Microsoft

Security Engineering Explained

patterns & practices

Microsoft®

Security Engineering Explained

patterns & practices

J.D. Meier

Alex Mackman

Blaine Wastell

Prashant Bansode

Jason Taylor

Rudolph Araujo

Contents

Introduction

Summary

This chapter summarizes the patterns & practices approach to security engineering. To design, build, and deploy secure applications, you must integrate security into your application development life cycle by including specific security-related activities in your current software engineering processes. Security-related activities include identifying security objectives; applying security design guidelines, patterns, and principles; conducting security architecture and design reviews; creating threat models; performing security code reviews; security testing; and conducting security deployment reviews. You can adopt these activities incrementally as you see fit. These security activities are integrated in MSF Agile, available with Visual Studio Team System. This combination provides tools, guidance, and workflow to help make security a seamless part of your development experience.

Overview

This guide describes an approach for integrating security into your software development life cycle. It describes the set of security activities that you should use to refine and extend your existing life cycle activities. This guide presents an overview of the approach and explains the main security engineering activates and how to adopt them. For further detailed guidance, on each of the activities see the Security Engineering Index at *http://msdn.com/securityengineering*.

This guide consists of the following chapters:

- **Chapter 1, "patterns & practices Security Engineering Approach,"** summarizes the patterns & practices approach to security engineering and shows how the key security engineering activities fit into your software development life cycle.
- **Chapter 2, "Security Objectives,"** summarizes the patterns & practices approach to security objectives by explaining what they are and why you should use them. Knowledge of your security objectives is essential to the success of all other security-related activities.
- **Chapter 3, "Security Design Guidelines,"** summarizes the patterns & practices approach to security design guidelines. Adopting security design guidelines can help reduce your attack surface by addressing common vulnerabilities, allowing you to focus on the unique aspects of your design.

- **Chapter 4, "Threat Modeling,"** summarizes the patterns & practices approach to threat modeling. Threat modeling is an engineering technique that you can use to help identify threats, attacks, vulnerabilities, and countermeasures that may be relevant to your application.
- **Chapter 5, "Security Architecture and Design Review,"** summarizes the patterns & practices approach to security architecture and design review by explaining what it is and why you should use it. It also describes the key concepts behind the approach.
- **Chapter 6, "Security Code Review,"** summarizes the patterns & practices approach to security code review. Security code review is an effective mechanism for uncovering security issues 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.
- **Chapter 7, "Security Deployment Review,"** summarizes the patterns & practices approach to security deployment review. A security deployment review is an activity that can be used to ensure that configuration and deployment problems are discovered before the application is in production.

Why We Wrote This Guide

We wrote this guide to accomplish the following:

- To provide guidance on how to build software that meets your security objectives.
- To help integrate security engineering throughout your application life cycle.
- To explain key security-related activities such as threat modeling and to show you how to implement these approaches.

Features of the Guidance

To help maximize the value of this guidance, it provides the following features:

- **Life cycle approach.** The guide provides end-to-end guidance on building software that meets your security objectives, throughout your application life cycle, to reduce risk and increase your return on software development costs.
- **Security frame.** The guidance uses a security frame which is 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. Patterns & practices security guidance includes context-specific security frames for each major application type.

- **Principles and practices.** These serve as the foundation for the guidance and provide a stable basis for recommendations. They also reflect successful approaches used in the field.
- **Processes and activities.** The guidance provides steps for key activities including threat modeling, security architecture and design reviews, security code reviews and security deployment reviews. For simplification and tangible results, the life cycle is decomposed into activities with inputs, outputs, and steps. You can use the steps as a baseline or to help you evolve your own activities.
- **How Tos.** The guidance includes a set of step-by-step procedures to help you implement key solutions from the guidance.
- **Modular.** Each module within the guidance is designed to be read independently. You do not need to read the guidance from beginning to end to get the benefits, although you are encouraged to read *Security Engineering Explained* to understand the big picture.
- **Job aids.** The guide provides a number of review activities, including a security architecture and design review, to help you evaluate the security implications of your architecture and design choices early in the life cycle. A security code review helps you spot potential security issues. Checklists that capture the key review elements are provided.
- **Subject matter expertise.** The guidance exposes insight from various experts throughout Microsoft and from customers in the field.
- **Validation.** The guidance is validated internally through testing. Also, extensive reviews have been performed by product, field, and product support teams. Externally, the guidance is validated through community participation and extensive customer feedback cycles.

Audience

This guide is valuable for anyone who cares about application security objectives. It is designed to be used by team members from many different disciplines, including business analysts, architects, developers, testers, security analysts, and administrators. The guidance is task-based, and is centered on key security activities that should be performed at the various stages of the application life cycle.

How To Use This Guidance

You can read this guide from beginning to end, or you can read specific chapters to learn more about specific security engineering activities. You can adopt the security engineering activities described in this guide in their entirety for your organization,

or if your software engineering processes do not include specific security activities you can incrementally adopt activities. The activities you should adopt first will depend on the security objectives you have identified, as well as any outstanding problems your process or application currently has.

Ways to Use the Guide

There are many ways to use this guidance. The following are some ideas:

- **Use it to learn about security engineering.** Use the guide as an introduction and then use the companion Web-based resources at *http://msdn.com /securityengineering* to learn more.
- **Incorporate security engineering into your application life cycle.** Adopt the activities incrementally and incorporate them into your application life cycle.
- **Create training and promote security engineering within your organization.** Create training from the concepts and activities described in this guide and use it to promote security engineering within your organization.

Applying Guidance to Your Role

This guide applies to the following roles:

- **Business analysts and the management team.** Use techniques described in Chapter 2, "Security Objectives" to identify initial security objectives early in the life cycle.
- **Architects and lead developers.** Use the principles and best-practice design guidelines in Chapter 3, "Security Design Guidelines," to help architect and design systems capable of meeting security objectives. You can also use the threat modeling activity described in Chapter 4, "Threat Modeling" to help assess design choices before committing to a solution and you can use the review activity described in Chapter 5, "Security Architecture and Design Review" to review architectural and design decisions before costly mistakes are made.
- **Developers.** Use the security code review techniques highlighted in Chapter 6, "Security Code Review" to analyze your code and identify security issues before your code is tested.
- **Administrators and operations staff.** Use the deployment review techniques descried in Chapter 7, "Security Deployment Review" to ensure that configuration errors do not introduce security vulnerabilities at application deployment time.

Feedback

Provide feedback by using either a Wiki or e-mail:

- **Wiki.** Security guidance feedback page at *http://channel9.msdn.com/wiki/default.aspx/Channel9.SecurityGuidanceFeedback*
- **E-mail.** Send e-mail to secguide@microsoft.com.

We are particularly interested in feedback regarding the following:

- Technical issues specific to recommendations
- Usefulness and usability issues

The Team That Brought You This Guide

This guide was produced by the following individuals:

- J.D. Meier
- Alex Mackman
- Blaine Wastell
- Prashant Bansode
- Jason Taylor
- Rudolph Araujo

Contributors and Reviewers

Many thanks to the following contributors and reviewers:

- **External Contributors and Reviewers:** Anil John, Johns Hopkins University - Applied Physics Laboratory; Frank Heidt; Keith Brown Pluralsight LLC; Mark Curphey, Foundstone Professional Services
- **Microsoft Services and PSS Contributors and Reviewers:** Adam Semel, Denny Dayton, Gregor Noriskin, Kate Baroni, Tom Christian, Wade Mascia
- **Microsoft Product Group:** Charlie Kaufman, Don Willits, Mike Downen, Rick Samona
- **Microsoft IT Contributors and Reviewers:** Akshay Aggarwal, Irfan Chaudhry, Shawn Veney, Talhah Mir
- **MSDN Contributors and Reviewers:** Kent Sharkey
- **Microsoft EEG:** Corey Ladas, James Waletzky

- **Test team:** Larry Brader, Microsoft Corporation; Nadupalli Venkata Surya Sateesh, Sivanthapatham Shanmugasundaram, Infosys Technologies Ltd.
- **Edit team:** Nelly Delgado, Microsoft Corporation; Sharon Smith, Linda Werner & Associates
- **Release Management:** Sanjeev Garg, Microsoft Corporation

Tell Us About Your Success

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

Conclusion

The patterns & practices approach to security engineering focuses on integrating security into your life cycle through the adoption of a limited set of key security activities. The specific activities that make up the security engineering discipline include defining security objectives, applying design guidelines for security, creating threat models, conducting architecture and design reviews for security, completing code reviews for security, and performing deployment reviews for security.

This chapter introduced you to the *Security Engineering Explained* guide. Subsequent chapters in this guide explain each of the security engineering activities in more detail.

Additional Resources

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

1

patterns & practices Security Engineering Approach

Summary

This chapter summarizes the patterns & practices approach to security engineering. To design, build, and deploy secure applications, you must integrate security into your application development life cycle by including specific security-related activities in your current software engineering processes. Security-related activities include identifying security objectives; applying secure design guidelines, patterns, and principles; conducting architecture and design reviews for security; creating threat models; performing regular code reviews for security; testing for security; and conducting deployment reviews to ensure secure configuration. You can adopt these activities incrementally as you see fit. These security activities are integrated in MSF Agile, available with Visual Studio® Team System. This combination provides tools, guidance, and workflow to help make security a seamless part of your development experience.

Overview

The patterns & practices approach to security engineering can be summarized as follows:

- **Integrate security into your lifecycle.** Upfront security design, secure coding practices, and testing for security must all be an integral part of your application development processes.
- **Identify your objectives.** Understand early what the security objectives are for your application. These objectives will play a critical role in shaping threat modeling, code reviews, and testing.

- **Know your threats.** Analyze your application in a structured and systematic way to recognize its threats and vulnerabilities. The threat modeling activity enables you to identify and understand threats, understand the risks each threat poses, and uncover vulnerabilities that you can use to shape subsequent security design and implementation decisions.
- **Use an iterative approach.** Some activities, such as code review, threat modeling and security testing should be performed multiple times during the development process to maximize application security.

Key Activities in the Life Cycle

Figure 1.1 shows the key security engineering activities and where they should be integrated into the application life cycle.

Security Overlay

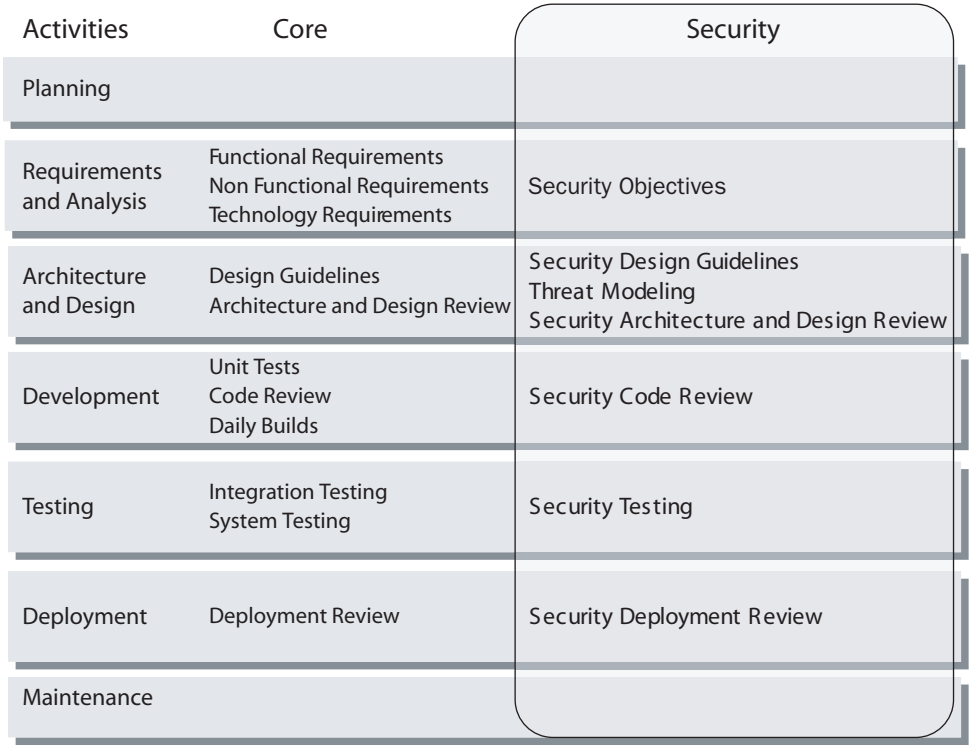

Figure 1.1
Security engineering activities in the application development life cycle

There are a number of distinct security-related activities that should be an integral part of your application life cycle. These are:

- **Security Objectives.** Define security objectives and requirements early in the process. Security objectives are goals and constraints that affect the confidentiality, integrity, and availability of your data and application.
- **Design Guidelines for Security.** To avoid many of the vulnerabilities introduced by poor design choices, your design activity should use proven design practices, patterns, and principles. By organizing these design patterns and practices into common vulnerability categories, you can focus on those areas where security mistakes are most often made.
- **Threat Modeling.** Threat modeling helps you to understand and identify the threats and vulnerabilities relevant to your specific application scenario.
- **Architecture and Design for Security.** The architecture and design review process analyzes the architecture and design from a security perspective. It examines a number of aspects including deployment and infrastructure, overall application architecture and design, and each tier in the application.
- **Code Review for Security.** All code should be subject to code inspections where the emphasis is on identifying security vulnerabilities. This should be a continuous activity during the development and test phases of the application life cycle.
- **Security Testing.** Use a risk-based approach and use the output from the threat modeling activity to help establish the scope of your testing activities and define your test plans.
- **Deployment Review for Security.** When your application is deployed, you need to be sure that weak or inappropriate configuration settings do not introduce security vulnerabilities.

How the Security Activities Work Together

Security-related activities start early and continue throughout the application life cycle, many in parallel with one another.

Figure 1.2 on the next page shows how security activities span the various **activities** of the application development life cycle.

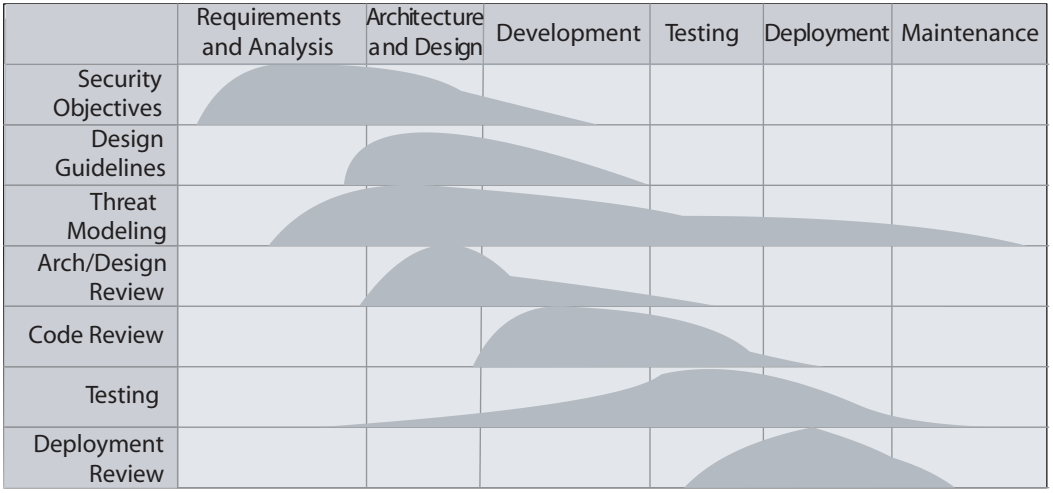

Figure 1.2
Security activities in the application development life cycle

Allow the results of each activity to influence the others in order to have a security engineering process that is more effective than the sum of its parts. For example:

- Your security objectives should be considered alongside other critical business objectives. Application specific security objectives should be identified and documented early during requirements and analysis and should be balanced along side other quality of service requirements such as performance, availability and reliability.
- Using security design guidelines will mitigate many threats found during the threat modeling process.
- Threat modeling allows you to identify threats and vulnerabilities. The identified vulnerabilities and subsequent mitigations should be used to shape and influence subsequent design, development, and testing decisions.
- Issues found during code review and testing may result in new threats added to the threat model which in turn will drive new ideas for testing and code review.

Incremental Adoption

If your current software engineering processes do not include specific security activities, it is possible to incrementally adopt the key security activities. The activities you should adopt first will depend on the security objectives you have identified, as well as any outstanding problems your process or application currently has.

For most organizations, the best results will come from adopting the activities in the following order:

- **Security Objectives.** If you do not know the security objectives for your application, it will be difficult to be successful with any other activity.
- **Architecture and Design Review for Security.** Bugs introduced in the design phase are the most expensive to deal with later. By introducing architecture and design reviews focused on security, you avoid the need for costly rework later in the life cycle.
- **Threat Modeling.** By adopting threat modeling, in addition to helping you focus your security development efforts, improving the overall quality of your software engineering, and ensuring that you address relevant threats, you can help your test teams create test plans to test for specific vulnerabilities. Threat models also serve as a focus for communication among the various roles and help to ensure that developers and IT professionals alike really understand the application.
- **Code Review for Security.** While design bugs are the most expensive, implementation bugs are the most common. Reviewing your code for security vulnerabilities can save you later rework or help avoid costly exploits.
- **Security Review for Deployment.** An application is only as secure as its weakest link. Even a highly effective process can be undone by a configuration error during deployment.
- **Design Guidelines for Security.** By adopting proven design principles and learning from others mistakes you can ensure your application is secure from the start.
- **Security Testing.** Testing should be used to validate designed mitigations and ensure nothing has slipped through the cracks.

Terminology

To get the most out of the content in this document, it is important to understand the difference between threats, vulnerabilities, attacks, and countermeasures.

- **Threat.** A threat is an undesired event. A potential occurrence, often best described as an effect that might damage or compromise an asset or objective. It may or may not be malicious in nature.
- **Vulnerability.** A vulnerability is a weakness in some aspect or feature of a system that makes an exploit possible. Vulnerabilities can exist at the network, host, or application levels and include operational practices.
- **Attack (or exploit).** An attack is an action taken that uses one or more vulnerabilities to realize a threat. This could be someone following through on a threat or exploiting a vulnerability.

- **Countermeasure.** Countermeasures address vulnerabilities to reduce the probability of attacks or the impacts of threats. They do not directly address threats; instead, they address the factors that define the threats. Countermeasures range from improving application design, or improving your code, to improving an operational practice.

Conclusion

The patterns & practices approach to security engineering focuses on integrating security into your life cycle through the adoption of a limited set of key security activities. It uses a pattern-based information model in the form of a set of vulnerability categories to help you systematically focus your efforts on areas where mistakes are most often made.

The specific activities that make up the security engineering discipline include defining security objectives, applying design guidelines for security, creating threat models, conducting architecture and design reviews for security, completing code reviews for security, and performing deployment reviews for security.

Additional Resources

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

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. This module summarizes the patterns & practices approach to security objectives by explaining what they are and why you should use them. It also describes the key concepts behind the approach.

Overview

Security objectives should be identified as early in the development process as possible, ideally in the requirements and analysis phase. Security objectives are critically important. If you do not know what the objectives are for your application, then it is difficult to be successful with any other security activity.

Security objectives are used to:

- Filter the set of design guidelines that are applicable.
- Guide threat modeling activities.
- Determine the scope and guide the process of architecture and design reviews.
- Help set code review objectives.
- Guide security test planning and execution.
- Guide deployment reviews.

In each activity, you can use the security objectives to help you focus on the highest value areas while avoiding issues that will not affect your application.

Discovery Process

Identifying security objectives is an iterative process that is initially driven by an examination of the application's requirements and usage scenarios. By the end of the requirements and analysis phase, you should have a first set of objectives that are not yet tied to design or implementation details. During the design phase, additional objectives will surface that are specific to the application architecture and design. During the implementation phase, you may discover a few additional objectives based upon specific technology or implementation choices that have an impact on overall application security.

Each evolution of the security objectives will affect other security activities. You should review the threat model, architecture and design review guidelines, and general code review guidelines when your security objectives change.

Types of Objectives

Security objectives are unique for each application. However, there is a set of common categories of objectives that you can use to jump-start the identification process. Each of these categories is associated with a set of questions you can use to build your objective list. Table 2.1 lists the security objective categories and associated questions.

Table 2.1. Application Vulnerabilities and Potential Problems

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?

Techniques

Use the following techniques to help you discover security objectives:

- Roles Matrix
- Derive From Functional Requirements

Roles Matrix

When an application supports multiple roles it is important to understand what each role should be allowed to do. This can be accomplished with a roles matrix that contains privileges in rows and roles in columns such as the one shown in Table 2.2.

Table 2.2: Role Matrix Example

	Objects				
Subjects	User creation	Permission modification	Object creation	Object removal	Object read
Admin	✓	✓			
Content creator			✓	✓	✓
Reader					✓
Anonymous					✓

Once the roles matrix has been created, you can generate security objectives to ensure the integrity of the application's roles mechanism. For example, the following objectives could be generated from the preceding matrix:

- Anonymous users should not be allowed to create users, modify permissions, create objects, or remove objects
- Content creator users should not be allowed to modify permissions or create users

Many systems have multiple roles and privileges can be assigned flexibly to any role. In this case your objectives need to be more general:

- Roles should only have access to the privileges to which they are assigned.
- Privilege changes should only be allowed by a role that has the 'permission modification' privilege.

Derive From Functional Requirements

You can generate security objectives by examining every functional requirement in your application through the lens of confidentiality, integrity, and availability (CIA). This provides a very effective mechanism for generating security objectives based on known application characteristics.

Consider the following abbreviated set of application requirements:

- Administrator must be able to add users and change user permissions through an administrator interface.
- Each application server must be able to serve 1000 concurrent users.
- Registered users must be able to purchase books from the Web user interface.

Examine each requirement while thinking about confidentiality, integrity and availability. The first step is to derive potential vulnerabilities from the requirement. For instance, the following vulnerabilities could be derived from the first requirement above:

- **Confidentiality.** User names and permissions are exposed to non-administrator.
- **Integrity.** Non-administrator is able to add users and change user permissions.
- **Availability.** Administrator is not able to add users and change user permissions.

Once you understand the potential vulnerabilities you can derive security objectives. Table 2.3 shows example output from this process.

Table 2.3 Objectives Derived from Functional Requirements

Original Requirement	Derived Security Objectives
Administrator must be able to add users and change user permissions through an administrator interface.	**C –** Protect user names and permissions from disclosure to a non-administrator **I –** Restrict non-administrator from adding users and changing user permissions **A –** Application is available so that the administrator is able to add users and change user permissions
Each application server must be able to serve 1000 concurrent users.	**A –** Application server is available for users
Registered users must be able to purchase books from the Web user interface	**C –** Protect registered user credentials and other sensitive information from non-administrators and non-registered users **I –** Restrict non-registered user from purchasing books from the Web user-interface **I –** Restrict registered users from obtaining books for the wrong price **A –** Web user interface is available for registered users to purchase books

Note: A requirement may generate zero or more objectives for confidentiality, integrity, or availability.

Conclusion

Identification of security objectives is the first, best step you can take to help ensure the security of your application. The objectives, once created, can be used to direct all the subsequent security activities that you perform. Security objectives do not remain static, but are influenced by later design and implementation activities.

3

Security Design Guidelines

Summary

Design guidelines represent proven practices that have evolved over time to reduce risks associated with designing applications. Adopting security design guidelines can help reduce your attack surface by addressing common vulnerabilities, allowing you to focus on the unique aspects of your design. This module summarizes the patterns & practices approach to security design guidelines by explaining what it is and why you should use it. It also describes the key concepts behind the approach.

Overview

Designing a secure application can present architects and developers with many challenges. Design guidelines represent the set of practices that can be employed to reduce the risk of security vulnerabilities.

Each guideline must meet the following qualifications before it is included:

- **Actionable.** Must be associated with a vulnerability that can be mitigated through the use of the guideline.
- **Relevant.** Must be associated with a vulnerability that is known to affect real applications.
- **Impactful.** Must represent key engineering decisions that will have a wide-ranging impact.

The set of guidelines is distilled into a pattern-based security frame, or framework, that describes all of the areas in which poor design can lead to security vulnerabilities. The security frame allows the inclusion of additional guidelines or the refinement of existing guidelines based on newly discovered vulnerabilities.

Design guidelines include both deployment considerations and design considerations.

Security Frame

The security frame is 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. Patterns & practices security guidance includes context-specific security frames for each major application type.

Categories of Vulnerabilities

Design guidelines are organized by the common application vulnerability categories contained in the security frame, including input / data validation, authentication, authorization, configuration management, sensitive data, cryptography, exception management and auditing and logging. These represent the key areas for application security design, where mistakes are most often made.

Table 3.1 lists the set of vulnerability categories that is common to most application types along with the associated potential problems that can occur due to bad design.

Table 3.1. Application Vulnerabilities and Potential Problems

Vulnerability Category	Potential Problem Due to Bad Design
Input / Data Validation	Insertion of malicious strings in user interface elements or public APIs. These attacks include command execution, cross - site scripting (XSS), SQL injection, and buffer overflow. Results can range from information disclosure to elevation of privilege and arbitrary code execution.
Authentication	Identity spoofing, password cracking, elevation of privileges, and unauthorized access.
Authorization	Access to confidential or restricted data, data tampering, and execution of unauthorized operations.
Configuration Management	Unauthorized access to administration interfaces, ability to update configuration data, and unauthorized access to user accounts and account profiles.
Sensitive Data	Confidential information disclosure and data tampering.
Cryptography	Access to confidential data or account credentials, or both.
Exception Management	Denial of service and disclosure of sensitive system-level details.
Auditing and Logging	Failure to spot the signs of intrusion, inability to prove a user's actions, and difficulties in problem diagnosis.

Application-Specific Guidelines

Depending on the application being designed, the types of issues that must be addressed vary. The categories defined in each application-specific security frame were defined by security experts who have examined and analyzed the top security issues across many applications. They have been refined with input from Microsoft consultants, product support engineers, customers, and Microsoft partners.

For example, when you design a secure Web application, it is important that you follow guidelines to ensure effective user authentication and authorization, to protect sensitive data as it is transmitted over public networks, and to prevent attacks such as session hijacking.

Some of the important Web application issues that must be addressed with secure design practices are shown in Figure 3.1.

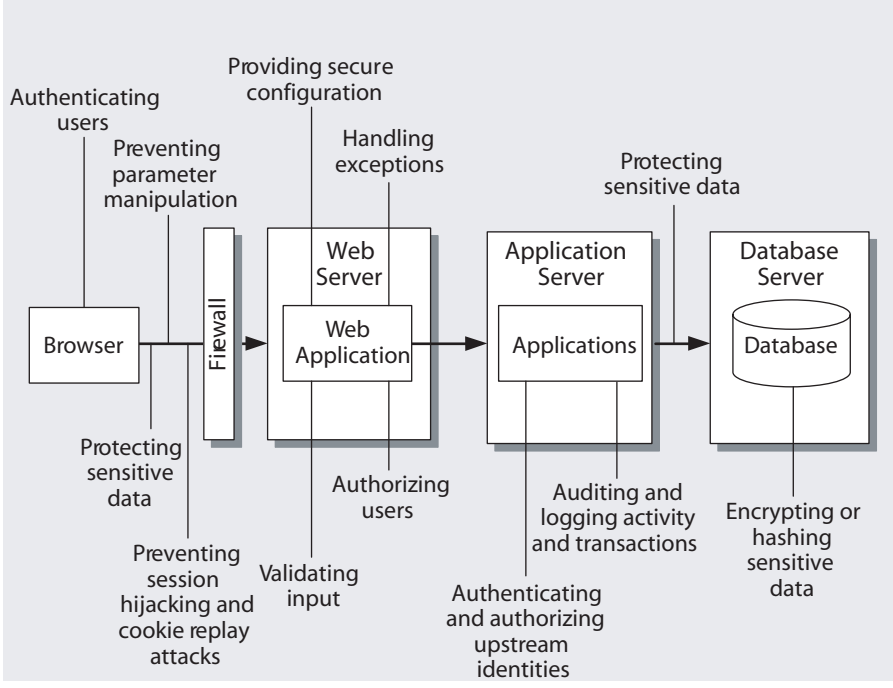

Figure 3.1
Web application design issues

When you design a secure, smart client application, the set of guidelines changes to address the most important threats for this application type. For example, authentication and authorization are no longer such important concerns; however, input / data validation and exception handling are.

Deployment Considerations

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

Figure 3.2 shows the various deployment aspects that require design-time consideration.

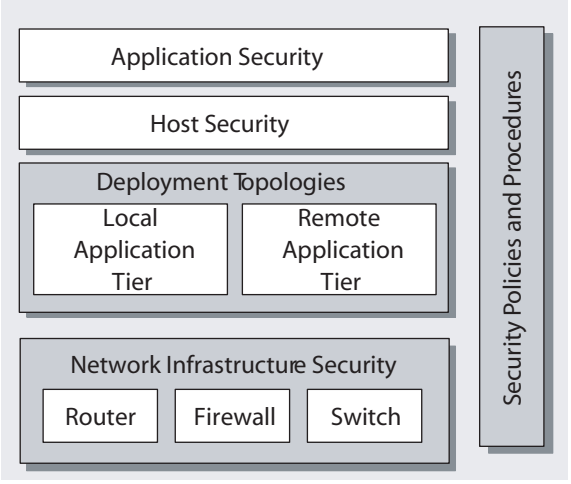

Figure 3.2
Deployment considerations

Design Guidelines Summary

Design guidelines are best practices associated with specific known vulnerabilities and common design mistakes. Patterns & practices security guidance contains design guidelines for a variety of application types. Table 3.2 summarizes general design guidelines that are common to most application types.

Table 3.2. Design Guidelines for Your Application

Category	Guidelines
Input / Data Validation	Do not trust input; consider centralized input validation. Do not rely on client-side validation. Be careful with canonicalization issues. Constrain, reject, and sanitize input. Validate for type, length, format, and range.
Authentication	Use strong passwords. Support password expiration periods and account disablement. Do not store credentials (use one-way hashes with salt). Encrypt communication channels to protect authentication tokens.
Authorization	Use least privileged accounts. Consider authorization granularity. Enforce separation of privileges. Restrict user access to system-level resources.
Configuration Management	Use least privileged process and service accounts. Do not store credentials in clear text. Use strong authentication and authorization on administration interfaces. Do not use the Local Security Authority (LSA). Secure the communication channel for remote administration.
Sensitive Data	Avoid storing secrets. Encrypt sensitive data over the wire. Secure the communication channel. Provide strong access controls for sensitive data stores.
Cryptography	Do not develop your own. Use proven and tested platform features. Keep unencrypted data close to the algorithm. Use the right algorithm and key size. Avoid key management (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 centralized exception management framework.
Auditing and Logging	Identify malicious behavior. Know what good traffic looks like. Audit and log activity through all of the application tiers. Secure access to log files. Back up and regularly analyze log files.

Conclusion

Design guidelines can be used as a tool to help you meet your application security objectives. The security frame provides a structure within which you can modify or add design guidelines as you learn about your application's architecture and deployment environment. Patterns & practices security guidance includes a core set of guidelines, organized by application type that you can use as a starting point for your application's guidelines. Each guideline should be actionable, impactful, and relevant; and guidelines should be organized into the categories contained in the security frame.

Additional Resources

For more information, see "Chapter 4, Design Guidelines for Secure Web Applications" at *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh04.asp*.

4

Threat Modeling

Summary

Threat modeling is an engineering technique that you can use to help identify threats, attacks, vulnerabilities, and countermeasures that may be relevant to your application. This module summarizes the patterns & practices approach to threat modeling by explaining what it is and why you should use it. It also describes the key concepts behind the approach.

What Is Threat Modeling?

Threat modeling is an engineering technique that you can use to help identify threats, attacks, vulnerabilities, and countermeasures that might be relevant to your application. The threat modeling activity helps you to recognize the following:

- Your security objectives
- Relevant threats
- Relevant vulnerabilities and countermeasures

Why Use Threat Modeling?

Threat modeling is performed to identify when and where more resources are required (or justified) to reduce risks. There are many possible vulnerabilities, threats, and attacks, and it is unlikely that your application will encounter all of them. It is also unlikely that your company would need to address all of them. Threat modeling helps you to identify where your organization needs to apply effort.

Threat modeling helps you to:

- Shape your application design to meet your security objectives.
- Help make engineering decisions that weigh security goals against other design goals.
- Address and improve the quality of your engineering efforts by eliminating common vulnerabilities.
- Improve quality of service by implementing necessary and effective countermeasures.
- Reduce risk of security issues arising during development and operations.

Where Does Threat Modeling Fit In?

Threat modeling begins early in the architecture and design phase of the application life cycle. It is performed with knowledge of your security objectives. Your security objectives are a key part of your business objectives and they are used to determine the extent of the threat modeling activity and where to spend the most effort.

As the life cycle progresses and your design and development evolve, you progressively add more detail to your threat model. Threat modeling is iterative, and you need to repeat the process as indicated by the following:

- Increase the detail of your model whenever new environmental facts become known.
- While you develop, increase the detail of your model as design and implementation decisions reveal new facts.
- Add detail as new uses and configurations of the application appear. This is during the application lifetime including after the application is in production and is maintained by the operations team.

Add information to the model as it becomes available to you, and keep identifying what you now know and what you need to know next.

Terminology

Threat modeling uses the following terms:

- **Asset.** An asset is a resource that has value. It varies by perspective. To your business, an asset might be the availability of information or the information itself, such as customer data. It might be intangible, such as your company's reputation. To an attacker, an asset could be the ability to misuse your application for unauthorized access to data or privileged operations.
- **Threat.** A threat is an undesired event or potential occurrence, often best described as an effect that could damage or compromise an asset or objective. It may or may not be malicious in nature.
- **Vulnerability.** A vulnerability is a weakness in some aspect or feature of a system that makes an attack possible. Vulnerabilities can exist at the network, host, or application level and include operational practices.
- **Attack (or exploit).** An attack is an action taken that uses one or more vulnerabilities to realize a threat. This could be someone following through on a threat or exploiting a vulnerability.
- **Countermeasure.** A countermeasure addresses a vulnerability to reduce the probability of attack or the impact of a threat. A countermeasure does not directly address a threat. Instead, it addresses the factors that define the threat. Countermeasures range from improving application design or improving code, to improving an operational practice.

Activity Overview

The five major threat modeling steps are shown in Figure 4.1 on the next page. You should progressively refine your threat model by repeatedly performing steps 2 through 5. You will be able to add more detail as you move through your application development life cycle and discover more about your application design.

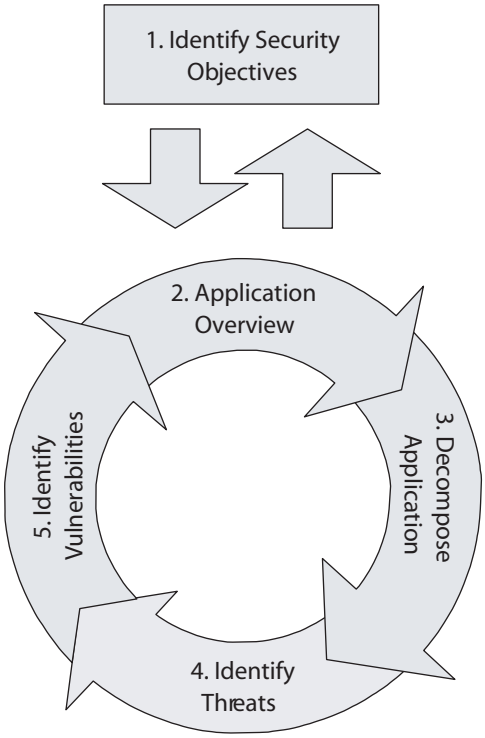

Figure 4.1
The iterative threat modeling process

The five threat modeling steps are:

- **Step 1: Identify security objectives.** Clear objectives help you to focus the threat modeling 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 in the context of your threats. Use vulnerability categories to help you to focus on those areas where mistakes are most often made.

You add progressively more detail to your threat model as you move through your application development life cycle and discover more details about your application design. Because key resources identified in threat modeling are also likely to be important from a performance and functionality perspective, you can expect to revisit and adjust your model as you balance your needs. This is normal and is a valuable outcome of the process.

Activity Summary Table

Table 4.1 summarizes the threat modeling activity and shows the input and output for each step.

Table 4.1: Activity Summary with Input and Output

Input	Step	Output
• Business requirements • Security policies • Compliance requirements	**Step 1: Identify security objectives**	• Key security objectives
• Deployment diagrams • Use cases • Functional specifications	**Step 2: Create an application overview**	• Whiteboard-style diagram with end-to-end deployment scenario • Key scenarios • Roles • Technologies • Application security mechanisms
• Deployment diagrams • Use cases • Functional specifications • Data flow diagrams	**Step 3: Decompose your application**	• Trust boundaries • Entry points • Exit points • Data flows
• Common threats	**Step 4: Identify threats**	• Threat list
• Common vulnerabilities	**Step 5: Identify vulnerabilities**	• Vulnerability list

Key Concepts

The patterns & practices threat modeling approach is optimized to help you identify vulnerabilities in your application. The key concepts on the next page represent best practices refined by industry security experts, Microsoft consultants, product support engineers, customers, and Microsoft partners.

Table 4.2: Key Concepts

Concept	Description
Modeling to reduce risk	Use threat modeling to identify when and where you should apply effort to eliminate or reduce a potential threat. Avoid wasted effort by threat modeling to clarify the areas of most risk.
Incremental rendering	Perform threat modeling iteratively. You should not be too concerned about missing details in any single iteration — instead focus on making each iteration productive.
Context precision	Understand application use cases and roles in order to identify threats and vulnerabilities that are specific to your application. Different application types, application usage, and roles can yield different threats and vulnerabilities.
Boundaries	Establish boundaries in order to help you define constraints and goals. Boundaries help you identify what must not be allowed to happen, what can happen, and what must happen.
Entry and exit criteria	Define entry and exit criteria to establish tests for success. You should know before you start what your threat model will look like when complete (good enough) and when you have spent the right amount of time on the activity.
Communication and collaboration tool	Use the threat model as a communication and collaboration tool. Leverage discovered threats and vulnerabilities to improve shared knowledge and understanding.
Pattern-based information model	Use a pattern-based information model to identify the patterns of repeatable problems and solutions, and organize them into categories.
Primary engineering decisions	Expose your high-risk engineering decisions and design choices with your threat model. These high-risk choices are good candidates for focusing prototyping efforts.

Conclusion

Threat modeling is a structured activity for identifying and evaluating application threats and vulnerabilities. You should start the threat modeling activity early in the architecture and design phase of your application life cycle, and then continually update and refine the model as you learn more about your design and implementation. Use threat modeling to shape your application design to meet your security objectives, to help weigh the security threat against other design concerns (such as performance) when making key engineering decisions, and to reduce the risk of security issues arising during development and operations.

Additional Resources

For more information on threat modeling, see *http://msdn.com/threatmodeling*.

5

Security Architecture and Design Review

Summary

This module summarizes the patterns & practices approach to security architecture and design review by explaining what it is and why you should use it. It also describes the key concepts behind the approach.

There are three important aspects to conducting an architecture and design review for security:

- You evaluate your application architecture in relation to its target deployment environment and infrastructure.
- You review your design choices in each of the key vulnerability categories defined by a security frame.
- Finally, you conduct a tier-by-tier component analysis and examine the security mechanisms employed by your key components, such as your presentation layer, business layer, and data access layer.

Overview

The architecture and design review process analyzes application architecture and design from a security perspective. Use this activity to expose the high-risk design decisions that have been made. Do not rely solely on the use of design documentation as some design decisions will not be explicit but will have to be discovered

through dialog and exploration. Use a combination of design documents, architecture experts and discussion to achieve the best results. The goal of the review is to decompose your application and identify key items, including trust boundaries, data flow, entry points, and privileged code. You must also keep in mind the physical deployment configuration of your application.

Pay attention to those areas defined by the security frame for your application as this is where you will most commonly find vulnerabilities. Patterns & practices security guidance includes context-specific security frames for each major application type that you can use to get started.

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.

- **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.
- **Security frame.** Review the approach to critical areas in your application, including authentication, authorization, input / data validation, exception management, and other areas. Use the application vulnerability categories as a roadmap and to make sure that you do not miss any key areas during the review.
- **Tier-by-tier analysis.** Walk through the logical tiers of your application, and evaluate security choices within your presentation, business, and data access layers.

Figure 5.1 shows this three-pronged approach to the review process.

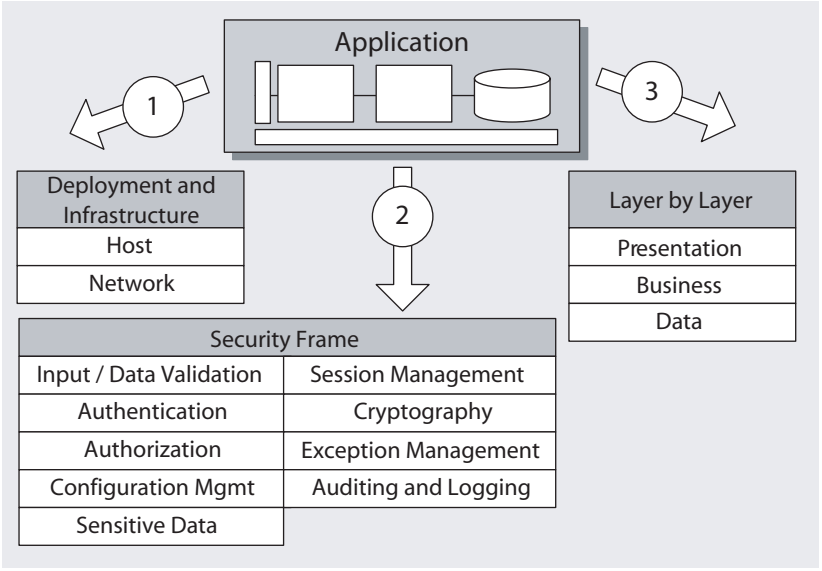

Figure 5.1
Application review

Checklists

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.

Patterns & practices security guidance includes checklists for each major application type. Use these checklists as a starting point and add new items as you learn more about architecture and design reviews. A small sample checklist containing items that are applicable to many applications is shown below. To view a complete checklist, refer to patterns & practices guidance on the Web at *http://msdn.microsoft.com/library/en-us/dnpag2/html/SecurityChecklistsIndex.asp.*

Deployment and Infrastructure Considerations

Check	Description
❑	The design identifies, and accommodates the company security policy.
❑	Restrictions imposed by infrastructure security (including available services, protocols, and firewall restrictions) are identified.
❑	The design recognizes 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.
❑	Secure communication features provided by the platform and the application are known.
❑	The design addresses the required scalability and performance criteria.

Application Architecture and Design Considerations

Input / Data 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.
❑	Centralized 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 sanitize 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 canonicalization issues.
❑	Input file names and file paths are avoided where possible.

Check	Description
❑	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 defense in depth to the input validation strategy by providing input validation across tiers.

Conclusion

If you spend time and effort at the beginning of your project to analyze and review your application architecture and design, you can eliminate design-related vulnerabilities and improve your application's overall security. It is much easier and less expensive to fix vulnerabilities at design time than it is later in the development cycle when substantial re-engineering might be required.

Use the architecture and design review as a checkpoint to ensure compliance with your team's design guidelines. The review should cover both deployment and design considerations. By considering your design in relation to the target deployment environment and the security policies defined by that environment, you can help to ensure a smoother and more secure application deployment.

Perform architecture and design reviews iteratively as your design evolves or as you learn more about the design and architecture of your applications. If your application has already been created, the architecture and design review is still an important part of the security assessment process that can help you to fix vulnerabilities and improve future designs.

Additional Resources

For more information, see "Security Architecture and Design Review Index" at *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/securityarchanddesignreviewindex.asp.*

6

Security Code Review

Summary

Security code review is an effective mechanism for uncovering security issues 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.

This module summarizes the patterns & practices approach to code review by explaining what it is and why you should use it. It also describes the key concepts behind the approach.

Overview

A properly conducted code review can do more for the security of your code than nearly any other activity. A large numbers of bugs can be found and fixed before the code makes it into an official build or into the hands of the test team. Additionally, the code review process lends itself very well to sharing security best practices among a development team, and it produces lessons that can be learned from to prevent future bugs.

To have an effective code review, you must first understand the patterns of bad code that you want to eradicate, and then review the code with a clear idea of what you are looking for. You can use security objectives to guide the code review process. Some vulnerability types may have elevated priority and others may be out of bounds based upon these objectives. Threat models can be used to create a more focused code review. Reviewing for specific known threats is more likely to result in bugs than a generic review that could be applied to any application.

Activity Overview

The four code review steps are shown in Figure 6.1. Each step builds upon the last, producing a result that is greater than the sum of its parts.

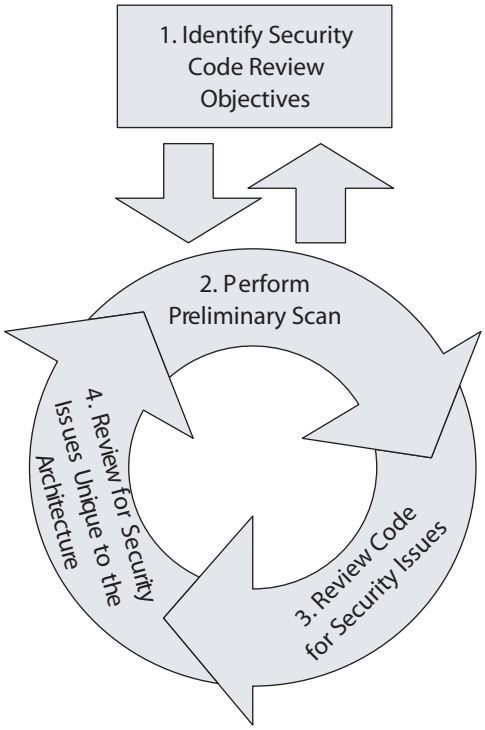

Figure 6.1
Code Review Steps

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.

Activity Summary Table

Table 6.1 summarizes the security code review activity and shows the input and output for each step.

Table 6.1: Activity Summary with Input and Output

Input	Step	Output
• Security Requirements • Code (including list of changes since last review) • Constraints	**Step 1. Identify code review objectives**	• Code review objectives
• Code • Code review objectives	**Step 2. Perform preliminary scan**	• Vulnerability list (false positives filtered out) • List of flagged areas
• Code • Code review objectives • List of flagged areas	**Step 3. Review code for security issues**	• Vulnerability list
• Code • Code review objectives	**Step 4. Review code for security issues unique to the application architecture**	• Vulnerability list

Techniques

An effective code reviewer makes use of the following techniques while reviewing code. These techniques are most effective when used in combination.

- **Control flow analysis.** Control flow analysis is the mechanism used to step through logical conditions in the code. The process works as follows:
  1. Look at a function and determine each branch condition. These can include loops, **switch** statements, **if** statements and **try/catch** blocks.
  2. Understand the conditions under which each block will be executed.
  3. Move to the next function and repeat.
- **Dataflow analysis.** Dataflow analysis is the mechanism used to trace data from the points of input to the points of output. Because there can be many data flows in your application, use your code review objectives and the flagged areas from step 2 to focus your work. The process works as follows:
  1. For each input location, determine how much you trust the source of input. When in doubt you should give it no trust.
  2. Trace the flow of data to each possible output, noting along the way any attempts at data validation.
  3. Move to the next input and continue.

Use a Question List

Using a question-driven approach can help with the review process. Patterns & practices security guidance includes question lists for each major application type. These lists each contain a set of questions that are known to be effective during code review. Apply these questions while using control flow and dataflow analysis, and add to the questions as you learn more about reviewing code. Keep in mind that some vulnerabilities require contextual knowledge of control and data flow while others are context free and can be found with simple pattern matching.

Hotspots

Each question list is organized into a set of *hotspots* or trouble areas that are based on implementation mistakes that most commonly result in application vulnerabilities. Table 6.2 shows example hotspots.

Table 6.2: Hotspots

Hotspot	What to look for in code
SQL Injection	A SQL injection attack occurs when un-trusted input can modify the logic of a SQL query in unexpected ways. As you are work through the code, ensure that any input that is used in a SQL query is validated, or make sure that the SQL queries are parameterized.
Cross Site Scripting	Cross-site scripting occurs when an attacker manages to input script code into an application so that it is echoed back and executed in the security context of the application. This allows an attacker to steal user information, including forms data and cookies. This vulnerability could be present whenever a Web application echoes unfiltered user input back to Web content.
Data Access	Look for improper storage of database connection strings and proper use of authentication to the database.
Input/Data Validation	Look for client-side validation that is not backed by server-side validation, poor validation techniques, and reliance on file names or other insecure mechanisms to make security decisions.
Authentication	Look for weak passwords, clear-text credentials, overlong sessions, and other common authentication problems.
Authorization	Look for failure to limit database access, inadequate separation of privileges, and other common authorization problems.
Sensitive data	Look for mismanagement of sensitive data by disclosing secrets in error messages, code, memory, files, or network.
Unsafe code	Pay particularly close attention to any code compiled with the **/unsafe** switch. This code will not be given all of the protection normal managed code is given. Look for potential buffer overflows, array out of bound errors, integer underflow and overflow, as well as data truncation errors.

Hotspot	What to look for in code
Unmanaged code	In addition to the checks performed for unsafe code, also scan unmanaged code looking for the use of dangerous APIs, such as **strcpy** and **strcat**. Be sure to review any interop calls and the unmanaged code itself to make sure that bad assumptions are not made as execution control passes from managed to unmanaged code.
Hard-coded secrets	Look for hard-coded secrets by examining the code for variable names such as "key", "password", "pwd", "secret", "hash", and "salt".
Poor error handling	Look for functions with missing error handlers or empty **catch** blocks.
Web.config	Examine your configuration management settings in the Web.config file to ensure that forms authentication tickets are protected adequately, that the correct algorithms are specified on the **machineKey** element and so on.
Code access security	Search for the use of asserts, link demands and **allowPartiallyTrustedCallersAttribute** (APTCA).
Code that uses cryptography	Check for failure to clear secrets as well as improper use of the cryptography APIs themselves.
Undocumented public interfaces	Most undocumented interfaces should not be in the product, and they are almost never given the same level of design and test scrutiny as the rest of the product.
Threading problems	Check for race conditions and deadlocks, especially in static methods and constructors.

Code Review Scenarios

There are several strategies for conducting code reviews, including:

- **Individual Review.** This strategy assumes that a single person will review the code.
- **Team Review.** This strategy assumes that multiple people will review the same code. This can be a highly effective code review strategy, but it requires additional organization to be successful.

In either strategy, you can select a reviewer who is familiar or unfamiliar with the code.

The advantage of using a reviewer who does not have prior knowledge of the code is that he or she will examine the code with fresh eyes and will be less likely to make assumptions than someone who is more familiar might be.

The advantage of using a reviewer with knowledge of the code is that he or she will be able to find subtle errors that require expert familiarity with the application under review.

Team Roles

During a code review, there are a several distinct tasks:

- **Present the objectives.** Describe the application/component purpose and security objectives.
- **Present the code.** Walk through the code and describe its design and intent.
- **Review the code.** Review the code and find bugs.
- **Take notes.** Take notes that describe the bugs found, any open questions, and areas for future investigation.

Typically, the following roles are involved:

- **Business analyst.** The business analyst describes the application purpose and security objectives. He or she is usually responsible for the presenting the objectives from a business standpoint.
- **Architect.** The architect describes the application purpose and component architecture. He or she is usually responsible for presenting the objectives from a systems architecture standpoint.
- **Developer.** The developer describes code details, reviews the code, and finds bugs. He or she is usually responsible for presenting the code and reviewing the code.
- **Tester.** The tester reviews the code and finds bugs, and typically has less code knowledge but a more destructive, break-the-code mind-set than the developer has. He or she is usually responsible for reviewing code and recording bugs.

Conclusion

Security code reviews are a powerful tool to reduce the number of vulnerabilities in your application. Through the use of control flow and dataflow analysis in conjunction with a question list it is possible to find and fix implementation bugs before they are delivered to your test team or customer. Use the lessons learned in code review to update the question list and spread secure development best practices through the development team.

Additional Resources

For more information, see "patterns & practices Security Code Review Index" at *http://msdn.microsoft.com/library/en-us/dnpag2/html/SecurityCodeReviewIndex.asp.*

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.

This module summarizes the patterns & practices approach to security deployment reviews by explaining what they are and why you should use them. It also describes the key concepts behind the approach.

Overview

Application security is dependent upon the security of the underlying infrastructure on which the application is deployed. The deployment review, depending upon your application, will cover configuration of both the network and the host.

When you review your security deployment, you can organize the precautions you must take and the settings you must configure into categories. These categories are shown in Figure 7.1 on the next page.

By using these configuration categories, you can systematically review the entire application, or pick a particular category and complete specific steps.

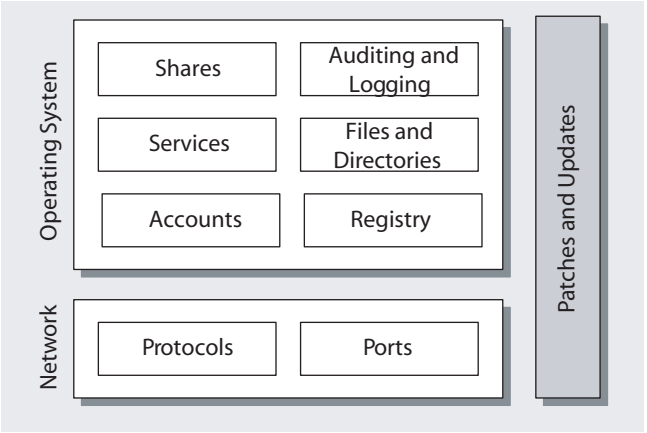

Figure 7.1
Server configuration categories

Techniques

Use following techniques when conducting a deployment review:

- **Use server security categories.** Use server security categories to help make deployment reviews for security systematic and repeatable.
- **Break down your deployment review.** You can use the categories to break down your application deployment for further analysis and to help identify vulnerabilities.
- **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.

Server Security Categories

Patterns & practices security guidance includes server security categories for each major application type. Use these categories as a starting point and add new items as you learn more about deployment reviews. Table 7.1 lists categories that are common to most deployed applications.

Table 7.1 Server Security Categories

Category	Practices
Patches and Updates	Patching and updating your server software is a critical first step.
Accounts	Accounts allow authenticated users to access a computer. These accounts must be audited. Configure accounts with least privilege to help prevent elevation of privilege. Remove any accounts that you do not need. Help to prevent brute force and dictionary attacks by using strong password policies, and then use auditing and alerts to detect logon failures.
Auditing and Logging	Auditing is one of your most important tools for identifying intruders, attacks in progress, and evidence of attacks that have occurred. Configure auditing for your server. Event and system logs also help you to troubleshoot security problems.
Files and Directories	Secure all files and directories with restricted permissions that only allow access to necessary services and accounts. Use auditing to allow you to detect when suspicious or unauthorized 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	Many security-related settings are stored in the registry. As a result, you must secure the registry. You can do this by applying restricted Windows access control lists (ACLs) and by blocking 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.

Application Security Categories

Patterns & practices security guidance includes application security categories for the appropriate application types. Some application types will require review of application security categories as well as server security categories. For example, you should review application-level Web.config file settings for ASP.NET applications. Use these categories as a starting point, and add new items as you learn more about deployment reviews.

Conclusion

Deployment reviews can help to ensure that application security is not compromised by poor configuration of the network or host. By using server security categories, you can conduct a systematic review that can be effectively repeated during the next deployment.

Additional Resources

For more information, see "patterns & practices Security Deployment Review Index" at *http://msdn.microsoft.com/library/en-us/dnpag2/html/SecurityDeploymentReviewIndex.asp*.