□	Content directories have deny write ACE for anonymous Internet accounts.
□	Remote IIS administration application is removed (\WINNT\System32\Inetsrv\IISAdmin).
□	Resource kit tools, utilities and SDKs are removed.
□	Sample applications are removed (\WINNT\Help\IISHelp, \Inetpub\IISSamples).

Shares

Check	Description
□	All unnecessary shares are removed (including default administration shares).
□	Access to required shares is restricted (the Everyone group does not have access).
□	Administrative shares (C$ and Admin$) are removed if they are not required (Microsoft Management Server (SMS) and Microsoft Operations Manager (MOM) require these shares).



Missing character

Ports

Check	Description
□	Internet-facing interfaces are restricted to port 80 (and 443 if SSL is used).
□	Intranet traffic is encrypted (for example, with SSL) or restricted if you do not have a secure data centre infrastructure.

Registry

Check	Description
□	Remote registry access is restricted.
□	SAM is secured (HKLM\System\CurrentControlSet\Control\LSA\NoLMHash). This applies only to standalone servers.

Auditing and Logging

Check	Description
□	Failed logon attempts are audited.
□	IIS log files are relocated and secured.
□	Log files are configured with an appropriate size depending on the application security requirement.
□	Log files are regularly archived and analysed.
□	Access to the Metabase.bin file is audited.
□	IIS is configured for W3C Extended log file format auditing.

Sites and Virtual Directories

Check	Description
▢	Web sites are located on a non-system partition.
▢	"Parent paths" setting is disabled.
▢	Potentially dangerous virtual directories, including IISSamples, IISAdmin, IISHelp, and Scripts virtual directories are removed.
▢	MSADC virtual directory (RDS) is removed or secured.
▢	Include directories do not have Read Web permission.
▢	Virtual directories that allow anonymous access restrict Write and Execute Web permissions for the anonymous account.
▢	There is script source access only on folders that support content authoring.
▢	There is write access only on folders that support content authoring and these folder are configured for authentication (and SSL encryption, if required).
▢	FrontPage Server Extensions (FPSE) are removed if not used. If they are used, they are updated and access to FPSE is restricted.

Script Mappings

Check	Description
▢	Extensions not used by the application are mapped to 404.dll (.idq, .htw, .ida, .shtml, .shtm, .stm, idc, .htr, .printer).
▢	Unnecessary ASP.NET file type extensions are mapped to 'HttpForbiddenHandler' in Machine.config.

Conflict error

ISAPI Filters

Check	Description
□	Unnecessary or unused ISAPI filters are removed from the server.

IIS Metabase

Check	Description
□	Access to the metabase is restricted by using NTFS permissions (%systemroot%\system32\inetsrv\metabase.bin).
□	IIS banner information is restricted (IP address in content location disabled).

Server Certificates

Check	Description
□	Certificate date ranges are valid.
□	Certificates are used for their intended purpose (for example, the server certificate is not used for e-mail).
□	The certificate's public key is valid, all the way to a trusted root authority.
□	The certificate has not been revoked.

Machine.config

Check	Description
□	Protected resources are mapped to **HttpForbiddenHandler**.
□	Unused HttpModules are removed.
□	Tracing is disabled. `<trace enable="false"/>`
□	Debug compiles are turned off. `<compilation debug="false" explicit="true" defaultLanguage="vb">`

Code Access Security

Check	Description
□	Code access security is enabled on the server.
□	All permissions have been removed from the local intranet zone.
□	All permissions have been removed from the Internet zone.

Other Check Points

Check	Description
□	IISLockdown tool has been run on the server.
□	HTTP requests are filtered. URLScan is installed and configured.
□	Remote administration of the server is secured and configured for encryption, low session time-outs and account lockouts.

Do’s and Don’ts

- Do use a dedicated machine as a Web server.
- Do physically protect the Web server machine in a secure machine room.
- Do configure a separate anonymous user account for each application, if you host multiple Web applications.
- Do not install the IIS server on a domain controller.
- Do not connect an IIS Server to the Internet until it is fully hardened.
- Do not allow anyone to locally log on to the machine except for the administrator.

Late home again =
unhappy girlfriend

Security Checklist:

Database Server
(SQL Server 2000)

Security Checklist: Database Server (SQL Server 2000)

How To Use This Module

This checklist is a companion to Chapter 18, 'Securing Your Database Server' from 'Improving Web Application Security: Threats and Countermeasures' on MSDN at **http://msdn.com/secnet** – use it to help you secure a database server and also as a snapshot of the corresponding chapter.

Installation Considerations for Production Servers

Check	Description
☐	Upgrade tools, debug symbols, replication support, books online and development tools are not installed on the production server.
☐	Microsoft SQL Server is not installed on a domain controller.
☐	SQL Server Agent is not installed if it is not being used by any application.
☐	SQL Server is installed on a dedicated database server.
☐	SQL Server is installed on an NTFS partition.
☐	Windows Authentication mode is selected unless SQL Server Authentication is specifically required, in which case Mixed Mode is selected.
☐	A strong password is applied for the **sa** account or any other member of the **sysadmin** role. (Use strong passwords for all accounts.)
☐	The database server is physically secured.

Patches and Updates

Check	Description
☐	The latest service packs and patches have been applied for SQL Server. (See **http://support.microsoft.com/default.aspx?scid=kb;EN-US;290211**)
☐	Post service-pack patches have been applied for SQL Server.

Services

Check	Description
□	Unnecessary Microsoft Windows services are disabled on the database server.
□	All optional services, including Microsoft Search Service, MSSQLServerADHelper and SQLServerAgent, are disabled if not used by any applications.
□	The Microsoft Distributed Transaction Coordinator (MS DTC) is disabled if it is not being used by any applications.
□	A least-privileged local/domain account is used to run the various SQL Server services, for example, backup and replication.

Protocols

Check	Description
□	All protocols except TCP/IP are disabled within SQL Server. Check this using the Server Network Utility.
□	The TCP/IP stack is hardened on the database server.

Accounts

Check	Description
□	SQL Server is running using a least-privileged local account (or optionally, a least-privileged domain account if network services are required).
□	Unused accounts are removed from Windows and SQL Server.
□	The Windows guest account is disabled.
□	The administrator account is renamed and has a strong password.
□	Strong password policy is enforced.
□	Remote logons are restricted.
□	Null sessions (anonymous logons) are restricted.
□	Approval is required for account delegation.
□	Shared accounts are not used.
□	Membership of the local administrators group is restricted (ideally, no more than two administration accounts).

Files and Directories

Check	Description
□	Restrictive permissions are configured on SQL Server installation directories (per the guide).
□	The Everyone group does not have permission to access SQL Server installation directories.
□	Setup log files are secured.
□	Tools, utilities and SDKs are removed or secured.
□	Sensitive data files are encrypted using EFS. (This is an optional step. If implemented, use EFS only to encrypt MDF files, not LDF log files.)

Shares

Check	Description
□	All unnecessary shares are removed from the server.
□	Access to required shares is restricted (the Everyone group doesn't have access).
□	Administrative shares (C$ and Admin$) are removed if they are not required (Microsoft Management Server (MS) and Microsoft Operations Manager (MOM) require these shares).

Ports

Check	Description
□	Restrict access to all ports on the server except the ports configured for SQL Server and database instances (TCP 1433 and UDP 1434 by default).
□	Named instances are configured to listen on the same port.
□	Port 3389 is secured using IPSec if it is left open for remote Terminal Services administration.
□	The firewall is configured to support DTC traffic (if required by the application).
□	The **Hide server** option is selected in the Server Network Utility (optional).

Registry

Check	Description
□	SQL Server registry keys are secured with restricted permissions.
□	The SAM is secured (standalone servers only).

Auditing and Logging

Check	Description
□	All failed Windows login attempts are logged.
□	All failed actions are logged across the file system.
□	SQL Server login auditing is enabled.
□	Log files are relocated from the default location and secured with access control lists.
□	Log files are configured with an appropriate size depending on the application security requirement.
□	Where the database contents are highly sensitive or vital, Windows is set to Shut Down mode on overflow of the security logs.

SQL Server Security

Check	Description
□	SQL Server authentication is set to **Windows only** (if supported by the application).
□	The SQL Server audit level is set to **Failure** or **All**.
□	SQL Server runs using a least-privileged account.

12 hours' coding
– need a coffee

SQL Server Logins, Users and Roles

Check	Description
▢	A strong **sa** password is used (for all accounts).
▢	SQL Server guest user accounts are removed.
▢	BUILTIN\Administrators server login is removed.
▢	Permissions are not granted for the public role.
▢	Members of **sysadmin** fixed server role are limited (ideally, no more than two users).
▢	Restricted database permissions are granted. Use of built-in roles, such as db_datareader and db_datawriter, are avoided because they provide limited authorisation granularity.
▢	Default permissions that are applied to SQL Server objects are not altered.

SQL Server Database Objects

Check	Description
▢	Sample databases (including Pubs and Northwind) are removed.
▢	Stored procedures and extended stored procedures are secured.
▢	Access to cmdExec is restricted to members of the sysadmin role.

Additional Considerations

Check	Description
□	A certificate is installed on the database server to support SSL communication and the automatic encryption of SQL account credentials (optional).
□	NTLM version 2 is enabled by setting **LMCompatibilityLeve**l to 5.

Staying Secure

Check	Description
□	Regular backups are performed.
□	Group membership is audited.
□	Audit logs are regularly monitored.
□	Security assessments are regularly performed.
□	Subscribe to the Microsoft Security Notification Service at **www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/notify.asp**

Warning – unstable coding ahead

Security Checklist:
ASP.NET version 2.0

Security Checklist: ASP.NET Version 2.0

How To Use This Module

This checklist is a companion to 'Security Guidelines: ASP.NET 2.0' available at **http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGGuidelines0001.asp** – use 'Security Guidelines: ASP.NET 2.0' to browse the ASP.NET 2.0 guidelines and learn what to do, why and how. Use this checklist to help you secure your ASP.NET 2.0 application. You should expand and evolve this security checklist by adding additional practices that you discover during software development.

Design Considerations

Check	Description
□	Security decisions should not rely on client-side validations; they are made on the server side.
□	The Web site is partitioned into public access areas and restricted areas that require authentication access. Navigation between these areas should not flow sensitive credentials information.
□	The identities used to access remote resources from ASP.NET Web applications are clearly identified.
□	Mechanisms have been identified to secure credentials, authentication tickets and other sensitive information over network and in persistent stores.
□	A secure approach to exception management is identified. The application fails securely in the event of exceptions.
□	The site has granular authorisation checks for pages and directories.
□	Web controls, user controls and resource access code are all partitioned in their own assemblies for granular security.

Application Categories Considerations

Auditing and Logging

Check	Description
□	Health monitoring is used for logging and auditing events.
□	Application is instrumented for user management events such as authentication success and failures, password resets, password changes and account lockout.
□	Application is instrumented for unusual activity such as multiple login attempts and replayed authentication tickets.
□	Access to significant business logic is instrumented.
□	Access to audit and log files are restricted, with application accounts having write access, administrative accounts having full access and operators have read access.
□	Application and audit events are logged on separate protected server.
□	Events are logged with appropriate levels of information to reconstruct system activity.
□	High volume, per-request events are captured with performance counters.

Authentication – Forms

Check	Description
□	Membership providers are used instead of custom authentication.
□	SSL is used to protect user credentials and authentication cookies.
□	If using SSL is not possible, the **SlidingExpiration** attribute is set to **false** and limited authentication cookie time-outs are used.
□	User login information is validated using the Regex class and/or your custom validation code.
□	Hashed password format is specified in provider configuration.
□	Passwords are not stored directly in the user store; password digests with salt are stored instead.
□	Strong passwords policies are enforced.
□	Access to the credential store is limited to application account.
□	Authentication cookies are not persisted.
□	Authentication cookie is encrypted and integrity checked.
□	Authentication cookies are restricted to HTTPS connections only by using the **requireSSL** attribute.
□	Site is partitioned to restricted areas and public areas.
□	Absolute URLs are used for navigation where the site is partitioned with secure and non-secure folders.
□	**httpOnlyCookies** attribute is set to **true** on authentication cookie to prevent client side script from accessing the cookie.
□	Unique cookie names and paths are used.

Conflict error

Authentication – Windows

Check	Description
□	Windows authentication is used where possible.
□	Strong passwords policies are enforced.
□	Impersonation is used only when original caller's security context is required for downstream tier for auditing or authorisation.
□	**LogonUser** is not used.
□	Protocol transition is used when multiple identities need to access downstream resources.

Authorisation

Check	Description
□	URL authorisation is used for page and directory access control.
□	File authorisation is used with Windows authentication.
□	Appropriate ACLs are configured on Web site files.
□	Role manager, instead of custom code, is used for roles authorisation.
□	Role caching is used if role store lookup is too costly.
□	If role caching is used, authorisation cookie is restricted to HTTPS connections by using the **requireSSL** attribute.
□	If using SSL is not possible, the **cookieSlidingExpiration** attribute is set to **false** and limited authentication cookie time-outs are used.
□	The authorisation cookie is not persisted on a user’s machine by setting the **createPersistentCookie** attribute to false.
□	Authorisation cookie is protected for tampering and reading information.

Code Access Security

Check	Description
□	Code access security is used when applications need to be isolated from each other.
□	The chosen trust level does not exceed your application's requirement.
□	If your application needs additional permissions, a custom trust policy is used.
□	Applications are isolated using Medium trust in hosted environments.
□	Attribute **allowOverride** is set to **false** in the machine-level Web.config file to ensure developers cannot change the trust level of their application.

Data Access

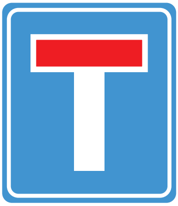
Check	Description
□	Connection strings are encrypted in configuration files using the Aspnet_regiis utility and Protected Configuration providers.
□	Connection string information is encrypted using strong encryption (for example, 3DES).
□	Connection to database is used with least-privileged service account.
□	Windows authentication is used when connecting to SQL Server.
□	Trusted service accounts are used to connect to SQL Server.
□	Mirrored local accounts are considered as an alternative if domain accounts cannot be used.
□	Strong passwords are used and enforced.
□	If SQL Server authentication is used, the credentials are secured over the network by using IPSec or SSL, or by installing a database server certificate.
□	Credentials in SQL connection strings are protected in configuration files.
□	RSA Protected Configuration provider is used to protect connection strings in a Web farm environment.
□	Untrusted input passed to data access methods is validated.
□	SQL queries use parameterised stored procedures and type-safe SQL parameters.
□	Dynamic queries that accept user input are used only if stored procedures cannot be used.

Exception Management

Check	Description
□	Structured exception handling is used.
□	Generic error pages with harmless messages are returned to the client.
□	Global error handlers are used to catch unhandled exceptions.
□	Set **mode** attribute in **customErrors** to **On** to prevent displaying detailed error messages to the caller.
□	Exception details are logged on the server.

Input/Data Validation

Check	Description
□	Free form input is sanitised to clean malicious data.
□	Application does not rely only on request validation.
□	All the input is validated for length, range, format and type. Input is checked for known valid and safe data and then for malicious, dangerous data.
□	Input from all the sources, including query strings, cookies and HTML controls, is validated using the Regex class and/or your custom validation code.
□	Application does not rely on only client-side validation.
□	Application avoids filename and path input from user where possible.
□	If input filenames are required, they are well formed and are verifiably valid within the application context.
□	Untrusted output is not directly echoed back to the user.
□	Output that contains untrusted data is encoded with HtmlEncode and UrlEncode.



Impersonation/Delegation

Check	Description
□	Trade-offs associated with use of impersonation are fully understood.
□	Use of **LogonUser** is avoided where possible.
□	Programmatic impersonation is avoided where possible.
□	Threading issues have been considered if impersonation is used.
□	Impersonation is reverted by using finally blocks.
□	Exceptions while impersonating are not allowed to propagate.

Parameter Manipulation

Check	Description
□	Security decisions are not made based on client parameters.
□	All the input parameters are validated for type, length, format and range.
□	Sensitive data is not stored in view state.
□	View state is encrypted if it does contain sensitive data.
□	**Page.ViewStateUserKey** is used to counter one-click attacks.
□	Query strings with server secrets are hashed.

Sensitive Data

Check	Description
□	Plaintext passwords are not used in configuration files (Web.config and Machine.config).
□	Sensitive data that is stored in .config files are encrypted using Protected Configuration providers.
□	Platform features are used and custom key management is avoided.
□	Sensitive data is not passed across pages; it is maintained using server-side state management.
□	Sensitive data passed over wire is secured using SSL or IPSec where appropriate.
□	Sensitive data is not cached.
□	Sensitive data is not stored in cookies, hidden form fields or query strings.
□	Output caching for pages that contain sensitive data is turned off.
□	Sensitive data is encrypted in the database.

Missing character

Session Management

Check	Description
□	Application does not rely on client-side state management options.
□	Windows authentication is used to connect to Microsoft SQL Server state database.
□	Session state connection strings are encrypted using protected configuration providers.
□	Out-of-process state service is protected.
□	Access to state data is restricted.
□	SQL Server session state is protected.
□	The session cookie is protected using SSL on all pages that require authenticated access.
□	The session state service is disabled if not used.
□	The session state service (if used) runs using a least-privileged account.
□	The communication channel to state store is encrypted (IPSec or SSL).
□	Session state port is changed from default of 42424.

Deployment Considerations

Check	Description
□	Least-privileged service account is used for running ASP.NET applications.
□	Configuration sections that contain sensitive data are encrypted using protected configuration providers.
□	Keys are stored in machine-level key store for application on dedicated server or multiple applications that run under the same identity.
□	Keys are stored in user-level key store for applications running in a shared hosting environment.
□	Protected file types are blocked using **HttpForbiddenHandler**.
□	The same machine keys are used consistently across all servers in a Web farm.
□	Configuration settings are locked by setting **allowOverride** to **false** where appropriate to enforce policy settings.
□	Set **mode** attribute in **customErrors** to **On** to prevent displaying detailed error messages to the caller.

Communication Security

Check	Description
□	Appropriate mechanism of secure communication (IPSec or SSL) is used, depending on application requirement.
□	For communication between Web browser and Web server, SSL is used when pages need to be encrypted and you need to guarantee that the server to which you send the data is the server that you expect.
□	For communication between servers, IPSec is used when secure server-to-server communication is required.
□	For communication between servers, SSL is used when an application does not trust other applications on a server.
□	Pages that use SSL are optimised.

Late home again = unhappy girlfriend

Security Checklist:

.NET Framework version 2.0

Security Checklist: .NET Framework 2.0

How To Use This Module

This checklist is a companion to 'Security Guidelines: .NET Framework 2.0' available at **http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGGuidelines0003.asp** – use 'Security Guidelines: .NET Framework 2.0' to learn about the .NET Framework 2.0 guidelines and to learn what you should do, why you should do it and how you can implement each guideline. Use this checklist as you develop your managed code. You should expand and evolve this security checklist by adding managed code practices that you discover during software development.

Assembly Design Considerations

Check	Description
□	Target trust environment is identified. Permissions available to partial trust code and APIs that require additional permissions are identified.
□	Design exposes a minimal number of public interfaces to limit the assembly's attack surface.

Class Design Considerations

Check	Description
□	To reduce visibility, classes and members use the most restrictive access modifier possible.
□	Base classes that are not intended to be derived from are sealed.
□	Strong naming or code access security is used to restrict code access.
□	Input is not trusted. Input is validated for type, range, format and length.
□	Fields are private. Properties are used to expose fields.
□	Properties are read-only unless write access is specifically required.
□	Where appropriate, private default constructors are used to prevent object instantiation.
□	Static constructors are private.

Strong Names

Check	Description
□	If required, strong names are used.
□	Strong names are not relied upon to create tamper-proof assemblies.
□	Delay signing is used to reduce the chance of private key compromise or to enable the use of a single public key across a team.
□	In full trust scenarios, **StrongNameIdentityPermission** is not relied upon to restrict code that can call the assembly.

APTCA

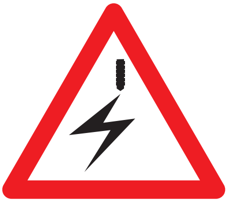
Check	Description
□	Except where necessary, APTCA usage is avoided.
□	Assemblies marked with APTCA are subjected to thorough security code review.
□	**SecurityTransparent** and **SecurityCritical** attributes are used appropriately.

Exception Management

Check	Description
□	Structured exception handling is used instead of returning error codes.
□	Sensitive data is not logged.
□	System or sensitive application information is not revealed. Only generic error messages are returned to the end user.
□	Code is not subject to exception filter issues where the filter higher in the call stack executes before code in a finally block.
□	Where appropriate, an exception management system is used.
□	Code fails early to avoid unnecessary processing.

File I/O

Check	Description
□	Code avoids untrusted input for filenames and file paths.
□	If filenames must be accepted through input, the names and locations are first validated.
□	Security decisions are not based on user-supplied filenames.
□	Where possible, absolute file paths are used.
□	Where appropriate, file I/O is constrained within the application's context.



Lightning coder

Registry

Check	Description
□	Sensitive data stored in HKEY_LOCAL_MACHINE is protected by ACLs.
□	Sensitive data in the registry is encrypted.

Communication Security

Check	Description
□	Transport-level encryption is used to protect secrets over the network. IPSec is used to protect the communication channel between two servers and SSL is used for more granular channel protection for an application.
□	Where appropriate, the **System.Net.Security.NegotiateStream** class is used for a TCP channel with .NET remoting.

Event Log

Check	Description
□	Sensitive data is not logged in the event log.
□	Event log data is not exposed to unauthorised users.

Data Access

Check	Description
□	Connection strings are not hard coded. Connection strings are stored in configuration files.
□	Connection strings are encrypted if they contain credentials.
□	To prevent SQL injection, input is validated and parameterised stored procedures are used.

Delegates

Check	Description
□	Delegates are not accepted from untrusted sources.
□	Where appropriate, permissions to the delegate are restricted.
□	Permissions are not asserted before delegate is called.

Serialisation

Check	Description
□	The **ISerializable** interface or the **NonSerialized** attribute are used to control serialisation of sensitive data.
□	Serialised data streams are validated when they are deserialised.

Threading

Check	Description
□	Multi-threaded code does not cache the results of security checks.
□	Impersonation tokens are not lost; they flow to the newly created thread.
□	Static class constructors are synchronised.
□	**Dispose** methods are synchronised.

12 hours' coding – need a coffee

Reflection

Check	Description
□	Full assembly names are used when **Activator.CreateInstance** loads add-ins.
□	Separate, low-trust application domains are used for assemblies created with user input.
□	Assemblies are not loaded dynamically based on user input for assembly or type names.
□	Untrusted code does not use **Reflection.Emit** to create dynamic assemblies.
□	Unless required, dynamic assemblies created by **Reflection.Emit** are not persisted.
□	**Assembly.ReflectionOnlyLoadFrom** is used only if you need to inspect code.

Obfuscation

Check	Description
□	Secrets are not stored in code.
□	Where appropriate, obfuscation is used to make intellectual property theft more difficult.

Cryptography

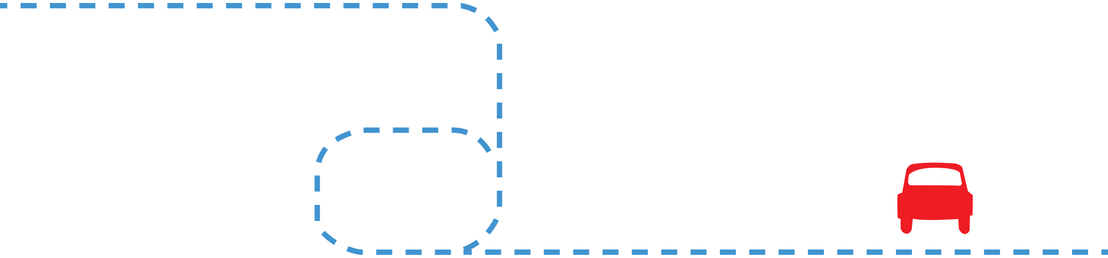
Check	Description
□	Platform-provided cryptographic services are used. Custom cryptography algorithms are not used.
□	Appropriate key sizes are used.
□	**GenerateKey** is used to generate random keys for a managed symmetric cryptographic class.
□	Where appropriate, DPAPI is used to protect secrets and to reduce or eliminate key management.
□	**Rfc2898DeriveBytes** is used to generate keys for password-based encryption.
□	Keys are not stored in code.
□	Access to persisted keys is restricted (for example with ACLs).
□	Keys are cycled periodically.
□	Exported private keys are protected.

Sensitive Data

Check	Description
□	Where appropriate, **SecureString** is used rather than **System.String**.
□	Secrets are held in memory for only a limited time.
□	Protected configuration is used to protect sensitive data and secrets in configuration files.

Unmanaged Code

Check	Description
□	Naming conventions are used (safe, native, unsafe) to identify unmanaged APIs.
□	Unmanaged API calls are isolated in a wrapper assembly.
□	String parameters that are passed to native code are constrained and validated to reduce the risk of buffer overrun, integer overflow and other vulnerabilities.
□	Array bounds are validated when an array is used to pass input to a native API.
□	File path lengths are checked when a filename and path are passed to an unmanaged API.
□	Unmanaged code is compiled with the /GS switch to enable stack probes.
□	Unmanaged code is inspected for potentially dangerous APIs.
□	Unmanaged types or handles are not exposed to partially trusted code.
□	The **SuppressUnmanagedCode** attribute is used only if assembly takes precautions to ensure that malicious code cannot coerce it into performing unwanted operations.
□	Pointers are held in private fields to prevent access violation or attempt to dereference them to gain access to sensitive information.



Security Checklist:

ADO.NET 2.0

Security Checklist: ADO.NET 2.0

How To Use This Module

This checklist is a companion to 'Security Guidelines: ADO.NET 2.0' available at **http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGGuidelines0002.asp** – use 'Security Guidelines: ADO.NET 2.0' to learn about the ADO.NET 2.0 guidelines and to learn what you should do, why you should do it and how you can implement each guideline. Use this checklist as you develop your data access code. You should expand and evolve this security checklist by adding data access practices that you discover during software development.

Input / Data Validation

Check	Description
□	Regular expressions are used to validate input against expected patterns.
□	In ASP.NET applications, ASP.NET validator controls are used to constrain and validate input.
□	The application does not rely only on ASP.NET request validation.
□	All untrusted input is validated inside data access methods.

SQL Injection

Check	Description
□	Input data is constrained and sanitised. Data is checked for type, length, format and range.
□	Type-safe SQL parameters are used for data access.
□	Where possible, dynamic queries that accept untrusted input are avoided.
□	With dynamic SQL, character escaping is used to handle special input characters.
□	The application login is restricted and has limited database permissions.

Configuration and Connection Strings

Check	Description
□	Where possible, Windows authentication is used to avoid placing credentials in connection strings.
□	Aspnet_regiis is used to encrypt credentials stored in connection strings in configuration files.
□	RSA encryption is used to protect credentials stored in connection strings on Web farm servers.
□	In the connection string, the **PersistSecurityInfo** attribute is not specified or is set to false or no.
□	Where possible, connection strings are not constructed with user input.
□	If user input must be used to build connection strings, the input is validated and **ConnectionStringBuilder** is used.
□	Where possible, Universal Data Link (UDL) files for OLE DB data sources are avoided.

Authentication

Check	Description
□	Where possible, Windows authentication is used to connect to the database.
□	If SQL authentication is used, then strong passwords are used and enforced.
□	If SQL authentication is used, then IPSec or SSL is used to protect credentials on the network.
□	If SQL authentication is used, then Aspnet_regiis is used to encrypt connection strings in configuration files.
□	RSA encryption is used to protect credentials stored in connection strings on Web farm servers.
□	The account used to connect to the database has restricted database permissions.

Conflict error

Authorisation

Check	Description
□	Role checks or declarative or imperative principal permission checks are used to restrict calling users.
□	Where appropriate, the data access library code is designed to restrict the access of calling code.
□	The data access library code uses strong names to constrain partial trust callers.
□	Application-specific data access code is placed in the application's bin directory.
□	The application's database login is restricted in the database and can execute selected stored procedures only. The application login has no direct table access.

Exception Management

Check	Description
□	Database connections are closed with **using** statements or in **finally** blocks.
□	ADO.NET exceptions are not propagated to users. Only generic exception information is displayed.
□	In ASP.NET applications, a generic error page is used to avoid accidentally returning detailed error information to the client.
□	ADO.NET exception details are logged on the server.

Sensitive Data

Check	Description
□	If sensitive data must be stored, then a strong symmetric encryption algorithm such as AES is used to encrypt it. DPAPI is used to protect symmetric encryption keys.
□	Sensitive data is protected with IPSec or SSL on the network.
□	Passwords are stored as irreversible hash values with added salt. Passwords are not stored in clear text or in encrypted format.

Code Access Security

Check	Description
□	A custom ASP.NET policy is used to access non-SQL Server databases from partial trust ASP.NET applications.
□	Extended **OleDbPermission** syntax is used to restrict database access on hosted servers.
□	**StrongNameIdentityPermission** is not the only means used to restrict full trust callers.

Deployment Considerations

Check	Description
□	Only required ports are opened and firewall restrictions are applied for the application.
□	If credentials are stored in configuration files, they are encrypted. RSA encryption is used on Web farm servers.
□	Database auditing is enabled and failed login attempts are logged.

Missing probe

Security Checklist:

Windows Communication Foundation

Security Checklist: Windows Communication Foundation

How To Use This Module

This module describes some common scenarios for protecting WCF services. Use it to help you configure and protect your WCF services.

Scenario: Message-Level Protection and Windows Authentication in a Self-Hosted Intranet Service using the TCP Transport

Description

You are building a WCF service for internal use inside an organisation. The WCF service is hosted in its own application, not IIS. Client applications connect to the service by using a TCP connection. Users are all members of a single Windows domain controlled by the organisation.

Service Configuration

Check	Description
□	Binding configuration added to the service configuration file based on**NetTcpBinding.**
□	The **mode** attribute of the **security** element of the binding configuration is set to **Message.**
□	The **algorithmSuite** attribute of the **message** element is set to an encryption algorithm.
□	The **clientCredentialType** attribute of the **message** element is set to **Windows.**
□	The binding configuration is specified in the endpoint configuration for the service.
□	A **net.tcp** address is specified in the **Address** attribute of the service endpoint configuration.

Client Configuration

Check	Description
□	Binding configuration added to the client configuration file based on **NetTcpBinding.**
□	The **mode** attribute of the **security** element of the binding configuration is set to **Message.**
□	The **algorithmSuite** attribute of the **message** element is set to the same encryption algorithm specified for the server.
□	The **clientCredentialType** attribute of the **message** element is set to **Windows.**
□	The binding configuration is specified in the endpoint configuration for the client.
□	The **net.tcp** address of the service is set in the **Address** attribute of the endpoint configuration for the client.

Scenario: Message-Level Protection and Windows Authentication in a Self-Hosted Intranet Service using the HTTP Transport

Description

You are building a WCF service for internal use inside an organisation. The WCF service is hosted in its own application, not IIS. Client applications connect to the service by using an HTTP connection. Users are all members of a single Windows domain controlled by the organisation.

Service Configuration

Check	Description
▢	Binding configuration added to the service configuration file based on **WSHttpBinding.**
▢	The **mode** attribute of the **security** element of the binding configuration is set to **Message.**
▢	The **algorithmSuite** attribute of the **message** element is set to an encryption algorithm.
▢	The **clientCredentialType** attribute of the **message** element is set to **Windows.**
▢	The binding configuration is specified in the endpoint configuration for the service.
▢	An **http** address is specified in the **Address** attribute of the service endpoint configuration.

Client Configuration

Check	Description
▢	A binding configuration is added to the client configuration file based on**WSHttpBinding.**
▢	The **mode** attribute of the **security** element of the binding configuration is set to **Message.**
▢	The **algorithmSuite** attribute of the **message** element is set to the same encryption algorithm specified for the server.
▢	The **clientCredentialType** attribute of the **message** element is set to **Windows.**
▢	The binding configuration is specified in the endpoint configuration for the client.
▢	The **http** address of the service is set in the **Address** attribute of the endpoint configuration for the client.

Note: The default security mode for the WSHttpBinding binding is Message. The default algorithm suite is Basic256, and the default message client credential type is Windows. If you don't need to change the default algorithm suite, use a default WSHttpBinding binding rather than creating a customised binding configuration.

Scenario: Transport-Level Protection and Basic Authentication in a Self-Hosted Intranet Service using the HTTPS Transport

Description

You are building a WCF service for internal use inside an organisation. The WCF service is hosted in its own application, not IIS. Client applications connect to the service by using an HTTPS connection over SSL. Users are all members of a single Windows domain controlled by the organisation.

Service Configuration

Check	Description
▢	An SSL certificate is installed in the **Personal** certificate store for the **Local Computer** account.
▢	The HTTP protocol for the service is configured to accept SSL requests; use the **httpcfg set ssl** command (Windows Server 2003), or the **netsh http add sslcert** command (Windows Vista®).
▢	A binding configuration is added to the service configuration file based on **BasicHttpBinding.**
▢	The **mode** attribute of the **security** element of the binding configuration is set to **Transport.**
▢	The **clientCredentialType** attribute of the **transport** element is set to **Basic.**
▢	The binding configuration is specified in the endpoint configuration for the service.
▢	An **https** address is specified in the **Address** attribute of the service endpoint configuration.

Client Configuration

Check	Description
▢	A binding configuration is added to the client configuration file based on **BasicHttpBinding.**
▢	The **mode** attribute of the **security** element of the binding configuration is set to **Transport.**
▢	The **clientCredentialType** attribute of the **transport** element is set to **Basic.**
▢	The binding configuration is specified in the endpoint configuration for the client.
▢	The **https** address of the service is specified in the **Address** attribute of the endpoint configuration for the client.
▢	Code is added to populate the **ClientCredentials** property of the client proxy to provide a valid username and password when connecting to the service.

Scenario: Transport-Level Protection and UserName Authentication in an Internet Service

Description

You are building a WCF service for public use. The WCF service is hosted by using IIS using SSL. Client applications connect to the service by using an HTTPS connection. Users are not members of a single Windows domain, but identify themselves at the message level by providing a username and password. You are using the SQL Role Provider to manage users and roles.

Note: The SQL Role provider operates at the message level and uses Forms Authentication to validate users . You can configure SSL to provide transport-level protection, but specify that messages include credentials that identify the user to the service.

Service Configuration

Check	Description
□	An SSL certificate is specified in the certificate store for the **Local Computer** account.
□	Using IIS Manager, an **https** Web site binding is added to the Web site hosting the Web service, specifying the SSL certificate.
□	Using IIS Manager, the SSL settings for the Web service are configured to require access by using SSL.
□	Using IIS Manager, **Anonymous Authentication**, and **Forms Authentication** are enabled. All other authentication mechanisms for the Web service are disabled.
□	A binding configuration is added to the service configuration file based on **WSHttpBinding.**
□	The **mode** attribute of the security element of the binding configuration is set to **TransportWithMessageCredential.**
□	The **clientCredentialType** attribute of the **message** element is set to **UserName.**
□	The **clientCredentialType** attribute of the **transport** element is set to **None.**
□	The binding configuration is specified in the endpoint configuration for the service.
□	The ASP.NET Web Site Administration tool is used to define users and roles for the service. The authentication type is set to From the Internet.
□	A service behaviour is added to the service configuration file, and a **serviceAuthorization** element with the **principalPermissionMode** element set to **UseAspNetRoles** and the **roleProviderName** element set to **AspNetSqlRoleProvider** is also added.
□	A **serviceCredentials** element is added to the service behaviour.
□	A **userNameAuthentication** element is added to the **serviceCredentials** element with the **userNamePasswordValidationMode** attribute set to **MembershipProvider**, and the **membershipProviderName** attribute set to **AspNetSqlMembershipProvider.**
□	The behaviour is specified in the service configuration for the service.

Client Configuration

Check	Description
□	A binding configuration is added to the client configuration file based on **WSHttpBinding.**
□	The **mode** attribute of the **security** element of the binding configuration is set to **TransportWithMessageCredential.**
□	The **clientCredentialType** attribute of the **message** element is set to **UserName.**
□	The **clientCredentialType** attribute of the **transport** element is set to **None.**
□	The binding configuration is specified in the endpoint configuration for the client.
□	The **https** address of the service is specified in the **Address** attribute of the endpoint configuration for the client.
□	Code is added to populate the **ClientCredentials** property of the client proxy to provide a valid username and password when connecting to the service.

Scenario: Message-Level Protection and Certificate Authentication in an Internet Service

Description

You are building a WCF service for public use. The WCF service is hosted by using IIS. Client applications connect to the service by using an HTTP connection. Users are not members of a single Windows domain, but identify themselves by providing a certificate. You are using the SQL Role Provider to manage users and roles.

Note: The SQL Role provider operates at the message level and uses Forms Authentication to validate users.

Service Configuration

Check	Description
▢	Using IIS Manager, **Anonymous Authentication**, and **Forms Authentication** is enabled for the Web service. All other authentication mechanisms are disabled for the Web service.
▢	A binding configuration is added to the service configuration file based on **WSHttpBinding.**
▢	The **mode** attribute of the **security** element of the binding configuration is set to **Message.**
▢	The **clientCredentialType** attribute of the **message** element is set to **Certificate.**
▢	Either: • The negotiateServiceCredential attribute of the message element is set to false, or • The negotiateServiceCredential attribute of the message element is set to true and the client certificates are imported into the appropriate certificate store on the service computer.
▢	The binding configuration is specified in the endpoint configuration for the service.
▢	Using the ASP.NET Web Site Administration tool users and roles are defined for the service. The authentication type is set to **From the Internet.**
▢	A service behaviour is added to the service configuration file, and a **serviceAuthorization** element with the **principalPermissionMode** element set to **UseAspNetRoles** is added and the **roleProviderName** element is set to **AspNetSqlRoleProvider.**
▢	A **serviceCredentials** element is added to the service behaviour.
▢	A **userNameAuthentication** element is added to the **serviceCredentials** element with the **userNamePasswordValidationMode** attribute set to **MembershipProvider**, and the **membershipProviderName** attribute set to **AspNetSqlMembershipProvider.**
▢	A **clientCertificate** element is added to the **serviceCredentials** element with the **certificateValidationMode** attribute of the authentication element set to the appropriate trust level for client certificates.
▢	The behaviour is specified in the service configuration for the service.

Client Configuration

Check	Description
□	The certificate for identifying the user is installed in the certificate store for the user account running the client.
□	A binding configuration is added to the client configuration file based on **WSHttpBinding.**
□	The **mode** attribute of the **security** element of the binding configuration is set to **Message.**
□	The **clientCredentialType** attribute of the **message** element is set to **Certificate**, and the **negotiateServiceCredential** attribute is set to the value specified in the service configuration file.
□	The binding configuration is specified in the endpoint configuration for the client.
□	The **https** address of the service is specified in the **Address** attribute of the endpoint configuration for the client.
□	Code is added to populate the **ClientCredentials** property of the client proxy to reference the client certificate when connecting to the service.

Scenario: Certificate Authentication of an Internet Service

Description

You are building a WCF service for public use. The WCF service is hosted by using IIS. Client applications connect to the service by using an HTTP connection. Client applications authenticate the service by using the service certificate.

Note: You can combine this scenario with the previous one to implement mutual authentication by using certificates.

Service Configuration

Check	Description
□	The certificate for identifying the service is installed in the certificate store for the **Local Computer** account.
□	Read access is granted to the file holding the certificate to the **ASPNET** account (Windows Server 2003, XP), or **NETWORKSERVICE** account (Windows Vista).
□	A binding configuration is added to the service configuration file based on **WSHttpBinding.**
□	The **mode** attribute of the **security** element of the binding configuration is set to **Message.**
□	A service behaviour is set to the service configuration file.
□	A **serviceCredentials** element is set to the service behaviour.
□	A **serviceCertificate** element is set to the **serviceCredentials** element, and the name and location of the service certificate is specified.
□	The behaviour is specified in the service configuration for the service.

Conflict error

Client Configuration

Check	Description
□	The service certificate is imported into the certificate store for the user account running the client.
□	A binding configuration is added to the client configuration file based on **WSHttpBinding.**
□	The **mode** attribute of the **security** element of the binding configuration is set to **Message.**
□	An endpoint behaviour is added to the client configuration file.
□	A **clientCredentials** element is added to the service behaviour.
□	A **serviceCertificate** element is added to the **clientCredentials** element, and the name and location of the service certificate is specified.
□	The behaviour is specified in the endpoint configuration for the client.
□	The **http** address of the service is specified in the **Address** attribute of the endpoint configuration for the client.

Scenario: Claims-Based Security in an Internet Service

Description

You are building a WCF service for public use. The WCF service is hosted by using IIS. Client applications connect to the service by using an HTTP connection. Users identify themselves by providing a security credential that is authenticated by a trusted third-party service.

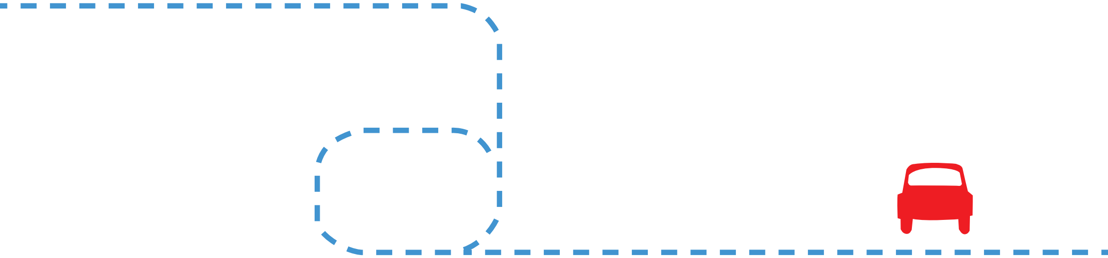
Service Configuration

Check	Description
□	A binding configuration is added to the service configuration file based on **WSFederationHttpBinding.**
□	The **issuedTokenType** attribute of the **security** element of the binding configuration is set to the type of token the service expects to receive from the identity provider.
□	A **claimTypeRequirements** element is added to the **security** element that specifies the claim types accepted by the service.
□	A certificate for identifying the service is installed in the certificate store for the **Local Computer** account.
□	A service behaviour is added to the service configuration file.
□	A **serviceCredentials** element is added to the behaviour, and the details of the service certificate are specified.
□	The behaviour is specified in the service configuration for the service.
□	The binding configuration is specified in the endpoint configuration for the service.
□	An **http** address is specified in the **Address** attribute of the service endpoint configuration.

Missing probe

Client Configuration

Check	Description
□	A binding configuration is added to the service configuration file based on **WSFederationHttpBinding.**
□	(Optional) A **claimTypeRequirements** element is added to the security element that specifies the claim types sent by the client.
□	The **issuedTokenType** attribute of the **security** element of the binding configuration is set to the type of token that the client sends to the service.
□	The service certificate is imported into the certificate store for the user account running the client.
□	The binding configuration is specified in the endpoint configuration for the service.
□	An **http** address is specified in the **Address** attribute of the service endpoint configuration.
□	An **identity** element is added to the binding configuration, and the details of the service certificate are specified.



Security Checklist:

Native Code Security

Security Checklist: Native Code

How To Use This Module

This is a native code security checklist for C++ developers. Use this checklist as you write and review native C++ code.

Compiler

Check	Description
□	/GS buffer security check. The /GS compiler switch is on to detect buffer overruns. For more information see **http://msdn2.microsoft.com/en-us/library/8dbf701c(VS.71).aspx**
□	/analyze (Enterprise Code Analysis): The /analyze compiler option is used so that potential security issues such as buffer overrun, un-initialised memory, null pointer dereferencing, and memory leaks are reported. For more information see **http://msdn2.microsoft.com/en-us/library/ms173498(VS.80).aspx**
□	/RTCs – Stack frame run-time error checking: /RTCs (Runtime error check with sub option s) compiler option is used to enable stack frame run-time error checking. For more information see **http://msdn2.microsoft.com/en-us/library/8wtf2dfz(VS.80).aspx**
□	/RTCc – Detects assignments that resulted in data loss: /RTCc (Runtime error check with sub option c) compiler option is used to report a data loss due to a value being assigned to a smaller data type. For more information see **http://msdn2.microsoft.com/en-us/library/8wtf2dfz(VS.80).aspx**
□	/RTCu – Report variable use without initialisation: /RTCu (Runtime error check with sub option u) compiler option is used to detect when a variable is used without having been initialised. For more information see **http://msdn2.microsoft.com/en-us/library/8wtf2dfz(VS.80).aspx**

Libraries

Check	Description
□	Security-Enhanced CRT: Security-enhanced C runtime library functions are used instead of the standard C and C++ functions. For more information see **http://msdn2.microsoft.com/en-us/library/9a89h429(VS.71).aspx**
□	Checked Iterators: While using standard C++ library, checked iterators are used to be notified of elements accessed outside the bounds of a container. For more information see **http://msdn2.microsoft.com/en-us/library/aa985965(VS.80).aspx**
□	Debug Iterator Support: Debug iterator support is used to detect incorrect iterator use. For more information see **http://msdn2.microsoft.com/en-us/library/aa985982(VS.80).aspx**

Linker

Check	Description
□	/SAFESEH (Image has Safe Exception Handlers): The /SAFESEH linker option is on to ensure that only legitimate exception handlers are executed. For more information see **http://msdn2.microsoft.com/en-us/library/9a89h429(VS.80).aspx**

Miscellaneous

Check	Description
□	Windows Application Verifier: AppVerifier tool is used to identify potential application compatibility, stability and security issues. For more information see **http://msdn2.microsoft.com/en-us/library/Aa480483.aspx**
□	UAC: When using Windows Vista, User Account Control is used so that the user accounts have limited privileges. For more information see **http://msdn2.microsoft.com/en-us/library/aa905330.aspx**

12 hours' coding
– need a coffee

Question Lists for Conducting Security Code Reviews

Question Lists for Conducting Security Code Reviews

How To Use The Question Lists

Use the question lists to help to conduct an effective code review for security. Each question category presents a set of questions that you can use to determine if your application is susceptible to the listed vulnerabilities. When you use the question lists, keep the following in mind:

- Use the question lists to help you perform the security review activity described in module 6, 'Security Code Review'.
- Use the question lists as a starting point while reviewing your code for security issues. Evolve and develop the question lists for the specifics of your own applications and environment.
- Prioritise questions for review. You may not need to answer all of the questions because some may not be relevant to your application.

.NET Framework 2.0 Question List

Use this question list while reviewing managed code written for .NET Framework 2.0.

Vulnerability	Questions
Code Access Security	
Improper use of link demands or asserts	Does the code use link demands or assert calls?
Code allows untrusted callers	Does your code use **AllowPartiallyTrustedCallersAttribute**? Does the code use potentially dangerous permissions? Does the code give dependencies too much trust?
Exception Management	
Failing to use structured exception handling	Does the code use proper and consistent error checking? Does the application fail securely in the event of exceptions?
Revealing too much information to the client	Do error messages give away too much information?
Impersonation	
Revealing service account credentials to the client	Does the application use hard coded impersonation credentials?
Code runs with higher privileges than expected	Does the code clean up properly when it uses impersonation?

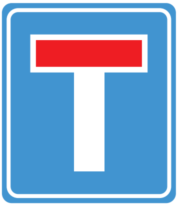
Vulnerability	Questions
Sensitive Data	
Storing secrets in code	Does the code store secrets?
Storing secrets in clear text	Is sensitive data stored in predictable locations?
Passing sensitive data in clear text over networks	Does the code store secrets?
Cryptography	
Using custom cryptography	Did the team develop cryptographic algorithms?
Using the wrong algorithm or too small a key size	Does the code use the right algorithm with an adequate key size? Does the code generate random numbers for cryptographic purposes?
Failing to secure encryption keys	How does the code manage and store encryption keys?
Using the same key for a prolonged period of time	How does the code manage and store encryption keys?
Unsafe Code	
Buffer overrun in unmanaged code or code marked /unsafe	Is the code susceptible to buffer overruns?
Integer overflow in unmanaged code or code marked /unsafe	Is the code susceptible to integer overflows?
Format string problem in unmanaged code or code marked /unsafe	Is the code susceptible to format string problems?
Array out-of-bounds in unmanaged code or code marked /unsafe	Is the code susceptible to array out-of-bound errors?
Data truncation in unmanaged code or code marked /unsafe	Is the code susceptible to data truncation errors?
Potentially Dangerous Unmanaged APIs	
A potentially dangerous unmanaged API is called improperly	Does the code call potentially dangerous unmanaged APIs?

Vulnerability	Questions
Auditing and Logging	
Sensitive data revealed in logs	Does the code log sensitive data?
Multi-Threading	
Race conditions	Is the code subject to race conditions?
Synchronisation issues	Does the code contain static class constructors? Does the code synchronise **Dispose** methods?

ASP.NET 2.0 Question List

Use this question list while reviewing managed ASP.NET 2.0 Web application code.

Vulnerability	Questions
SQL Injection	
Non-validated input used to generate SQL queries	Is your application susceptible to SQL injection? Does your code use parameterised stored procedures? Does your code use parameters in SQL statements? Does your code attempt to filter input?
Cross-Site Scripting	
Unvalidated and untrusted input in the Hypertext Markup Language (HTML) output stream	Does the code echo user input or URL parameters back to a Web page? Does the code persist user input or URL parameters to a data store that could later be displayed on a Web page?



Server connection error

Vulnerability	Questions
Input / Data Validation	
Reliance on client-side validation	Does the code rely on client-side validation?
Use of input filenames, URLs or user names for security decisions	Is the code susceptible to canonicalisation attacks?
Application-only filters for malicious input	Does the code validate data from all sources? Does the code centralise its approach? Does the code validate URLs? Does the code use MapPath?
Authentication	
Weak passwords	Does the code enforce strong user management policies?
Clear text credentials in configuration files	Does the code enforce strong user management policies? Does the code partition the Web site into restricted and public access areas?
Passing clear text credentials over the network	Does the code use **protection="All"**? Does the code restrict authentication cookies to HTTPS connections? Does the code use SHA1 for HMAC generation and AES for encryption?
Long sessions	Does the code reduce ticket lifetime?
Mixing personalisation with authentication	Does the code keep personalisation cookies separate from authentication cookies? Does the code use distinct cookie names and paths?
Forms Authentication	
Failure to protect the forms authentication cookie	Does the code persist forms authentication cookies? Does the code reduce ticket lifetime? Does the code use **protection="All"**? Does the code restrict authentication cookies to HTTPS connections? Does the code use SHA1 for HMAC generation and AES for encryption? Does the code keep personalisation cookies separate from authentication cookies?
Forms authentication cookies are shared by multiple applications	Does the code use distinct cookie names and paths?
Passwords are stored in a database in clear-text	How does the code store passwords in databases?

Vulnerability	Questions
Authorisation	
Reliance on a single gatekeeper	How does the code protect access to restricted pages? How does the code protect access to page classes? Does the code use **Server.Transfer**?
Code Access Security	
Improper use of link demands or asserts	Does the code use link demands or assert calls?
Code allows untrusted callers	Does the code use **AllowPartiallyTrustedCallers Attribute**? Does the code use dangerous permissions? Does the code give dependencies too much trust?
Exception Management	
Failing to use structured exception handling	Does the code use proper and consistent error checking? Does the code ensure that the application fails securely if exceptions occur?
Revealing too much information to the client	Do error messages give away too much information?
Impersonation	
Revealing service account credentials to the client	Does the code use hard-coded impersonation credentials?
Code runs with higher privileges than expected	Does the code clean up properly when it uses impersonation?

Vulnerability	Questions
Sensitive Data	
Storing secrets in code	Does the code store secrets?
Storing secrets in clear text	Is sensitive data stored in predictable locations?
Passing sensitive data in clear text over networks	Does the code store secrets?
Data Access	
Failing to protect database connection strings	Does the code use SQL authentication?
Using overly privileged accounts to access SQL Server	How does the code store database connection strings?
Cryptography	
Using custom cryptography	Did the team develop cryptographic algorithms?
Using the wrong algorithm or too small a key size	Does the code use the right algorithm with an adequate key size? Does the code generate random numbers for cryptographic purposes?
Failing to secure encryption keys	How does the code manage and store encryption keys?
Using the same key for a prolonged period of time	How does the code manage and store encryption keys?

Vulnerability	Questions
Unsafe Code	
Buffer overrun in unmanaged code or code marked /unsafe	Is the code susceptible to buffer overruns?
Integer overflow in unmanaged code or code marked /unsafe	Is the code susceptible to integer overflows?
Format string problem in unmanaged code or code marked /unsafe	Is the code susceptible to format string problems?
Array out-of-bounds in unmanaged code or code marked /unsafe	Is the code susceptible to array out-of-bound errors?
Data truncation in unmanaged code or code marked /unsafe	Is the code susceptible to data truncation errors?
Potentially Dangerous Unmanaged APIs	
A potentially dangerous unmanaged API is called improperly	Does the code call potentially dangerous unmanaged APIs?
Auditing and Logging	
Sensitive data revealed in logs	Does the code log sensitive data?

Late home again =
unhappy girlfriend

Part III

What's New for Security in the Microsoft .NET Framework 2.0

This part summarises the new and enhanced security features in the Microsoft® .NET Framework version 2.0 and Microsoft Visual Studio® .NET 2005.

What's New for Security in the Microsoft .NET Framework 2.0

Category	Description and More Information
Access Control	
Programming ACLs	You can now use the **System.Security. AccessControl** namespace to program Access Control Lists (ACLs) and Access Control Entries (ACEs) directly from managed code. **For more information, see http://msdn2.microsoft.com/en-us/library/system.security.accesscontrol.aspx**
Active Directory	
Active Directory Programming	The new **System.DirectoryServices. ActiveDirectory** namespace exposes an object model that you can use to perform tasks with Microsoft Active Directory. **For more information, see http://msdn2.microsoft.com/en-us/library/ms180923.aspx**
ASP.NET	
Configuration File Encryption	You can now encrypt sections of configuration files by using the new protected configuration feature and the Aspnet_regiis.exe utility. Providers for Data Protection Application Programming Interface (DPAPI) and RSA encryption are available. **For more information, see 'How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI,' at http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000005.asp, and 'How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA,' at http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000006.asp**
Membership	The new membership feature provides secure credential storage for Web application users. It also provides a membership Application Programming Interface (API) that simplifies the task of validating user credentials when used with forms authentication. Membership providers abstract the underlying store used to maintain user credentials. ASP.NET version 2.0 includes the following providers: • ActiveDirectoryMembershipProvider. This uses either an Active Directory or Active Directory Application Mode (ADAM) user store. • SqlMembershipProvider. This uses a Microsoft SQL Server user store. **For more information, see 'How To: Use Membership in ASP.NET 2.0' at http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000022.asp**

Category	Description and More Information
Login Controls	New login controls are provided to simplify the creation of forms authentication login and registration pages. These include Login, LoginView, LoginStatus, LoginName, PasswordRecovery, CreateUserWizard and ChangePassword controls. **For more information, see 'ASP.NET Login Controls Overview' at http://msdn2.microsoft.com/en-us/library/ms178329.aspx**
Role Management	The new role management feature provides protected role storage and an API for managing and checking role membership. The role manager supports a provider model. ASP.NET 2.0 includes the following providers: • **SqlRoleProvider** used with SQL Server role stores • **WindowsTokenRoleProvider** used with Microsoft Windows® authentication, which uses Windows groups as roles • **AuthorisationStoreRoleProvider** used with Microsoft Windows Server™ 2003 Authorisation Manager for managing roles in Active Directory or ADAM **For more information, see 'How To: Use Role Manager in ASP.NET 2.0,' at http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000013.asp**
Health Monitoring	The new health monitoring feature supports many standard events that you can use to monitor the health of your application. Examples of security related events that are automatically generated include logon failures and successes when using the ASP.NET membership system, attempts to tamper with or reuse forms authentication tickets, and infrastructure events such as disk access failures. You can also create custom events to instrument your application for other security and non-security related notable events. **For more information, see 'How To: Use Health Monitoring in ASP.NET 2.0,' at http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000011.asp**

Category	Description and More Information
Access Control	
URL Authorisation Enhancements	In ASP.NET version 1.1, Uniform Resource Locator (URL) authorisation only applies to file types that are mapped by Internet Information Services (IIS) to the ASP.NET ISAPI extension (Aspnet_isapi.dll). In ASP.NET 2.0 on Windows Server 2003, URL authorisation protects all files in a directory, even files that are not mapped to ASP.NET, such as .html, .gif and .jpg files.
machineKey Enhancements	The **machineKey** configuration file element now supports a decryption attribute that specifies the symmetric encryption algorithm used to encrypt and decrypt forms authentication tickets. ASP.NET 2.0 provides support for Advanced Encryption Standard (AES) symmetric encryption, which is used by default, in addition to Data Encryption Standard (DES) and 3DES. **For more information, see 'How To: Configure MachineKey in ASP.NET 2.0,' at http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000007.asp**
Sending E-mail	The **System.Net.Mail.SmtpPermission** is now available to applications configured to run at high or medium trust. This allows partial trust applications to send e-mail. **For more information, see http://msdn2.microsoft.com/en-us/library/system.net.mail.smtppermission.aspx**
Code Access Security	
Global Assembly Cache Installation Means Full Trust	Assemblies in the global assembly cache are now always granted full trust, regardless of the local machine security policy.
Full Trust Assemblies Now Satisfy Any Code Access Security Demands	Any fully trusted assembly will now satisfy any demand, including a link demand for an identity permission such as a **StrongNameIdentityPermission** that the assembly does not satisfy.
DataProtection-Permission	This new **System.Security.Permissions.DataProtectionPermission** type is used to control access to DPAPI functionality exposed through DPAPI managed wrapper classes. **For more information, see http://msdn2.microsoft.com/en-us/library/system.security.permissions.dataprotectionpermission.aspx**
Security Transparency	You can now mark assemblies with the **SecurityTransparent** attribute to let the common language runtime (CLR) know that your code will not perform security-sensitive code access security operations, such as asserting permissions or using stack walk modifiers to escalate privileges. If your code or any code that you call attempts such operations, a security exception is generated. This is particularly useful if your code loads third-party plug-ins.

Category	Description and More Information
ClickOnce Deployment	The new **ClickOnce deployment** technology supports sandboxed execution for Windows forms applications. The sandbox in which the application runs is restricted and only has a limited set of code access security permissions. Extensible Markup Language (XML) manifest files specify an application's permission requirements. **For more information, see 'ClickOnce Deployment' at http://msdn2.microsoft.com/en-us/library/t71a733d.aspx**
Simple Sandboxing	In the .NET Framework 1.x, to set up a sandboxed application domain — for example, to host untrusted code — you must create an application domain policy level, create a series of code groups and define the permission sets to be granted to each one. In the .NET Framework 2.0, you can use a new overload of the static AppDomain.CreateDomain method to help simplify this process. **For more information, see 'Find Out What's New with Code Access Security in the .NET Framework 2.0' at http://msdn.microsoft.com/msdnmag/issues/05/11/CodeAccessSecurity/default.aspx**
HostProtection-Attribute	You can add a **HostProtectionAttribute** to an application to inform the runtime about any functionality within that may potentially damage the underlying host, the application itself, or threads either within or outside the application. **For more information, see http://msdn2.microsoft.com/en-us/library/system.security.permissions.hostprotectionattribute.aspx**
PermCalc	**PermCalc.exe** is a permissions calculator to help you declaratively specify your assembly's permission requirements – based on the assemblies that you call. **For more information, see http://msdn2.microsoft.com/en-us/library/ms165077.aspx**

Category	Description and More Information
Communication Security	
SSPI	The .NET Framework version 2.0 provides new managed wrappers for the **Security Service Provider Interface (SSPI)** that enable you to implement both client and server-side secure channels by using Kerberos or Secure Sockets Layer (SSL).
Tcp Channel Security	The **TcpChannel** class now uses the SSPI to support both encryption and authentication over remoting channels. **For more information, see 'Security in Remoting' at http://msdn2.microsoft.com/en-us/library/9hwst9th.aspx**
Ipc Channel	The new **IpcChannel** class is ideal for communication between components on the same computer. The underlying implementation uses named pipes that can be secured with ACLs.
C++	
Safe CRT Libraries	The safe CRT libraries are updated versions of the standard C and C++ libraries, including the C Runtime (CRT) Library, Standard C++ Library (SCL), Active Template Library (ATL) and Microsoft Foundation Classes (MFC). The updates are designed to protect applications compiled with Visual C++®. They add appropriate buffer checks to functions known to be vulnerable to attack, parameters are validated, and other functions such as strcpy, which are known to be vulnerable to attack, are deprecated. You should use the secure version of a function if it exists. If a new secure function exists, the older, less secure version is marked as deprecated, and the new version has the _s (secure) suffix. For example, use strcpy_s instead of strcpy. Note that the compiler generates a warning if you use a deprecated function. **For a list of secure CRT functions, see 'Security-Enhanced Versions of CRT Functions' at http://msdn2.microsoft.com/en-us/library/wd3wzwts.aspx**
/Gs Switch	When code compiled with /GS detects a stack-based buffer overrun, the library code terminates it immediately with a minimum of extra code running in the process, thereby reducing the attack surface. The /Gs switch functionality has been further extended in Visual Studio .NET 2005. For more information see 'Write Faster Code with the Modern Language Features of Visual C++ 2005' at **http://msdn.microsoft.com/msdnmag/issues/04/05/VisualC2005/**
Prefix and Prefast	Prefix is a defect-detection tool that performs static analysis on code to find errors like memory leaks and other problems. Prefast is a lighter-weight programme analysis tool for detecting defects through the use of static analysis.

Category	Description and More Information
Cryptography	
XML Encryption	You can use the **System.Security.Cryptography.Xml.EncryptedXml** class to protect sensitive XML that must be stored on disk. XML encryption is a World Wide Web Consortium (W3C) compliant implementation, so you can exchange encrypted data with other implementations. You can encrypt portions of an XML document with the same or different keys. A number of encryption algorithms are supported, including DES, AES 128, AES 192, AES 256, RSA and **X509CertificateEx.** **For more information, see 'How To: Encrypt XML Elements with X.509 Certificates' at http://msdn2.microsoft.com/en-us/library/ms229744.aspx**
XML Digital Signatures	You can use the classes in the **System.Security.Cryptography.Xml** namespace to sign an XML document or part of an XML document with a digital signature. **For more information, see 'How To: Sign XML Documents with Digital Signatures' at http://msdn2.microsoft.com/en-us/library/ms229745.aspx**
PKCS Support	The **System.Security.Cryptography.Pkcs** contains the managed code implementation of the Cryptographic Message Syntax (CMS) and Public-Key Cryptography Standards #7 (PKCS #7) standards. CMS is a superset of PKCS #7. Classes within this namespace support extended certificate manipulation, including the retrieval of certificate properties, certificate validation and chain building. The **System.Security.Cryptography.Pkcs** namespace exposes a greater range of the Public Key Infrastructure (PKI) APIs that the Windows platform provides.
Rfc2898DeriveBytes	In .NET version 1.1, you could use **PasswordDeriveBytes** to generate keys from a password. .NET version 2.0 still supports this for backward compatibility, but you should now use **Rfc2898DeriveBytes**. The main advantage is that it supports the RSA Password-Based Key Derivation Function version 2 (PBKDF2), which is an improved version of the PBKDF1 standard implementation used by **PasswordDeriveBytes.**

Category	Description and More Information
Data Access	
Partial Trust Support	In ADO.NET version 1.1, you could only use the .NET Framework Data Provider for SQL Server from partial trust applications, because this provider was the only one that did not demand full trust. In ADO.NET version 2.0, the .NET Framework Data Provider for Oracle, the .NET Framework Data provider for OLE DB and the .NET Framework Data Provider for Open Database Connectivity (ODBC) no longer demand full trust. This enables you to access SQL Server and other databases from partial trust applications. Note, however, that medium-trust ASP.NET policy only grants the **SqlClientPermission** required by the .NET Framework Data Provider for SQL Server. To use the other providers from a partial trust Web application, you need to customise policy and grant the appropriate permission, for example, **OleDbPermission**, **OraclePermission** or **OdbcPermission**. **For more information about using OLE DB from partial trust ASP.NET applications, see 'How To: Use Medium Trust in ASP.NET 2.0' at http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000020.asp**
Improved Connection String Encryption	You can now use protected configuration to encrypt sections of your Machine.config and Web.config files by using either DPAPI or RSA encryption. This is particularly useful for encrypting database connection strings. **For more information, see 'How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI' at http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000005.asp, and 'How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA' at http://msdn.microsoft.com/library/en-us/dnpag2/html/PAGHT000006.asp**
New Connection String Settings Class	The **System.Configuration** namespace provides classes for working with information stored in configuration files. You can use the new **ConnectionStringSettings** class to retrieve connection strings from configuration files. The ConnectionString property contains the connection string value, and the Name property contains the name of the connection string specified in the ConnectionStrings section.
Connection String Builder Classes	The new **DbConnectionStringBuilder** class provides the base class from which strongly typed connection string builders derive. They enable you to programmatically create syntactically correct connection strings and to parse and rebuild existing connection strings. The following built-in data providers supply strongly typed classes that inherit from **System.Data.Common.DbConnectionStringBuilder:** **System.Data.SqlClient.SqlConnectionStringBuilder,** **System.Data.OracleClient.OracleConnectionStringBuilder,** **System.Data.Odbc.OdbcConnectionStringBuilder,** **System.Data.OleDb.OleDbConnectionStringBuilder.**

Warning – unstable coding ahead

Category	Description and More Information
New Abstract Base Class for Database Exceptions	The new **System.Data.Common.DbException** is the base class for all exceptions thrown on behalf of a data source. This abstract class is used as the base class for provider-specific exception class implementations. These include: **System.Data.SqlClient.SqlException, System.Data.OleDb.OleDbException, System.Data.OracleClient.OracleException and System.Data.Odbc.OdbcException**
Changing Passwords in SQL Server 2005	You can use the new **SqlConnection.ChangePassword** method to change the SQL Server password for the user indicated in the connection string to the supplied new password. This enables .NET Framework applications to change the password of a user account without requiring administrator intervention on Microsoft Windows Server$_{TM}$ 2003 or later.
Diagnostics	
SecurityException Enhancements	The **SecurityException** class has been enhanced to provide more information in the case of a failed permission request. The new exception object has also been tightly integrated into Visual Studio .NET 2005. When a **SecurityException** is raised during debugging, Visual Studio .NET observes the object's properties and displays context-sensitive help and rich feedback to help you diagnose the issue.
Reflection	
ReflectionOnlyLoad-From	The new **Assembly.ReflectionOnlyLoadFrom** method enables you to load code purely to examine its members. The loaded code is not allowed to run.

Category	Description and More Information
Sensitive Data	
DPAPI Managed Wrapper	In .NET Framework 1.1, you had to use P/Invoke to access **DPAPI** functions. In .NET Framework 2.0, you no longer need to use P/Invoke. You can use the new **ProtectedData** class instead. ProtectedData contains two static methods: Protect and Unprotect. Managed code requires the new **DataProtectionPermission** to be able to use DPAPI. To use DPAPI to encrypt data in memory, you can use the new ProtectedMemory class. **For more information, see 'How To: Use Data Protection' at http://msdn2.microsoft.com/en-us/library/ms229741.aspx**
SecureString	This new type uses DPAPI to ensure that secrets stored in string form are not exposed to memory or disk-sniffing attacks. **For more information, see 'SecureString Application Sample' at http://msdn2.microsoft.com/en-us/library/07b9wyhy.aspx**
Visual Studio .NET Development System	
Permissions Calculator	The permissions calculator checks the security requirements of an application by statically checking for called APIs. A permission set is returned for each library API. The tool outputs an estimate of the minimum set of permissions required to run the application.
Sandboxed Debugging for Partial Trust Applications	You can now debug your applications in a sandboxed execution environment with restricted code access security permissions.
IntelliSense® in Zone	You can use this feature together with code access security policies to help make the right choice about the APIs that you use in your application. When running in Visual Basic®, the IntelliSense in zone feature provides visual feedback to let you know about any APIs that are not available within the current execution zone and hence those that would cause the application to violate security policy.
FxCop Enhancements	FxCop is now integrated in Visual Studio .NET. The tool now scans managed code for 200 total defects including SQL injection, permissions and pointers. **For more information, see http://www.gotdotnet.com/team/fxcop**

Conflict error

Part IV

Patterns & Practices

Security Resources

Patterns & Practices Security Resources

Microsoft *patterns & practices* are Microsoft's recommendations for how to design, develop, deploy and operate architecturally sound applications for the Microsoft application platform. Microsoft *patterns & practices* contain deep technical guidance and tested source code based on real-world experience. The technical guidance is created, reviewed and approved by Microsoft architects, product teams, consultants, product support engineers and by Microsoft partners and customers. *Patterns & practices* guidance includes guides available on Microsoft MSDN®, as well as guidance available in the form of books; reference implementations that are sample applications designed to illustrate specific patterns and guidance; and Application Blocks, which are reusable components that provide solutions to common development challenges.

For more information about *patterns & practices*, see http://msdn.microsoft.com/practices

Application Blocks

Application Blocks are a specific form of *patterns & practices* guidance. They are reusable source-code components designed to demonstrate solutions to common development challenges. You can use them without modification and integrate them directly into your applications, or you can extend and customise them to suit your specific requirements. Full source code is provided together with design documentation that outlines the Application Block's purpose, usage scenarios and architecture. Quick-start tutorials and Software Development Kit (SDK)-style reference documentation is also provided.

In most cases, a separate architecture guide is also provided as a companion that presents the architectural or patterns overview. The Application Block implementation exhibits the best practices that are defined by the companion guidance. Application Blocks are built to address specific recurring problem domains common to most enterprise applications, such as data access, logging, exception management, creation of composite user interfaces, cryptography and authorisation.

For more information about the range of available application blocks, see http://msdn.microsoft.com/practices/guidetype/appblocks

Enterprise Library

Enterprise Library is a collection of reusable and extensible Application Blocks for enterprise Microsoft .NET development. The blocks in Enterprise Library assist in the following scenarios: caching, configuration, cryptography, data access, exception handling, logging and security. A *patterns & practices* Enterprise Library is available for the Microsoft .NET Framework version 1.1, version 2.0 and version 3.0.

For more information about the Enterprise Library for .NET 2.0 and 3.0, see http://www.microsoft.com/downloads/details.aspx?familyid=4c557c63-708f-4280-8f0c-637481c31718&displaylang=en

Application Blocks for Security

The following *patterns & practices* security-related Application Blocks are available:

Application Block	Description and More Information
Cryptography Application Block	The **Microsoft Enterprise Library Cryptography Application Block** simplifies how you incorporate cryptography functionality into your applications. You can use the Application Block in your applications for a variety of tasks, including encrypting information, creating a hash from data and comparing hash values to verify that data has not been altered. **For more information, see http://msdn.microsoft.com/practices/compcat/default.aspx?pull=/library/en-us/dnpag2/html/crypto1.asp**
Security Application Block	The **Microsoft Enterprise Library Security Application Block** helps you implement common security-related functionality into your applications. You can use this Application Block in your applications in a variety of situations, such as authenticating and authorising users against a database, retrieving role and profile information and caching user profile information. **For more information, see http://msdn.microsoft.com/library/en-us/dnpag2/html/security1.asp** **Note:** For .NET Framework 2.0, much of the functionality that this Application Block provides is now provided by the platform through the new **System.Web.Security.Membership** class and System.Web.ProFilenamespace. The new Security Application Block for .NET Framework 2.0 still includes factories, interfaces and providers for authorisation and security caching.

Security Guides

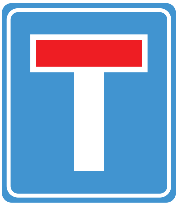
The following *patterns & practices* security-related guides are available:

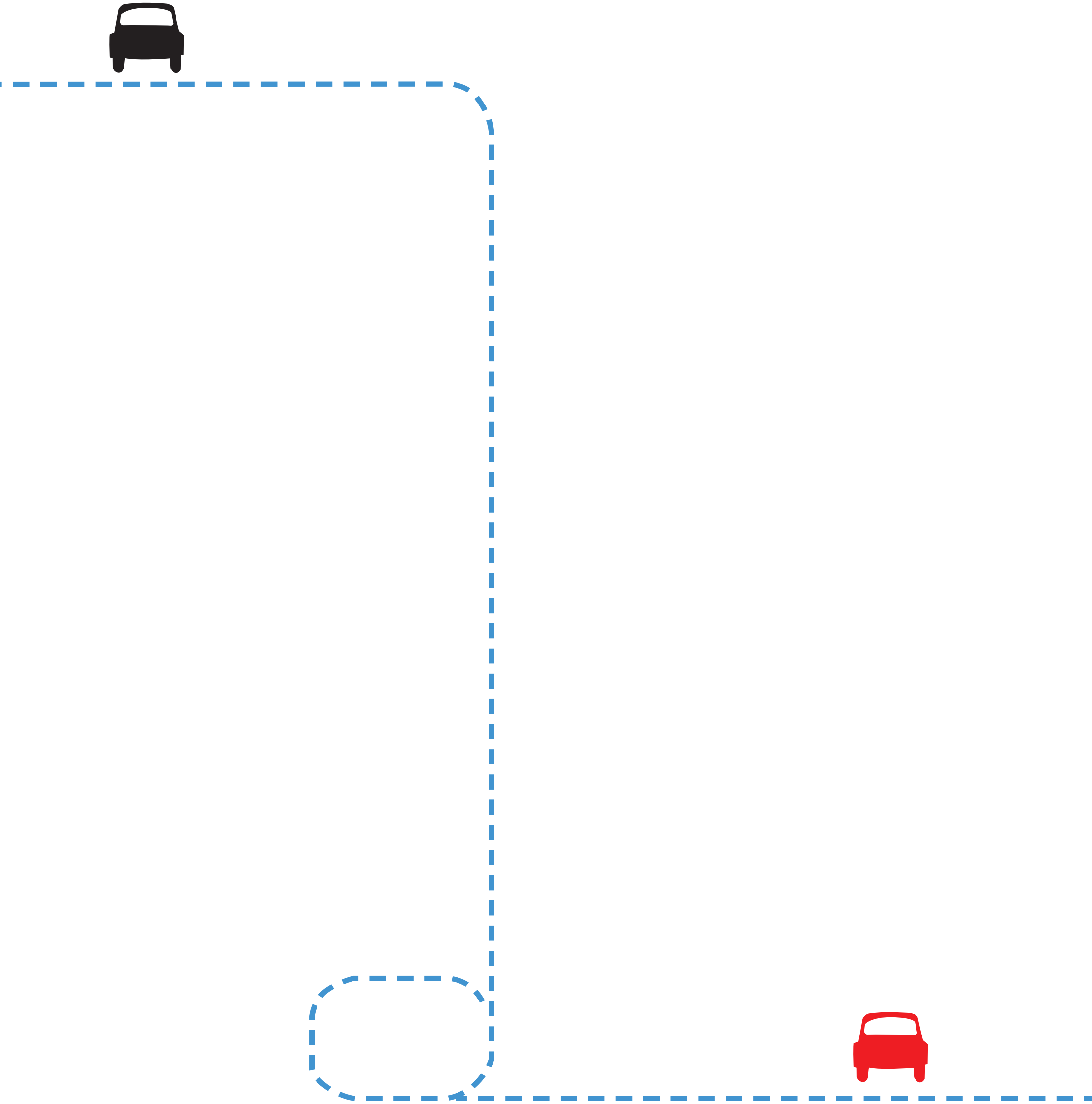
Guide	Description and More Information
Building Secure ASP.NET Applications: Authentication, Authorisation and Secure Communication	This guide presents a practical, scenario-driven approach to designing and building secure ASP.NET applications for Microsoft Windows 2000 and the .NET Framework version 1.0. It focuses on the key elements of authentication, authorisation and secure communication within and across the tiers of distributed .NET Web applications. **For more information, see http://msdn.microsoft.com/library/en-us/dnnetsec/html/secnetlpMSDN.asp**
Improving Web Application Security: Threats and Countermeasures	This guide provides a solid foundation for designing, building and configuring secure ASP.NET Web applications. The guide adopts a lifecycle-based approach to security. Whether you have existing applications or are building new ones, you can apply the guidance to help you make sure that your Web applications are hack-resilient. This guide applies to .NET Framework 1.1 and Windows 2000. **For more information, see http://msdn.com/secnet**
Designing Application-Managed Authorisation	This guide provides guidelines for designing and coding application-managed authorisation for single- or multi-tier applications that are based on .NET. It focuses on common authorisation tasks and scenarios, and it provides information that helps you choose the best approaches and techniques. **For more information, see http://msdn.microsoft.com/practices/Topics/security/default.aspx?pull=/library/en-us/dnbda/html/damaz.asp**
Authentication in ASP.NET: .NET Security Guidance	This document offers guidance to the application architect who is responsible for designing a security model for an ASP.NET Web application. The guide explains the relationship between Internet Information Services (IIS) and ASP.NET from a security standpoint and describes the set of available authentication methods. It also contains procedures that can help you choose the most appropriate authentication method based on your particular application scenario. **For more information, see http://msdn.microsoft.com/practices/Topics/security/default.aspx?pull=/library/en-us/dnbda/html/authaspdotnet.asp**
Web Service Security	Scenarios, Patterns and Implementation Patterns for Web Services Enhancements 3.0. **For more information, see http://msdn.microsoft.com/library/en-us/dnpag2/html/wssp.asp**

Reference Implementations for Security

The following *patterns & practices* security-related reference implementations are available:

Reference Implementation	Description and More Information
WS-I Basic Security Profile 1.0 Reference Implementation: Preview Release for the .NET Framework version 1.1	The **WS-I Basic Security Profile Reference Implementation** is built by using Web Services Enhancements (WSE) 2.0 and the .NET Framework 1.1 to illustrate how to build resilient, real-world, secure, interoperable Web services. **For more information, see http://msdn.microsoft.com/practices/compcat/default.aspx?pull=/library/en-us/dnpag2/html/MSWSIBSP.asp**





The drive for safer coding

Microsoft® | Application Security

The Developer Highway Code

To build software that meets your security objectives, you must integrate security activities into your software development lifecycle. This handbook captures and summarises the key security engineering activities that should be an integral part of your software development processes. These security engineering activities have been developed by Microsoft *patterns & practices* to build on, refine and extend core lifecycle activities with a set of security-specific activities. These include identifying security objectives, applying design guidelines for security, threat modelling, security architecture and design, security code and security deployment reviews.

What this handbook includes:

- Activity summaries that show you the steps necessary to perform each activity
- Security checklists that you can use as job aids while developing your software
- Security questions to ask while performing security code reviews
- Managed code and native code security checklists
- Windows Communication Foundation security checklists
- Other *patterns & practices* security resources
- Pointers to additional reading and reference materials

Who should read this handbook:

- Application architects
- Developers
- Testers and quality assurance specialists

This handbook was compiled by the following people:

- **Paul Maher.** Paul is a Technical Evangelist at Microsoft in the UK. He is responsible for driving Microsoft's next generation technologies and leading the UK Application Security initiative. You can contact Paul at **pmaher@microsoft.com** and view his blog at **http://blogs.msdn.com/paul_maher**

- **Alex Mackman**. Alex is a Principal Technologist at CM Group Ltd. (**www.cm-group.co.uk**) where he creates developer-focused courseware and technical content. Alex has 15 years' experience in the software development industry and has designed, created and delivered training and consultancy on many aspects of Windows software development. He has spent the majority of the last four years working with the Microsoft *patterns & practices* team developing .NET application security guidance. You can contact Alex at **alexm@cm-consulting.com**

Microsoft Application Security Web site: www.microsoft.com/uk/msdn/security

Microsoft

K1406