In a typical application development environment, architects and developers share similar experiences. They deploy business applications in a highly compressed time frame—making the applications work, testing the functionality at all levels, ensuring that they meet expected system performance or service levels, and wrapping the applications with an attractive client presentation and user documentation. Ensuring the security of the application at all levels has usually been considered at the last phase of the development process. If this is your company’s current application development process, then you are not alone.

End-to-end security should be adopted and accomplished as part of the early application design and development process. It should not be addressed at the end of the deployment phase or even considered just before the system testing in a pre-production environment. If you wait to consider security at either of these points, your options for reactive or post-mortem security fixes are very limited. And it is important to note the fact that there is no rollback for an application security breach.

In an enterprise application development life-cycle process, different architects may have different security architecture design perspectives for the same set of security requirements. Some assume that they can secure applications with infrastructure security protection (for example, firewall policy and proxy topology). Some would prefer to secure applications using a specific-vendor security framework and infrastructure solutions that are categorized as best-practice solutions for application security. Nevertheless, what was considered secure application design may appear to be insecure if someone discovers a loophole in the
application that the security architects have overlooked in the early design stage. It would be challenging to create a quality application security design that is repeatable yet reliable, so that architects could ensure all aspects of application security are considered during the early design stage in a structured manner. In addition to that, there are industry best-practices for applying security that need to be put in place before the application design process. It is always accepted as a good practice to proactively check and verify the security design for risks, trade-offs, security policies, proactive defensive strategies, and reality checks upon completion of the application design phase. After production deployment, it is also important to adopt reactive security measures and defensive strategies to ensure service continuity and recovery in case of a security breach or malicious attack. These help in identifying and thwarting security-related threats and vulnerabilities in all facets of the application development life-cycle process, from use-cases to components, from components to prototypes, from prototypes to final implementation, from implementation to production deployment, and until retirement. With such a detailed verification process, architects and developers can reduce critical security risks within the software development life cycle and prior to production deployment. This mandates a security design methodology that provides a systematic approach and a well-defined process.

This chapter will discuss the prescription for a robust security architecture design, which is the alchemy of securing business applications end-to-end at all levels. In particular, it will cover the rationale for adopting a security methodology, the process steps of security methodology, and how to create and use security patterns within that methodology. It will also look at how and why to do a security assessment as well as adopting a security framework.

The Rationale

An application or service may consist of a single functional component or multiple sets of disparate components that reside locally or over a network. Security is often considered as a complex process, encompassing a chain of features and tasks related to computer system security, network security, application-level security, authentication services, data confidentiality, personal privacy issues, cryptography, and so forth. More importantly, these features must be designed and verified independently and then made to work together across the system. Applying a security feature often represents a unique function that can be a safeguard or a countermeasure, which guarantees the application or service by preventing or reducing the impact of a particular threat or vulnerability and the likelihood of its reoccurrence.
The Security Wheel

Security is represented as a set of features that fortifies the entire application or service with safeguards and countermeasures for potential risks and vulnerabilities. Each security feature is like a spoke in a wheel. This means that each functional component in the entire system must be secured or the wheel will not have structural integrity and may well break apart. In order to accomplish this, a methodical process must be put in place to ensure that security is addressed properly and integrated across all of these varying components. From the user who is accessing the application or service over the network to the routers and firewalls on the perimeter of the system and then up through the application or service and the OS on which it resides—a security design must identify the risks and address the safeguards and countermeasures of the system holistically. Incorporating fundamental security principles plays a vital role during the software design and architecture, and it also helps identifying and eliminating the risks and threats in the early phases of the software development cycle. The concept of a Security Wheel provides the basis for verifying the fundamental security principles mandated for securing an application or service.

Figure 8–1 illustrates the Security Wheel, which represents all of the fundamental principles of security.
Chapter 8  The Alchemy of Security Design

The Security Wheel is a logical representation of the fundamental security principles required for establishing Security by Default in an application or a service. It provides guidelines that need to be taken into consideration during the entire software development life-cycle, and it can be applied across all or selected components of an application or a service.

The Hub
At the core of the hub of the Security Wheel sits the service or application that you are building. In this representation, it refers more to the business logic than the application as a whole. The service resides in a secured server host with minimized and hardened OS. (OS Minimization refers to fewer software components on a server infrastructure, and Hardened OS refers to a reconfigured OS that applies security measures specified by the OS vendor and retains no non-essential programs, protocols, or services.) The secured host includes storage devices and accessories. Both the service and the target host environment must be configured and deployed through a secure configuration management and reliable provisioning mechanisms. The service makes use of a common identity management solution that provides repository and supporting mechanisms for verifying an entity and its associated credentials, for logging, and for reporting all activities.

The Spokes
The spokes represent the following 12 core security services applicable to an application or a service.

- **Authentication** provides the process of verifying and validating the evidence and eligibility of an entity to carry out a desired action.
- **Authorization** provides the process of verifying and validating the rights and privileges granted to the authenticated entity.
- **Confidentiality** provides mechanisms of protecting the information during transit or in storage from intentional or unintentional unauthorized entities.
- **Integrity** provides the mechanisms for maintaining the information tamper-proof and unmodified by unauthorized entities.
- **Policy** provides the rules and procedures that can provide access control directives or a regulatory function to all entities.
- **Auditing** provides a series of records of events about an application or service activity. These records are maintained to support forensic investigation. It also helps in determining regulatory compliance.
Management provides the mechanisms for centrally administering all security operations.

Availability provides mechanisms for ensuring reliability and timely access to the application or service and also its prolonged continuity in the event of a disaster or service interruption.

Compliance provides the assurance of a degree of constancy and accuracy by adherence to standards or regulatory requirements.

Logging provides the mechanisms for recording events that can provide diagnostic information in case of errors, problems, and unexpected behaviors. The recording of these events is usually not driven by business requirements and is generally short-term and transient in nature. Failure to log such events will usually not necessitate cancellation of a transaction.

PKI provides key management support for applying cryptographic mechanisms to protect data, transactions, and communication using a public-key infrastructure.

Labeling is a process of classifying information based on roles and responsibilities to prevent unauthorized disclosure and failure of confidentiality.

The above-mentioned security services are the guiding security principles for providing a robust security architecture. Applications or services can be reviewed with these security measures during their design phases or at appropriate phases prior to deployment.

The Wheel Edge

The wheel edge represents the perimeter security: the network security components such as routers, firewalls, packet-filtering appliances, intrusion detection systems (IDS), crypto accelerators, and other devices that sit between the Internet and your network. They make up the solution for protecting the network perimeter from connection attacks based on IP addresses, TCP ports, protocols, and packet filters.

Across the service and OS and all the way to the perimeter security, every security principle must be addressed as a service that contributes to the overall security architecture. In some cases, many of these security principles, represented as spokes in the wheel, are only applicable to a few components of the overall application or a service. Nevertheless, each component within the system must be examined to determine the associated risks and trade-offs. Adopting a structured security methodology helps to ensure that all security principles are addressed and captured during the software development life cycle or prior to production.
Secure UP

To get started, we must first identify a process to guide us through the software development life cycle so that we can meet the business and security goals we set forth. Adopting the Unified Process (UP) provides a comprehensive approach for ensuring that business requirements are defined, implemented, and tested within the software development life cycle. UP is an industry standard process with a proven track record. It defines the development disciplines, along with an iterative approach, for gathering, analyzing, implementing, and testing functional business requirements. For these reasons, we have chosen it to achieve our business requirements. What UP fails to address are how to incorporate the non-functional requirements of the system. These requirements are assumed but never adequately defined as part of the process.

Security is a non-functional requirement, in particular, that must be baked into the process right from the beginning of the inception phase. Too often, it is retrofitted into the application at the end of the construction phase, leading to vulnerabilities and performance and/or usability impacts. To avoid this situation, it is necessary to extend UP with a security discipline that will ensure that all of the security requirements of the application are defined, designed appropriately, implemented, and thoroughly tested. We will refer to the incorporation of these security disciplines into the Unified Process as **Secure UP**.

Secure UP establishes the prerequisites for incorporating the fundamental security principles. It also defines a streamlined security design process within a software development life cycle. UP introduces a security discipline with a set of new security activities. At first glance, the security disciplines seem to overlap heavily with the standard UP disciplines. Why do we need to split hairs over the difference between business requirements and security requirements, or between implementing a functional use case and a security use case? The answer is that universally, for each of these disciplines, there is a wide gap between the people who know and understand the business needs of the application and those who know and understand the security needs. Figure 8–2 depicts Secure UP and the integrated security disciplines.

The Secure UP–Security disciplines define the following activities:

- Security Requirements
- Security Architecture
- Security Implementation
- White Box Testing
- Black Box Testing
These activities coalesce to form the basis of a robust security infrastructure and deliver an end-to-end security solution for an application or service. The security discipline activities pertain to different phases of the software development cycle and do not include the sustaining functions in production, such as managing changes, updates, and patch management.

An overview of the activities in the security discipline is broken down as follows:

- **Security Requirements**: In this activity, one or more analysts will define the business-mandated security requirements of the system. This includes requirements based on industry regulations, corporate policies, and other business-specific needs. The analysts will be well-versed in regulatory compliance as well as corporate security policies.
Chapter 8 The Alchemy of Security Design

• **Security Architecture:** This activity focuses on the creation of an overall security architecture. Architects will take the mandated security requirements specified by the analysts and then create a draft of the candidate security architecture. This activity qualifies the architectural decisions through a well-defined risk analysis and trade-off analysis processes in order to identify the risks and trade-offs and how to mitigate them. This candidate architecture will also identify a set of security patterns that covers all of the security requirements within the component architecture and will detail them in a high-level way, addressing the known risks, exposures, and vulnerabilities. The candidate architecture will then be prototyped and refined before the final security design activity is begun. This activity will also address the combination of security design with the other non-functional requirements to ensure that the security implementation does not compromise other functional or quality-of-service requirements.

• **Security Design:** The Security Design activity takes the security architecture and refines it using approaches such as factor analysis, tier analysis, security policy design, threat profiling, trust modeling, information classification, and labeling. A senior security developer will create and document the design based on the candidate security architecture, analysis results, and taking into account the best practices and pitfalls regarding the strategies of each of the patterns.

• **Security Implementation:** In this activity, security-aware developers will implement the security design. A good security design will decouple security components from the business components and therefore not require the security developers to have strong interaction or integration with business developers. The security developers will implement the security patterns by using the strategies defined in the security design and incorporating the best practices for securing the code.

• **White Box Testing:** The White Box Testing activity is for white box, or full knowledge, security testing of the code. In this activity, security testers will review the code and look for security holes or flaws that can be exploited. They will test a variety of security attacks aimed at compromising the system or demonstrating how the security requirements can be bypassed.

• **Black Box Testing:** This activity is for black box, or zero knowledge, security testing of the system. During this activity, security testers will attempt to break into the system without any knowledge of the code or its potential weaknesses. They will use a variety of tools and approaches to hack the system. They will use “out-of-the-box” techniques to break into
the system by all possible means at the application level and end-user level. This will provide an overall assessment of the security of the system.

- **Monitoring**: The monitoring activity is an ongoing activity for the system while it is in deployment. In this activity, operations personnel will monitor the application and all security facets of it. This consists of a broad range of areas, starting at the perimeter with the routers and firewalls and extending all the way back to the application itself. Monitoring is an integral part of security and an ongoing activity.

- **Security Auditing**: In this activity, security auditors will come in and audit the system for security. They assure that the systems are in compliance with all industry, corporate, and business regulations and that proper audit trails are being maintained and archived properly. These audit trails may also be reviewed for suspicious activity that may indicate a possible security breach.

These activities take place at different points in the application life cycle and have dependencies on each other. Figure 8–3 shows the roles and activities representing the Secure UP security discipline activities.

In the above activity diagram, we see the high-level view of the security specific software development life-cycle activities divided by swimlanes representing the different roles. At the start of the application life cycle, analysts gather the mandated security requirements. Once the requirements gathering process is complete, an architect will create a conceptual model of the security architecture. The architect will refine the model further and then define a candidate architecture. He or she will identify appropriate patterns, risks, and trade-offs. He or she will also represent the relevant security principles and perform some conceptual prototyping to validate the architectural decisions. Based on the results of the prototyping, the applicable patterns and the overall architectural approach will then be transitioned to the designer.

The designer will take the high-level candidate architecture, decompose it, and create a security design that addresses all component-level requirements and analyses. The resulting security design is a refinement of the architecture based on other functional requirements, non-functional requirements, factor analysis, security policy design, tier analysis, trust modeling, and threat profiling. Once complete, the design is transitioned to the developers to implement.

The developers implement the design with an eye on code-level security. Each security requirement is implemented and verified through unit testing, and all code is unit tested before being turned over to the test team for overall system-level tests, including security tests to verify the systemic qualities of architecture.
Figure 8–3 Secure UP Activity diagram
The designer takes the completed code and performs a variety of system tests to ensure that the functional requirements are met as well as the non-functional requirements. This includes application-specific regression tests and reality checks. The designer is responsible for ensuring the adequacy of all systemic qualities contributing to the QoS and SLA agreements of the system as a whole. Upon verification of the system tests, the designer transitions the application to the testers for further security testing, such as penetration tests, operational security testing, application-level scanning, and probing tests. Probing tests include network mapping, vulnerability scanning, password cracking, file integrity checking, malicious code testing, and so on. Two sets of security testers test in parallel. The White Box testers test the system based on a review of the code and full knowledge of the architecture, design, and implementation of the system. They usually find the most security holes. Black box testers test the security of the application from the outside, with no knowledge of the inner workings. They usually find holes in the system as a whole, not particularly in the application alone. If any holes are found, they are transitioned back to the system designer for security analysis. From there, they may require modification to the design and then go back through that particular flow again. If no holes are found, or if they are labeled as acceptable risks, the application is transitioned to the operations staff.

Operations will then deploy the application into production. Once in production, operations will be responsible for monitoring the system for security activity. This includes all aspects of the system from the router to the database and from the hardware to the application. It also means constantly checking for and applying hardware and software patches to keep the system available and secure. Once deployed, the system will also be transitioned to the security auditors for auditing. Like monitoring, auditors will perform routine security audits of the application for the duration of its lifetime. Finally, all activity ceases when the application is retired and pulled out of production.

**Secure UP—Artifacts**

For each of the security disciplines in our Secure UP, there is a mandated set of artifacts. These artifacts represent the work product of the discipline and serve as milestones that allow for transition to the start of another discipline within the software development life cycle. The following is a list of artifacts by security discipline.

- **Security Requirements**: The artifacts from the Security Requirements phase define the security requirements specific to the business, management, and operational security of the applications. Some of those business
requirements are represented with organizational roles/rights, policies, and regulatory compliance requirements. These will be in document format with business-functional security requirements broken down and tracked by business-requirement identification number.

- **Create Security Use Cases**: The business security requirements are documented as a list of requirements with no real cohesion. To make sense of these requirements, they must be structured into developer-friendly use cases. These use cases will be in document format with business-functional security requirements broken down, combined into logical groups, and assigned use case numbers. The use cases then track back to any supporting business requirements or external policies that drove the use case.

- **Security Architecture**: The Security Architecture discipline has several artifacts. The first is a conceptual security model. This model represents the high-level security architecture that addresses the business security requirements defined in the use cases from the security requirements phase. The next artifact is the candidate security architecture. This architecture will be refined through the rest of the security activities in this phase, including risk analysis and trade-off analysis. Finally, a core set of security patterns will be chosen as part of the refined conceptual security model.

- **Security Design**: There are four significant artifacts of the Security Design discipline. The first is the policy design document. This document defines the policies for the application based on relevant industry, corporate, and business policies pertaining to the system. The second artifact is the trust model. This is created from factor and tier analyses of the policy design. The third artifact is the threat profile. This document defines the types of attacks and their associated risks based on the trust model, the policy design, and the refined security model. The last artifact of the Security Design discipline is the Security Design itself. This document or set of documents defines the detailed security design formulated from the union of all the artifacts. It will contain the patterns and other design material needed for implementation.

- **Security Implementation**: The four artifacts of Security Implementation are the source code, build/configuration, infrastructure, and the security unit tests. The source code in this case is the security-related code, such as the security patterns implementation and any security frameworks. The build artifacts are any security-related configurations (such as J2EE deployment descriptors) or documents for configuring security in third-
party products. The infrastructure artifacts specify the firewall rules, mini-
mization, hardening profiles, and so forth. The unit tests artifacts are those
tests that developers use to test that their code complies with the use cases
and provides the functionality specified in the design document.

- **White Box Testing**: This discipline has only one artifact, the test results
document. This document will specify the tests performed, and their results
identify the source code, configuration, and infrastructure failures/suc-
cesses as well as status, and severity.

- **Black Box Testing**: Black Box Testing has only one artifact as well, the
black box test results identifying the code flaws. This document will also
contain the tests run, tools used, and any techniques found to exploit the
application and its infrastructure weaknesses.

- **Environment Setup**: Environment Setup has several artifacts. To begin
with, the first artifact is to have all of the hardware and software installed
and configured. The next artifact is to have one or more Standard Operat-
ing Procedure (SOP) documents detailing how to install, configure, main-
tain, and troubleshoot the environment as well as how to manage crises in
the operations center. Another artifact will be completion of a change
management request (CMR) system for tracking and fulfilling change
requests. Also, the infrastructure layout and design is an important artifact.
This would consist of the network design (VLANs and DMZs), ACLs, and
trust zones. In some instances, honey pots may be implemented in order to
detect and observe and possibly capture intruders. System hardening, min-
imization, business continuity, and other system setup tasks may be treated
as artifacts individually or as a whole. These and other artifacts are better
described in a book focused on data center operations.

- **Patch Management**: The artifacts for Patch Management are similar to
those for Environment Setup. A patch management tool in conjunction
with the patch management procedures is the foremost artifact. This
allows operations staff to patch and track all of the various systems within
the data center. This is often a severely underestimated task. It is also the
source of many production outages—just ask anyone whoever tracked
down a random bug that turned out to be caused by a missing patch. Patch
management is an ongoing task and therefore the artifacts are evolutionary
in nature.
Chapter 8  The Alchemy of Security Design

- **Monitoring**: Service-level agreement (SLA) is usually associated with monitoring. It is represented as an ongoing task using a logging mechanism that captures all the security-specific alerts and issues. This artifact could be a periodic activity report for designating monitoring tools and procedures used in production as well as a method of support for forensic investigation.

- **Security Auditing**: Security Auditing delivers many artifacts associated with SLAs, including organizational policies, verifying compliance requirements, application/host/network configuration, and user activity. These artifacts will be outlined in the policy design. Like monitoring, security auditing is an ongoing process.

---

**Iterative Development**

One of the major tenets of the Unified Process is iterative development. The activities stated thus far resemble a waterfall approach in terms of how they are tied together in sequence. This is merely a by-product of the representation of the activity diagram tool and not intended to imply that the process is not iterative. It therefore must be stated clearly that the security disciplines are intended to fit into the overall iterative approach to development. Each use case will be addressed in an incremental and iterative manner. While the swim lanes in the activity diagram illustrate some parallelism, the exact breakdown of what can be done in parallel and to what extent tasks are performed iteratively will vary from application to application. It is beyond the scope of this book to discuss the intrinsics of iterative development, and therefore we will simply state that the security activities should be performed iteratively, the same as any other Unified Process activities.

---

**Risk Analysis (RA)**

RA is the process of describing threats—their impacts, possible consequences, and their probability and frequency of occurrence. RA also helps determine how to mitigate those identified risks by establishing the selection criteria for safeguards and countermeasures meant for preventing or reducing those risks to an acceptable level. The acceptable risks are termed as transferred risks that are manageable. Depending upon the security requirements, the RA process may include a range of activities such as risk identification, risk assessment, risk characterization, risk communication, risk mitigation, and risk-specific policy definition. RA
influences the security design process by helping the decision making process related to choosing applicable tools and mechanisms. This ensures that security measures are appropriate and fully commensurate with the risks to which the application or service is exposed.

A typical RA artifact will gather information based on the following techniques:

- **Asset Valuation**: This technique is the fundamental process of determining the value of an asset. An assessment of the overall business value of the application or service is made. Factors included are: initial and ongoing cost, insurance value and total estimated value, including the infrastructure and other intellectual properties. Asset valuation helps to justify the cost benefits in preventing or reducing the known risk and to satisfy the stakeholders, legal, and other regulatory requirements.

- **Quantitative Risk Analysis**: This technique identifies all key risk elements and estimates the value associated with each risk, such as infrastructure cost, potential threat, frequency, business impact, potential loss value, safeguard option, safeguard effectiveness, and safeguard value. Based on this information, it is possible to estimate the potential losses, analyze the potential threats, compute the Annual Loss Expectancy (ALE), and then identify countermeasures and safeguards. The formulas for computing the ALE are as follows:
  - Exposure Factor (EF) = Percentage of asset loss caused by the potential threat.
  - Single Loss Expectancy (SLE) = Asset value \times Exposure Factor
  - Annualized Rate of Occurrence (ARO) = Frequency of threat per year.
  - Annual Loss Expectancy (ALE) = SLE \times ARO

- **Qualitative Risk Analysis**: This technique identifies the threats and vulnerabilities specific to applicable scenarios identified through security reality checks. Based on the findings, it helps to mitigate the risks by identifying the appropriate safeguards and countermeasures to prevent them or to reduce the likelihood and effect of occurrence.

Table 8–1 represents an example template of a qualitative risk analysis artifact that identifies the known risks in aspects of *architecture tiers*, *possibility of occurrence* expressed in terms of number of prospects, *probability* expressed in terms of likelihood of possible occurrences, *Impact* expressed in terms of effect that affects the overall architecture, and *Exposure* expressed as the level of acceptability.
### Table 8-1 Sample RA Template for Qualitative Risk Analysis

<table>
<thead>
<tr>
<th>Known Risks</th>
<th>Tier/Component</th>
<th>Possibility Of Occurrence (Single/Multiple)</th>
<th>Probability</th>
<th>Impact</th>
<th>Exposure</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td></td>
<td></td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

**Issue:**

**Mitigation:**
Trade-Off Analysis (TOA)

The purpose of trade-off analysis is to improve the quality of security architecture with explicit, efficient, and rational decisions. TOA provides stakeholders with a systematic way of improving and validating the security architecture with its use of multiple security criteria, options, alternatives, and recommendations. In the security architecture discipline, it helps to weigh choices of security features against potential threats or vulnerabilities. This assists in justifying a financial case or identifying alternative options.

A typical TOA artifact contains all of the security architecture criteria and safeguard options and alternatives. It will also include an Effect Matrix, where the security options/alternatives are represented in columns and security criteria are represented in rows. The cells have two values: the top value indicates the magnitude of impact on a scale from –10 to 10, and the bottom value indicates the relative importance of that security criterion on a scale from +1 to +10. Table 8–2 illustrates an example Effect Matrix.

Thus, TOA is a ranking index for making architectural security decisions with clear assumptions and for addressing associated uncertainties.

Table 8–2  Effect Matrix Table (An Example)

<table>
<thead>
<tr>
<th>Safeguard Option 1</th>
<th>Safeguard Option 2</th>
<th>Safeguard Option 3</th>
<th>Safeguard Option 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Criterion A</td>
<td>+7</td>
<td>+6</td>
<td>+5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Security Criterion B</td>
<td>+2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Security Criterion C</td>
<td>–3</td>
<td>0</td>
<td>–2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
Chapter 8   The Alchemy of Security Design

Security Patterns

Good application design is often rooted in appropriate security design strategies and leverages proven best practices using design patterns. Design strategies determine which application security tactics or design patterns should be used for particular application security scenarios and constraints. Security patterns are an abstraction of business problems that address a variety of security requirements and provide a solution to the problem. They can be architectural patterns that depict how a security problem can be resolved architecturally (or conceptually), or they can be defensive design strategies upon which quality security protection code can later be built.

This section will note the existing security patterns available in the industry today and then introduce a new set of security patterns that are specific to J2EE-based applications, Web services, identity management, and service provisioning. These new security patterns will be further elaborated in the following chapters of this book.

Understanding Existing Security Patterns

There are a few known enterprise or information security patterns available on the Web. Most of them address generic information security issues related to the infrastructure of application security. Some of them are adapted from the Gang of Four [GoF] design patterns. They focus on security solutions dealing with the infrastructure or quality of services (for example, how to make the security service highly available), and are used as enterprise security design strategies. They do not delve into the feature and functional characteristics of using a security technology or how such technology can be incorporated to represent an end-to-end security model.

These known security patterns are summarized in the sections that follow. Table 8–3 through Table 8–6 outlines them.

Web Tier

Table 8–3 shows a list of known security patterns that support the Web Tier, which represents the components responsible for the presentation logic and delivery. The Web Tier accepts the client requests and handles access control to business service components. The security patterns shown in the table enable securing the client-to-server or server-to-server communication in the infrastructure as well as the application.
## Table 8–3 Existing Security Patterns Supporting the Web Tier

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Communication</td>
<td>HTTPS; SSL (TLS), IPsec</td>
<td>This pattern describes the use of a secure data transport layer for client-to-server and server-to-server communication. Reference: [YoderBarcalow1997], p. 24; [OpenGroup], p. 27</td>
<td>Protected System; Login Tunnel; Secure Access Layer</td>
</tr>
<tr>
<td>Secure Association</td>
<td>SSL (TLS); Cryptographic standards supported by JSSE, JCE, and JGSS</td>
<td>This pattern shows how to make secure interactions between two entities; for example, protecting the session between the browser and Web server using SSL or TLS, and secure e-mail using encryption and proxies. Secure Communication pattern is more specific to encrypting the communication channel between the client and server. Typically, HTTPS is a means of implementing the pattern. Secure Association pattern is broader, and covers any secure interaction, including the session between the browser and Web server via HTTPS. Reference: [OpenGroup], p. 32.</td>
<td>Secure Communication</td>
</tr>
</tbody>
</table>

(continues)
Chapter 8  The Alchemy of Security Design

Table 8–3  Existing Security Patterns Supporting the Web Tier (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Access Point</td>
<td>HTTPS; SSL (TLS)</td>
<td>This pattern enforces a single point of entry to the business services and applications that provides a login prompt or login page. It is usually implemented by forms-based authentication and Secure Socket Layer (SSL) with J2EE declarative security. Reference: [Berry], pp. 203-204; [YoderBarcalow1997], p. 4; [WassermannBetty], p. 18</td>
<td>Protected System</td>
</tr>
<tr>
<td>Check Point</td>
<td>JAAS</td>
<td>This pattern centralizes the authentication and authorization process logic to a “checkpoint” entity. It assumes using JAAS to implement the checkpointed system. Reference: [Berry], p. 204; [Monzillo]; [YoderBarcalow1997], p. 7; [OpenGroup], p. 47; [WassermannBetty], p. 27</td>
<td>Authentication Gateway; Self Registration; Checkpointed System</td>
</tr>
</tbody>
</table>

(continues)
Session

Secure applications need to track global information throughout the application life cycle. This pattern identifies session information (for example, HTTP session variables, RPC call information, service requester details in the JMS or the SOAP messages) that needs to be maintained for security tracking.

This pattern differs from the Singleton pattern in that the session information needs to be maintained and shared in a multi-threaded, multi-user, or distributed environment.

Reference:
[YoderBarcalow1997], p. 14; [Amos], p. 3

Security Provider

This pattern describes what a client should operate to perform authentication against the identity service provider for authentication or authorization assertion. It is part of the single sign-on process for enterprise identity management.

Reference:
[Romanosky2002], p. 11

Table 8–3 Existing Security Patterns Supporting the Web Tier (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td></td>
<td>Secure applications need to track global information throughout the application life cycle. This pattern identifies session information (for example, HTTP session variables, RPC call information, service requester details in the JMS or the SOAP messages) that needs to be maintained for security tracking. This pattern differs from the Singleton pattern in that the session information needs to be maintained and shared in a multi-threaded, multi-user, or distributed environment. Reference: [YoderBarcalow1997], p. 14; [Amos], p. 3</td>
<td>Authenticated Session; User’s Environment; Namespace; Threaded-based Singleton; Localized Globals</td>
</tr>
<tr>
<td>Security Provider</td>
<td></td>
<td>This pattern describes what a client should operate to perform authentication against the identity service provider for authentication or authorization assertion. It is part of the single sign-on process for enterprise identity management. Reference: [Romanosky2002], p. 11</td>
<td>Authoritative Source of Data; Enterprise Access Management; Enterprise Identity Management</td>
</tr>
</tbody>
</table>
Chapter 8  The Alchemy of Security Design

Business Tier

Table 8–4 shows a list of known security patterns that support the security services in the Business Tier. The Business Tier represents the business data and business logic.

Table 8–4  Existing Security Patterns Supporting the Business Tier

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role</td>
<td>J2EE declarative security features</td>
<td>This pattern shows the dissociation of a specific user from their privileges by using roles. J2EE declarative security allows such role-based access control to be defined and managed in the ejb-jar.xml and Web.xml deployment descriptors. Reference: [Berry], p. 205; [Monzillo]; [YoderBarcalow1997], p. 11</td>
<td>Class-scoped Authorization</td>
</tr>
<tr>
<td>Subject Descriptor</td>
<td>J2EE Security Access Controller</td>
<td>This pattern allows access to the security attributes of a subject or principal via the operations. It corresponds to the javax.security.auth.Subject and java.security.Principal classes in JAAS. This pattern can be used to check rights or credentials. The Full View with Errors and Limited View patterns refer to the access rights of the application functionality, not to the subject or principal. Reference: [OpenGroup], p. 22</td>
<td>PEP</td>
</tr>
</tbody>
</table>

(continues)
### Table 8–4 Existing Security Patterns Supporting the Business Tier *(continued)*

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Context</td>
<td>J2EE Security Access</td>
<td>This pattern provides a container for access to security attributes, such as effective user ID and group ID. In the context of J2EE technology, this pattern refers to the class AccessControlContext that provides a check permission API. Reference: [OpenGroup], p. 40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full View with Errors</td>
<td></td>
<td>This pattern provides a full view to users with errors incurred, including exceptions when necessary. Reference: [YoderBarcalow1997], p. 17</td>
<td></td>
</tr>
<tr>
<td>Limited View</td>
<td></td>
<td>This pattern allows users to see what they can access. Reference: [YoderBarcalow1997], p. 19; [Amos], p. 4</td>
<td>Client Input Filter</td>
</tr>
</tbody>
</table>

*(continues)*
Chapter 8  The Alchemy of Security Design

Table 8–4  Existing Security Patterns Supporting the Business Tier (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Event Logging</td>
<td>JMX; Java API for Logging</td>
<td>This pattern is related to the capture and tracking of security-related events for logging and audit trail. Logged information can be used for risk assessment or analysis. A variant of this pattern is the Risk Analysis pattern, which relates the overall security risk to the sum of security threat, the cost of protecting the resources or losing the resources, and the vulnerability. Once the overall security risk is determined, then the priority will be allocated to protect resources appropriately. Reference: [Romanosky2001], p. 8; [Romanosky2002], p. 4; [Amos], p. 4; [Berry], p. 205</td>
<td>Risk Assessment and Management; Risk Analysis</td>
</tr>
</tbody>
</table>

Integration Tier

Table 8–5 shows a list of security patterns that facilitate integration with external data sources.

Infrastructure and Quality of Services

Table 8–6 shows a list of security patterns that describe enabling infrastructure capabilities and QoS requirements, such as availability, reliability, and scalability.

As you may be aware by now, the focus and scope of existing security patterns are mostly limited to the infrastructure level, and they do not address the core application-specific security issues and their associated challenges. Nor do they attempt to adopt the core security services such as authentication, authorization, auditing, confidentiality, non-repudiation, and other requirements mandated by enterprise-scale applications and Web services.
Table 8–5  Existing Security Patterns Supporting the Integration Tier

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authoritative Source of Data</td>
<td></td>
<td>This pattern verifies the data source for authenticity and data integrity.</td>
<td>Reference:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reference: [Romanosky2001], p. 5; [Romanosky2002], p. 2; [Berry], p. 206</td>
<td></td>
</tr>
<tr>
<td>Third-Party Communication</td>
<td></td>
<td>This pattern helps identify the risks of the third-party relationship and applies relevant security protection measures for the third-party communication. Reference: [Romanosky2001], p. 10; [Romanosky2002], p. 6</td>
<td>Enterprise Partner Communication</td>
</tr>
</tbody>
</table>

Table 8–6  Existing Security Patterns for Infrastructure Quality of Services

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Balancing PEP</td>
<td></td>
<td>This pattern shows how to make horizontal scalable authentication components using load balancer and multiple instances of Policy Enforcement Points (PEPs). Reference: [OpenGroup], p. 18</td>
<td>Load Balancer; PEP; Subject Descriptor</td>
</tr>
</tbody>
</table>

(continues)
Clustered PEP This pattern makes highly available authentication components over clustered Web containers.
Reference: [OpenGroup], p. 46

Layered Security
This pattern configures multiple checkpoints.
Reference: [Romanosky2001], p. 7

Cold Standby
This pattern describes how to structure a security system or service to resume service after a system failure. The Cold Standby pattern typically consists of one active Recoverable Component and at least one standby Recoverable Component.
The Cold Standby pattern differs from the Clustered PEP pattern in that the latter primarily provides an authentication service as a Policy Enforcement Point, while the former may be any security service (including PEP).
Reference: [OpenGroup], p. 49

Disaster Recovery; Recoverable Component; Hot Standby; Cold Standby; Comparator-checked Fault Tolerant System

Table 8–6 Existing Security Patterns for Infrastructure Quality of Services (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustered PEP</td>
<td></td>
<td>This pattern makes highly available authentication components over clustered Web containers. Reference: [OpenGroup], p. 46</td>
<td>Recoverable Component; Hot Standby; Cold Standby; Comparator-checked Fault Tolerant System</td>
</tr>
<tr>
<td>Layered Security</td>
<td></td>
<td>This pattern configures multiple checkpoints. Reference: [Romanosky2001], p. 7</td>
<td>Check Point; Authentication Gateway; Self Registration; Checkpointed System</td>
</tr>
<tr>
<td>Cold Standby</td>
<td></td>
<td>This pattern describes how to structure a security system or service to resume service after a system failure. The Cold Standby pattern typically consists of one active Recoverable Component and at least one standby Recoverable Component. The Cold Standby pattern differs from the Clustered PEP pattern in that the latter primarily provides an authentication service as a Policy Enforcement Point, while the former may be any security service (including PEP). Reference: [OpenGroup], p. 49</td>
<td>Disaster Recovery; Recoverable Component; Hot Standby; Cold Standby; Comparator-checked Fault Tolerant System</td>
</tr>
</tbody>
</table>
Table 8–6  Existing Security Patterns for Infrastructure Quality of Services (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparator-checked Fault Tolerant System</td>
<td></td>
<td>This pattern structures a system that enables the detection of independent failure of any component. It requires a fault-detecting mechanism to be in place to report or detect any system fault for a security system, for example, polling the state of the security device periodically, or checking the heartbeat of the Secure Service Proxy, Secure Daemon, or similar intermediaries. Reference: [OpenGroup], p. 51</td>
<td>Tandem System</td>
</tr>
<tr>
<td>Journaled Component</td>
<td></td>
<td>This pattern specifies how to capture changes to a security component’s state for future system state recovery. Reference: [OpenGroup], p. 53</td>
<td></td>
</tr>
<tr>
<td>Hot Standby</td>
<td></td>
<td>This pattern describes how to structure a security system or service to provide highly available security services, or to protect system integrity from system failure. This is usually done by synchronizing state updates to the replica or back-up security components without temporary loss of security services in case of full or partial system failure. Reference: [OpenGroup], p. 55</td>
<td>Synchronized Distributed System; Replicated Transaction</td>
</tr>
</tbody>
</table>
Chapter 8  The Alchemy of Security Design

Security Patterns for J2EE, Web Services, Identity Management, and Service Provisioning

There are new security patterns specific to delivering end-to-end security in J2EE applications, Web services, identity management, and service provisioning. These security patterns differ from existing security design patterns in that they address the end-to-end security requirements of an application by mitigating security risks at the functional and deployment level, securing business objects and data across logical tiers, securing communications, and protecting the application from unauthorized internal and external threats and vulnerabilities.

A simple taxonomy by logical architecture tiers are made here: Web Tier, Business Tier, Web Services Tier, and Identity Tier. Ideally, these patterns and others like them will be maintained in a patterns catalog that will be consulted during the security architecture activity in order to feed patterns into the security design. Through many versions of the application and across applications, these patterns will continue to grow and their implementation will be refined.

These patterns are usually structured and represented using a standard pattern template that allows expressing a solution for solving a common or recurring problem. The template captures all the elements of a pattern and describes its motivation, issues, strategies, technology, applicable scenarios, solutions, and examples.

Security Pattern Template

To facilitate using the security patterns, we adopted a pattern template that consists of the following:

- **Problem**: Describes the security issues addressed by the pattern.
- **Forces**: Describes the motivations and constraints that affect the security problem. Highlights the reasons for choosing the pattern and provides justification.
- **Solution**: Describes the approach briefly and the associated mechanisms in detail.
- **Structure**: Describes the basic structure of the solution using UML sequence diagrams and details the participants.
- **Strategies**: Describes different ways a security pattern may be implemented and deployed.
- **Consequences**: Describes the results of using the security pattern as a safeguard and control measure. It also describes the trade-offs.
Security Patterns

- **Security Factors and Risks**: Describes the factors and risks to be considered while applying the pattern.
- **Reality Checks**: Describes a set of review items to identify the feasibility and practicality of the pattern.
- **Related Patterns**: Lists other related patterns from the Security Patterns Catalog or from other related sources.

In the following sections, we will present the security patterns catalog and discuss each pattern and its logical tier. We will use sample scenarios and describe how these security patterns relate to each other and together contribute to the end-to-end security of an application.

**Security Patterns Catalog**

In this section we introduce the security design patterns that facilitate securing J2EE-based applications, Web services, identity management, and service provisioning technologies. We will identify the patterns based on their logical tier representations, such as Web Tier, Business Tier, Web Services Tier, Identity Tier, and Service Provisioning.

**Web Tier Security Patterns**

Table 8–7 shows a list of security patterns that are available in the Web Tier.

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication Enforcer</td>
<td>HTTPS; SSL/TL; IPsec; JAAS; JSSE; JCE; JGSS;</td>
<td>This pattern shows how a browser client should authenticate with the server. It creates a base Action class to handle authentication of HTTP requests. Refer to Chapter 9, “Securing the Web Tier: Design Strategies and Best Practices,” for details.</td>
<td>Context Object [CJP]; Intercepting Filter [CJP]</td>
</tr>
</tbody>
</table>

(continues)
## Chapter 8 The Alchemy of Security Design

### Table 8–7 Web Tier Security Design Patterns (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization Enforcer</td>
<td>JACC JAAS; JSSE; JCE; JGSS;</td>
<td>This pattern creates a base Action class to handle authorization of HTTP requests. Refer to Chapter 9 for details.</td>
<td>Context Object; Intercepting Filter [CJP]</td>
</tr>
<tr>
<td>Interception Validator</td>
<td>JSP Servlets</td>
<td>This pattern refers to secure mechanisms for validating parameters before invoking a transaction. Unchecked parameters may lead to buffer overrun, arbitrary command execution, and SQL injection attacks. The validation of application-specific parameters includes validating business data and characteristics such as data type (string, integer), format, length, range, null-value handling, and verifying for character-set, locale, patterns, context, and legal values. Refer to Chapter 9 for details.</td>
<td>Message Inspector; Interceptor [POSA]</td>
</tr>
<tr>
<td>Pattern Name</td>
<td>Standards and Technologies</td>
<td>Description</td>
<td>Related Patterns</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Secure Base Action</td>
<td>JSP; Servlets; and helper classes</td>
<td>The secure base action is a pattern for centralizing and coordinating security-related tasks within the Presentation Tier. It serves as the primary entry point into the Presentation Tier and should be extended, or used by a Front Controller. It coordinates use of the Authentication Enforcer, Authorization Enforcer, Secure Session Manager, Intercepting Validator, and Secure Logger to ensure cohesive security architecture throughout the Web Tier. Refer to Chapter 9 for details.</td>
<td>FrontController [CJP]; Command[GoF]; Authentication Enforcer; Authorization Enforcer; Secure Logger; Intercepting Validator</td>
</tr>
<tr>
<td>Secure Logger</td>
<td>JMX; Java API for logging</td>
<td>This pattern defines how to capture the application-specific events and exceptions in a secure and reliable manner to support security auditing. It accommodates the different behavioral nature of HTTP servlets, EJBs, SOAP messages, and other middleware events. Refer to Chapter 9 for details.</td>
<td>Abstract Factory Pattern[GoF]; Secure Pipe;</td>
</tr>
</tbody>
</table>
Chapter 8  The Alchemy of Security Design

Table 8–7  Web Tier Security Design Patterns (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Pipe</td>
<td>HTTPS; SSL/TLS; IPsec</td>
<td>This pattern shows how to secure the connection between the client and the server, or between servers when connecting between trading partners. In a complex distributed application environment, there will be a mixture of security requirements and constraints between clients, servers, and any intermediaries. Standardizing the connection between external parties using the same platform and security protection mechanism may not be viable. It adds value by requiring mutual authentication and establishing confidentiality or non-repudiation between trading partners. This is particularly critical for B2B integration using Web services. Refer to Chapter 9 for details.</td>
<td>Message Interceptor Gateway</td>
</tr>
</tbody>
</table>
Secure Service Proxy

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Service Proxy</td>
<td>Servlets, JAX-RPC, SAAJ</td>
<td>This pattern is intended to secure and control access to J2EE components exposed as Web services endpoints. It acts as a security proxy by providing a common interface to the underlying service provider components (for example, session EJBs, servlets, and so forth) and restricting direct access to the actual Web services provider components. The Secure Service Proxy pattern can be implemented as a Servlet or RPC handler for basic authentication of Web services components that do not use message-level security. Refer to Chapter 9 for details.</td>
<td>Proxy [GoF]; Interceptor Web Agent; Secure Message Router; Message Interceptor Gateway; Extract Adapter [Kerievsky]</td>
</tr>
</tbody>
</table>

(continues)
### Table 8–7  Web Tier Security Design Patterns (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Session Manager</td>
<td>Servlets, EJB</td>
<td>This pattern defines how to create a secure session by capturing session information. Use this in conjunction with Secure Pipe. This pattern describes the actions required to build a secure session between the client and the server, or between the servers. It includes the creation of session information in the HTTP or stateful EJB sessions and how to protect the sensitive business transaction information during the session. The Session pattern is different from the Secure Session Manager pattern in that the former is generic for creating HTTP session information. The latter is much broader in scope and covers EJB sessions as well as server-to-server session information.</td>
<td>Context Object [CJP]</td>
</tr>
<tr>
<td>Intercepting Web Agent</td>
<td>Web server plug-in</td>
<td>This pattern helps protect Web applications through a Web Agent that intercepts requests at the Web Server and provides authentication, authorization, encryption, and auditing capabilities. Refer to Chapter 9 for details.</td>
<td>Proxy [GoF]</td>
</tr>
</tbody>
</table>
**Business Tier Security Patterns**

Table 8–8 shows a list of security patterns that are available in the Business Tier.

**Table 8–8  Business Tier Security Design Patterns**

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit Interceptor</td>
<td>Java API for Logging</td>
<td>The Secure Logger pattern provides instrumentation of the logging aspects in the front, and the Audit Interceptor pattern enables the administration and manages the logging and audit in the back-end. Refer to Chapter 10, “Securing the Business Tier–Design Strategies and Best Practices,” for details.</td>
<td>Secure Logger Intercepting Filter [CJP]</td>
</tr>
<tr>
<td>Container Managed Security</td>
<td>EJB</td>
<td>This pattern describes how to declare security-related information for EJBs in a deployment descriptor. Refer to Chapter 10 for details.</td>
<td>Secure Pipe</td>
</tr>
<tr>
<td>Dynamic Service Management</td>
<td>JMX</td>
<td>This pattern provides dynamically adjustable instrumentation of security components for monitoring and active management of business objects. Refer to Chapter 10 for details.</td>
<td>Secure Pipe; Secure Message Router</td>
</tr>
</tbody>
</table>

(continues)
### Table 8–8  Business Tier Security Design Patterns (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obfuscated Transfer Object</td>
<td>JCE</td>
<td>This pattern describes ways of protecting business data represented in transfer objects and passed within and between logical tiers. Refer to Chapter 10 for details.</td>
<td>Transfer Object [CJP];</td>
</tr>
<tr>
<td>Policy Delegate</td>
<td>JACC EJB XACML</td>
<td>This pattern creates, manages, and administers security management policies governing how EJB tier objects are accessed and routed. Refer to Chapter 10 for details.</td>
<td>Secure Base Action; Business Delegate [CJP]</td>
</tr>
<tr>
<td>Secure Service Façade</td>
<td>EJB</td>
<td>This pattern provides a session façade that can contain and centralize complex interactions between business components under a secure session. It provides dynamic and declarative security to back-end business objects in the service façade. It shields off foreign entities from performing illegal or unauthorized service invocation directly under a secure session. Session information can be also captured and tracked in conjunction with the Secure Logger pattern. Refer to Chapter 10 for details.</td>
<td>Secure Service Proxy; Session Façade [CJP]</td>
</tr>
</tbody>
</table>

(continues)
Table 8–8  Business Tier Security Design Patterns (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Session Object</td>
<td>EJB</td>
<td>This pattern defines ways to secure session information in EJBs facilitating distributed access and seamless propagation of security context. Refer to Chapter 10 for details.</td>
<td>Transfer Object [CJP]; Session Façade[CJP]</td>
</tr>
</tbody>
</table>

Web Services Tier Security Patterns

Table 8–9 shows a list of security patterns that are available in the Web Services Tier.

Table 8–9  Web Services Tier Security Design Patterns

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Inspector</td>
<td>XML Encryption; XML Signature; SAAJ; JAX-RPC; WS-Security; SAML; XKMS;</td>
<td>This pattern checks for and verifies the quality of XML message-level security mechanisms, such as XML Signature and XML Encryption in conjunction with a security token. The Message Inspector pattern also helps in verifying and validating applied security mechanisms in a SOAP message when processed by multiple intermediaries (actors). It supports a variety of signature formats and encryption technologies used by these intermediaries. Refer to Chapter 11, “Securing Web Services–Design Strategies and Best Practices,” for details.</td>
<td>Message Interceptor Gateway, Secure Message Router</td>
</tr>
</tbody>
</table>
# Chapter 8 The Alchemy of Security Design

## Table 8–9 Web Services Tier Security Design Patterns (continued)

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Interceptor Gateway</td>
<td>JAX-RPC; SAAJ; WS-Security XML Signature; XML Encryption; SAML XACML WS-*</td>
<td>This pattern provides a single entry point and allows centralization of security enforcement for incoming and outgoing messages. The security tasks include creating, modifying, and administering security policies for sending and receiving SOAP messages. It helps to apply transport-level and message-level security mechanisms required for securely communicating with a Web services endpoint. Refer to Chapter 11 for details.</td>
<td>Secure Access Point, Message Inspector, Secure Message Router</td>
</tr>
<tr>
<td>Secure Message Router</td>
<td>WSS-SMS XML Signature XML Encryption WS-Security Liberty Alliance SAML XKMS</td>
<td>This pattern facilitates secure XML communication with multiple partner endpoints that adopt message-level security and identity-federation mechanisms. It acts as a security intermediary component that applies message-level security mechanisms to deliver messages to multiple recipients where the intended recipient would be able to access only the required portion of the message and remaining message fragments are made confidential. Refer to Chapter 11 for details.</td>
<td>Secure Access Point, Message Inspector, Message Interceptor Gateway</td>
</tr>
</tbody>
</table>
Security Patterns for Identity Management and Service Provisioning

Table 8–10 shows a list of security patterns available for the Identity Tier and Secure Service Provisioning.

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Standards and Technologies</th>
<th>Description</th>
<th>Related Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion Builder</td>
<td>SAML; Liberty Alliance</td>
<td>This pattern defines how an identity assertion (for example, authentication assertion or authorization assertion) can be built. Refer to Chapter 12, “Securing the Identity–Design Strategies and Best Practices,” for details.</td>
<td></td>
</tr>
<tr>
<td>Credential Tokenizer</td>
<td>SAML; Liberty Alliance</td>
<td>This pattern describes how a principal’s security token can be encapsulated, embedded in a SOAP message, routed, and processed. Refer to Chapter 12 for details.</td>
<td></td>
</tr>
<tr>
<td>Single Sign-on (SSO) Delegator</td>
<td>SAML; Liberty Alliance</td>
<td>This pattern describes how to construct a delegator agent for handling a legacy system for single sign-on (SSO). Refer to Chapter 12 for details.</td>
<td></td>
</tr>
<tr>
<td>Password Synchronizer</td>
<td>SPML</td>
<td>This pattern describes how to securely synchronize principals across multiple applications using service provisioning. Refer to Chapter 13, “Secure Service Provisioning–Design Strategies and Best Practices,” for details.</td>
<td></td>
</tr>
</tbody>
</table>
Security Patterns and their Relationships

Security patterns can seem very complex before we know the role and context of how they are related to each other, how they are relevant to the scenario, and how to apply them end-to-end in a typical application design process. Figure 8–4 depicts how all the security patterns just presented work together in the Web Tier (interacting with the clients), the Business Tier (encapsulating business logic and related processes), the Web Services Tier (integrating with internal or external application infrastructure), and the Identity Tier (for signing-on the authenticated identity with identity infrastructure providers).

Applying Security Patterns

Let’s consider a Web-based business portal as an example. The portal hosts business services from multiple business partner resources and provides member rewards redemption services. In a typical scenario, a subscriber logs in to the member rewards provider portal to check his membership award balance and submits a request to an affiliate content provider (a trading partner of the service provider) to redeem points and obtain a gift.

Web Tier

The subscriber uses a Web browser to sign on to the rewards portal. The portal initiates a secure communication channel between the client browser and the Web server using the Secure Pipe pattern. The Secure Pipe establishes the transport-layer security between the client and server using secure handshake protocols (such as SSL or TLS), which provide an encrypted data exchange and digital signatures for guaranteed message integrity.

Once the secure communication channel is established, the Front Controller pattern is used to process application requests (refer to http://java.sun.com/blueprints/patterns/FrontController.html for details). The Front Controller uses a Secure Base Action pattern that attempts to validate the session. Finding that the session information does not exist, the Secure Base Action uses the Authentication Enforcer pattern to authenticate the subscriber. The Authentication Enforcer prompts the subscriber for his user credentials. Upon successful authentication of the user credentials by the Authentication Enforcer, the Secure Base Action pattern uses the Secure Session Manager pattern to create a secure session for that user. It then applies the Authorization Enforcer pattern to perform access control on the request. Based on the user credentials and the relevant user provisioning information, it creates a secure session to access the required membership functions. During this process, the application uses the Secure Logger pattern to make
Figure 8–4 Security patterns and their relationships

The Alchemy of Security Design
use of the application logging infrastructure and initiates logging of all the user requests and responses by recording the sensitive business information and transactions, including success or failure attempts.

Figure 8–5 depicts the scenario with a sequence diagram showing the participants in the Web Tier.

In Figure 8–5, the actors denote the security patterns used. The Service Requester (or client) sends a request to initiate business services in the online portal. The Secure Pipe secures the service request in the transport layer. The Secure Base Action validates the session and uses the Authentication Enforcer to authenticate the session. The Authentication Enforcer will in turn request user credentials from the service requester and log the authentication result in the Secure Logger. Upon successful authentication, the Secure Base Action will create a secure session under the Secure Session Manager. It will also use the Authorization Enforcer to authorize the request and use the Intercepting Validator to validate the parameters in the request. Upon successful authorization processing, the Secure Base Action will log the events using the Secure Logger.

**Business Tier**

Under the secure session, a Container Managed Security pattern may be used to delegate authentication and authorization handling of the requests to the application server container. Policy can then be applied declaratively, in an XML deployment descriptor, or programmatically, using the container’s J2EE security APIs.

In our example, the business portal architects and designers require a more dynamic policy framework for the Business Tier and choose not to use container managed security due to the relative static nature of the deployment descriptors. Instead they use a combination of Business Tier patterns to provide security in the back-end business portal services.

Once the request has been processed in the Web Tier, the application invokes the relevant Business Tier services. These services are fronted using the Secure Service Façade pattern. This pattern can be augmented by the Container Managed Security pattern and is used for authenticating, authorizing, and auditing requests from the Web Tier.

Anticipating a large user volume through the business portal, its Web Tier and Business Tier are placed on separate machines (horizontally scaling) in order to enable high-scalability. Since the Business Tier lives in a different application server instance, authentication and authorization must be enforced on the Business Tier via security context propagation. This may seem redundant, but were it not done this way, there would be a significant security risk.
Figure 8–5  Web Tier security patterns sequence diagram
Chapter 8  The Alchemy of Security Design

The Secure Service Façade represents the functional interface to the back-end application services. This may include a service to inquire about the membership award balance and the submission of the reward redemption request. These may be business functions to which the subscriber is not entitled. The Secure Service Façade will use the Policy Delegate pattern to determine and govern the business-related security policies for the services to which the requester is entitled. When a request is first made to the Secure Service Façade, it will use the Dynamic Service Management pattern to load and manage the Policy Delegate class and any security-related supporting classes. The Dynamic Service Management pattern allows the application to maintain up-to-date policy capabilities by providing the ability to dynamically load new classes at runtime. In addition, it provides JMX management interfaces to the Policy Delegate for management and monitoring of policy operations.

Once the Policy Delegate is loaded, it can provide authentication and authorization of requests. When the customer requests their rewards balance, the Policy Delegate authenticates and then authorizes the request. It then uses the Secure Session Object pattern to create a session object such as an SSO (Single Sign-on) token that can then be used in subsequent service calls or requests to verify the identity of the requester.

The Secure Service Façade provides business and security auditing capabilities by using the Audit Interceptor pattern. Upon invocation, it notifies the Audit Interceptor of the requesting service. The Audit Interceptor then determines if, when, and how to log the request. Different types of requests may be logged in different locations or through different mechanisms. For the membership award balance service, the Audit Interceptor disregards the balance inquiries and generates an audit entry message that gets logged each time a redemption request is made.

Since confidential material is passed via the Secure Service Façade and the back-end services, it is necessary to provide a means for securing data, such as account numbers, balances, and credit card information, which must be prevented from disclosure in log files and audit entries. The Secure Service Façade uses the Obfuscated Transfer Object pattern to obscure business data from potential unauthorized interception and intentional or unintentional access without authorization. In this case, our customer’s credit card number, account number, and balance amount are obfuscated so that they will not show up in any logs or audit entries.

Figure 8–6 depicts a sequence diagram with some details about the scenario in the Business Tier.

In Figure 8–6, the actors denote the security patterns used. Typically, once the service request is processed by the Web Tier security patterns, a Business Delegate pattern (refer to http://java.sun.com/blueprints/corej2eepatterns/Patterns/BusinessDelegate.html for details) will be used to invoke Business Tier objects.
Figure 8–6
Business Tier security patterns sequence diagram

1: Create secure session
2: Notify AuditInterceptor
3: Load Dynamic Service Management class
4: Create PolicyDelegate
5: Notify PolicyDelegate
6: Process request
7: Create Secure Session Object
8: Create Obfuscated and Transfer Object
9: Create Obfuscated and Transfer Object
10: Notify application events
11: InVOKE business class
Chapter 8  The Alchemy of Security Design

The Business Delegate will create a service session using the Secure Service Façade (either local or remote synchronous invocation). The Secure Service Façade will instruct the Audit Interceptor to initiate the auditing process either synchronously or asynchronously. It will also load the Dynamic Service Management pattern for forceful instrumentation of management and monitoring process for business components. The Dynamic Service Management creates an instance of the Policy Delegate. The Secure Service Façade will start processing the request and invoke the Policy Delegate functions to process the request with the relevant policies defined for the objects or the service requester. It creates an instance of a Secure Session Object for the online portal transactions. The Secure Service Façade will invoke the business object to process business data in the service request. This may involve accessing business information related to the membership reward balance or requesting reward redemption services, in our sample scenario. To protect the business data in the transfer object, the business object can create instances of Obfuscated Transfer Object for delivery. Upon completion of the service request, the Secure Service Façade instructs the Audit Interceptor to capture and verify application-related events.

Web Services Tier

To communicate with the content providers, the service provider portal acts as a Web Services requester using a SOAP/XML-based Web services backbone to send membership award catalog/redemption requests to service providers hosted via Web services. The Web services–based service provider intercepts the service request from the member portal using the Message Interceptor Gateway pattern. The SOAP service request (using RPC-style messaging or the request-reply model) is verified and validated for message-level security credentials and other information by applying the Message Inspector pattern. Then the underlying services apply the Secure Message Router pattern that securely routes the message to the appropriate service provider or recipients. Upon successful message verification and validation using the Message Inspector pattern, the response message will be routed back to the intending client application. If asynchronous messaging intermediaries (using document-style messaging) initiate the SOAP messages, the Message Interceptor Gateway pattern at each intermediary will process these SOAP messages and apply similar techniques. This process may also involve forwarding the request to an identity provider infrastructure for verification of the authenticity of the credentials.

Figure 8–7 depicts a sequence diagram with some details about the scenario for Web services.
Figure 8–7 Web Services Tier security patterns sequence diagram
Chapter 8  The Alchemy of Security Design

In Figure 8–7, the actors denote the security patterns used. The Service Requester sends a request to invoke business services for the membership award catalog or redemption requests with other content providers. The Secure Pipe secures the service request. The Message Interceptor Gateway intercepts the SOAP message and uses the Message Inspector to verify and validate the security elements in the SOAP message. The Message Inspector confirms with the Message Interceptor Gateway that the message is validated (or not validated). Upon successful validation, the Message Interceptor Gateway forwards the SOAP message to the Secure Message Router, which will send the SOAP message to the final service endpoints provided by the Service Provider. The Service Provider will process the service request and return the result of the membership award request to the Service Requester via the Secure Pipe.

Identity Tier

The member portal currently has the capability to allow subscribers to sign on with other underlying services, including the business applications hosted by the service provider or the remote business services provided by the content provider (trading partners). To establish identity and grant access to users to other business services to which they are entitled, the portal uses protocols based on SAML and Liberty Alliance specifications.

Using the Assertion Builder pattern, the application creates a SAML authentication assertion for each business service that the subscriber chooses to invoke from the Authorization Activator pattern. It then encapsulates the user credentials in the security token in the SAML assertion using the Credential Tokenizer pattern. Because the customer loyalty system runs on the legacy back-end systems, the SSO Delegator pattern can be applied to integrate with the legacy back-end EIS system to provide the single sign-on access. This also facilitates global logout capability.

Using the Password Synchronizer as a supporting infrastructure function, the application runs secure service provisioning to synchronize user accounts across service providers. It complements the single sign-on security functionality provided by the SSO Delegator pattern and the Assertion Builder pattern. The subscriber inquires about the account balance of his or her membership award.

Figure 8–8 depicts a sequence diagram with some details about the scenario in the Identity Tier.

In Figure 8–8, the actors denote the security patterns used. Before the subscriber can use different remote membership reward services, his or her user account and password need to be registered first, using the Password Synchronizer. Once the subscriber signs on to the Web portal, he or she initiates a single sign-on
Figure 8–8
Identity Tier security patterns sequence diagram

1: Provision user account and password
2: Initiate single sign-on session
3: Create SAML assertion statement
4: Request SAML assertion
5: Create SAML assertion statement
6: Create security token using user credentials
7: Digitally sign SAML assertion statement
8: Return SAML assertion statement
9: Send SAML assertion
10: Issue global logout request
11: Issue logout request
12: Complete global logout request
request using the Single Sign-on (SSO) Delegator to the Web portal services and the associated service providers. The SSO Delegator will initiate remote security services. In order to process requests from the subscriber under the single sign-on environment, the service provider requires a SAML assertion (for example, a SAML authentication assertion). The SSO Delegator will then create a SAML assertion statement using the Assertion Builder, which will use the Credential Tokenizer to digitally sign the SAML assertion statement with the user credentials. Upon completion, the Assertion Builder will return the SAML assertion statement to the SSO Delegator, which will forward the SAML assertion statement to the service provider. After the subscriber has finished the membership reward request, he decides to log out from the online portal. He issues a global logout request to the SSO Delegator, which will issue the logout request to the service providers. Upon completion, the SSO Delegator notifies the service requester of the global logout result.

Patterns-Driven Security Design

As we discussed in the above example, a security design methodology is essential to any security-conscious service or application. Part of that methodology is adopting a patterns-driven security design process for addressing security requirements throughout the software development life cycle.

This security design process starts in the security architecture phase and continues into the security design phase. Figure 8–9 presents a patterns-driven security design process.

In Figure 8–9, in the architecture phase, the architects identify potential security patterns that can be used to satisfy the application-specific security requirements and rationalize the mitigated risks and trade-offs. Based on those inputs, the security design process will be carried out. The security designers perform factor analysis, tier analysis, trust models, threat profiling. They then create security policies that realize and validate the security use cases, architecture, and the identified patterns. During the design process, if there exists a security pattern that corresponds to the security use case requirements and it is architecturally significant, it can be incorporated into the security design. If there is no security pattern available, a new design approach must be taken. It can then be considered for reuse. If it is found to be reused enough, it can be classified as a pattern for inclusion into the Security Pattern Catalog.

In the build and integration portions of the development life cycle, architects and designers apply the relevant security patterns to the application design that satisfy the security use cases. They choose to use their preferred security framework tools to implement the application using the security patterns.
Prior to the deployment process, testers evaluate the application to ensure no security requirements or risk areas were overlooked. If a gap is identified that requires a change to the design, architects can revisit the security patterns to see if any additional security patterns or protection mechanisms are necessary. White and black box testing is an essential security measure that must be performed prior to deploying an application.

In summary, the security architecture and design process can be broken down into the following steps:

1. Identify the required security features based on the functional and non-functional requirements and organizational policies (Security Use Cases).
2. Create a conceptual security model based on architecturally significant use cases (Candidate Architecture).
Chapter 8  The Alchemy of Security Design

3. Perform risk analysis and mitigate risks by applying security patterns (Risk Analysis).
4. Perform trade-off analysis to justify architectural decisions (Trade-off Analysis).
5. Identify the security factors for each component or service specific to the application or system infrastructure (Factor Analysis).
6. Review the factors that impact the security of applications or Web services elements under each logical architecture tier (Tier Analysis).
7. Define the object relationship for security protection and identify the associated trust models or security policies (Trust Model).
8. Identify any security risks or threats that are specific to the use case requirements (Threat Profiling).
10. Apply security patterns wherever appropriate. Sometimes, rearchitecting or reengineering may be required (Security Pattern Design, Security Pattern Catalog, Apply Security Pattern).
11. Prior to implementation, use Security Reality Checks to review and assess the security levels by logical architecture tiers (Security Reality Check).

By applying this design process within a structured methodology, architects should be able to complete a secure architecture design using security patterns and derive a secure application architecture addressing the known risks and vulnerabilities. They should also be able to maintain a customized security pattern catalog based on past design and implementation experience or known design patterns.

Security Design Processes

In this section, we will take a look at each of the design processes in detail as part of the security design.

Factor Analysis

The objective of end-to-end application security is to provide reliable and secure protection mechanisms in business applications that can support authentication, authorization, data integrity, data privacy (encryption), non-repudiation (digital signature), single sign-on (for better efficiency and cost-effective security administration), monitoring and audit control, and protection from various security
threats or attacks. The related security attacks can be malicious code attacks, Denial of Service (DoS)/Distributed DoS attacks, dictionary attack, replay attacks, session hijacking, buffer overflow attacks, unauthorized intrusion, content-level attacks, session hijacking, identity spoofing, identity theft, and so on.

In an end-to-end security perspective, the security design will vary by a number of application-, platform-, and environment-specific requirements and factors, including the following.

**Infrastructure**
- Target deployment platform (and the underlying technologies and implementation constraints)
- Number or type of access points or intermediaries
- Service provider infrastructure (centralized, decentralized, distributed, or peer-to-peer), and the associated constraints of connecting to the infrastructure (for example, the data transport security requirement)
- Network security requirements

**Web Tier**
- Authentication-specific requirements (for example, multifactor authentication mechanism)
- Client devices or platform used (for example, Java card, Biometrics)
- Key management strategy (for example, whether key pairs are generated by a Certificate Authority and how the key pairs are stored)
- Authorization-specific requirements based on the sensitivity of the access requests.

**Business Tier**
- Nature of the business transaction (for example, non-sensitive information has a lower security protection requirement than sensitive, high-value financial transactions).

**Web Services Tier**
- Service invocation methods (RPC-style, document-style, synchronous, or asynchronous communication).
- Service aggregation requirements (for example, whether business services need to be intercepted, filtered, or aggregated from multiple service providers).
Chapter 8  The Alchemy of Security Design

Identity Tier
- Identity management strategy (for example, how network identity is established, validated, and managed).
- Policy management strategy (for example, management policies for who can access the SOAP messages and whether the service requester can access the full or partial content).
- Legacy security integration constraints (for example, security credential propagation).
- Single sign-on and sign-out requirements.

Quality of Services
- Service-level requirements (for example, quality-of-services requirements for high availability, performance, and response time).

Relating the Factor Analysis to apply Security Patterns. The security factor analysis is a good practice to use to identify the important application-specific and environment-specific constraints of the target applications and the target clients in relation to the overall security requirements. This will also help with locating the appropriate security patterns that can be used to address the business problems.

For example, in a Web-services security design scenario, we address the application- and environment-specific security requirements and constraints by representing the following security patterns:

- Secure the transport layer (Secure Pipe pattern, Secure Message Router pattern).
- Validate the SOAP message for standards compliance, content-level threats, malicious payload, and attachments (Message Interceptor Gateway pattern, Message Inspector pattern).
- Validate the message at the element level and the requesting identity (Message Inspector pattern).
- Establish the Identity policies before making business requests (Assertion Builder pattern).
- Protect the exposed business services and resources by service masking (Secure Service Proxy pattern).
- Protect the service requests from untrusted hosts, XML DoS, Message replay, Message tampering (Message Interceptor Gateway pattern).
- Timestamping all service requests (Secure Message Router pattern).
- Log and audit all service requests and responses (Secure Logger and Audit Interceptor pattern).
- Route requesting to multiple service endpoints by applying message-level security and Liberty SSO mechanisms (Secure Message Router pattern).

**Applying to the Media and Devices.** The security factors will be different when applied to different media or client devices. Different media and client devices, ranging from a Web browser, Java card, J2ME phones, and a rich client to legacy systems, have different memory footprints. Some of them may have more memory capacity to store the key pairs, or some of them have less memory to perform required security checking.

For instance, Web browsers are able to store the certificates keys and provide a flexible way to download signed Java applets, establish client-certificate-based authentication, and use SSL communication. J2ME based mobile phones and client devices operate on a lesser memory footprint and lesser processing speed. It is harder to use encryption and signature mechanisms and to perform complex cryptographic processing with these phones and devices due to their memory capacity and environment constraints.

A possible security artifact for the factor analysis is to produce a summary of the security factors based on the application-specific, platform-specific security requirements and the technology constraints in the security requirements document. This can be a separate appendix or a dedicated section that highlights the key areas of security requirements. The factor analysis provides an important input to the security architecture document. From the factor analysis, security architects and developers can justify which security design patterns or security design decisions should be used.

**Tier Analysis**

Tier analysis refers to the analysis of the security protection mechanisms and design strategies based on the business applications residing in different logical architecture tiers. In particular, it identifies the intra-tier communication requirements and dependencies. For instance, architects can use the HTTPS protocol to secure the data transport for applications residing in the Web Tier, but the same security protection mechanism will not work for applications residing in the Business Tier or in the Integration Tier. Similarly, the security protection mechanisms for asynchronous Web services will not work for synchronous Web services due to the difference in the RPC-style service invocation and document-style messaging architecture. The security design strategies and patterns discussed in this book
Chapter 8  The Alchemy of Security Design

are grouped by tiers to reflect what security protection mechanisms are relevant for each logical architecture tier.

A possible security artifact for the tier analysis is to produce a Tier matrix of security features by architecture tiers and by application layers. This matrix identifies the key security capability and design elements and their relation to different architecture tiers and application layers. During the security review, security architects and developers can evaluate the appropriateness and reliability-availability-scalability of the security design based on the tier matrix.

Threat Profiling

Threat profiling denotes profiling of architecture and application configurations for potential security weaknesses. It helps to reveal the new or existing security loopholes and the weaknesses of an application or service. Thus, it enumerates the potential risks involved and how to protect the solutions built and deployed using them. This will involve defining and reinforcing security deployment and infrastructure management policies dealing with updating and implementing security mechanisms for the application security infrastructure on an ongoing basis. It can be applied to the newly designed application systems, existing applications, or legacy system environments.

A possible security artifact for threat profiling identifies and categorizes the types of threats, potential security vulnerabilities, or exposures that can attack the application systems. A use-case–driven data flow analysis can also be used to trace the potential risks. For example, a threat profile may identify and list the threats and vulnerabilities as follows:

- Actual or attempted unauthorized access
- Introduction of viruses, Trojan horses, and malicious code
- Actual or attempted unauthorized probing of content
- Denial of service attacks
- Arbitrary code execution
- Unauthorized alteration and deletion of data
- Unauthorized access
- Unauthorized disclosure
- Unauthorized privilege escalation

In addition, it would discuss the security considerations and risk management techniques for all the identified loopholes and flaws.
Trust Model

A trust model is the backbone of the security design. It provides mechanisms that establish a central authority of trust among the components of the security architecture and that verify the identity of participating user entities and their credentials, such as name, password, certificates, and so forth. In simpler terms, a trust-modeling process is defined as follows:

- A trust model identifies specific mechanisms meant for responding to a specific threat profile, where a threat profile is a set of threats or vulnerabilities identified through a set of security use cases.
- A trust model facilitates implicit or explicit validation of an entity’s identity or the characteristics necessary for a particular event or transaction.

A trust model may contain a variety of systems infrastructure, business application, and security products. From a security design perspective, a trust model allows test-driving the patterns used, imposing a unique set of constraints, and determining the type and level of threat profiling required. Significant effort must go into the analysis preceding creation of the trust model to ensure that the trust model can be implemented and sufficiently tested. A trust model must be constructed to match business-specific requirements, because no generic trust model can be assumed to apply to all business or security requirements and scenarios.

Let’s take a server-side SSL example in which we assume that an SSL session is initiated between a Web browser and a server. The Web browser determines the identity of the server by testing the credentials embedded in the SSL session by means of its underlying PKI. The testing of credentials proves a “one-way trust model” relationship; that is, the Web browser has some level of confidence that the server is who it claims to be. However, the server has no information for testing the Web browser. Essentially, the server is forced to trust the Web-browser–returned content after initiating its SSL session.

Two possible security artifacts from the trust model can be produced. First, the analysis of the trust model usually specifies the security requirements and system dependencies for authentication and authorization in the security requirements specification. This provides the basic design consideration for authentication and authorization and provides input to the definition of system use cases for authentication and authorization. Second, the trust model will identify the security risks associated with the trust relationship. These form an important component in the overall risk document. For an example of a trust model, refer to [Liberty1] and [XACML2].
Chapter 8  The Alchemy of Security Design

Policy Design

Security policies are a set of rules and practices that regulate how an application or service provides services to protect its resources. The security policies must be incorporated into the security design in order to define how information may be accessed, what pre-conditions for access must be met, and by whom access can be permitted. In a typical security design artifact, security policies are presented in the form of rules and conditions that use the words must, may, and should. These rules and conditions are enforced on the application or service during the design phase by a security authority by defining the rights and privileges with respect to accessing an application resource or conducting operations.

Security policies applied to an application or service can be categorized as the following six types:

- **Identity policies**: Define the rules and conditions for verifying and validating the requesting entity’s credentials. These include usage of username/passwords, digital certificates, smart cards, biometric samples, SAML assertions, and so forth. This policy is enforced during authentication, authorization, and re-verification requirements of an identity requesting access to an application.

- **Access control policies**: Define the rules and conditions applied to a requesting entity for accessing a resource or executing operations exposed by an application or service. The requesting entity can be a user, device, or another application resource. The access control policies are expressed as rights and privileges corresponding to the identity roles and responsibilities of the requesting entity.

- **Content-specific policies**: Define the rules and conditions for securing the content during communication or storage. This policy enforces the content-specific privacy and confidentiality requirements of an application or service.

- **Network and Infrastructure policies**: Define the rules and conditions for controlling the data flow and deployment of network and hosting infrastructure services for private or public access. This helps to protect the network and hosting infrastructure services from external threats and vulnerabilities.

- **Regulatory policies**: Define the rules and conditions an application or service must adhere to in order to meet compliance, regulation, and other legal requirements. These policies typically apply specifically to financial, health, and government institutions (for example, the SOX, GLBA, HIPAA, and Patriot Act).
• **Advisory and informative policies**: These rules and conditions are not mandated but they are strongly suggested with respect to an organization or to business rules. For example, these policies can be applied to inform an organizational management team about service agreements with external partners for accessing sensitive data and resources, or to establish business communication.

In addition, in some cases we need to design and apply target application environment and business-specific policies such as:

• User registration, revocation, and termination policy
• Role-based access control policy
• PKI management policy
• Service provider trust policy
• Data encryption and signature verification policy
• Service audit and traceability policy
• Password selection and maintenance policy
• Information classification and labeling policy
• DMZ Environment access policy
• Application administration policy
• Remote access policy
• Host and network administration policy
• Application failure notice policy
• Service continuity and recovery policy

The security policy artifacts must capture these policy requirements and define the roles and responsibilities of the stakeholders who are responsible for implementing and enforcing them. It is also important to incorporate updates based on the changes in the organization and the application environment.

**Classification**

Classification is a process of categorizing and designating data or processes according to an organization’s sensitivity to its loss or disclosure. In an application or service, not all data has the same value to the requesting entity or to the business. Some data, such as trade-secrets, legal information, strategic military information, and so on, may be more sensitive or valuable than other data in terms
Chapter 8  The Alchemy of Security Design

of making business decisions. Classification is primarily adopted in information-sensitive applications or services in order to prevent the unauthorized disclosure of information and the failure of confidentiality and integrity.

The classification of data or processes in an application or service is typically represented as classes with five levels ranging from the lowest level of sensitivity to the highest. The least sensitive level is 1, and the most sensitive is 5.

1. **Unclassified**: The data or process represented with this classification is neither sensitive nor classified. The information is meant for public release and the disclosure does not violate confidentiality.

2. **Sensitive But Unclassified (SBU)**: The data or process represented with this classification may contain sensitive information but the consequences of disclosure do not cause any damage. Public access to this data or processes must be prevented. For example: General health care information such as medications, disease status, etc.

3. **Confidential**: The data or process in this classification must be protected within the organization and also from external access. Any disclosure of this information could affect operations and cause significant losses. For example: Loss of customer credit card information from a business data center.

4. **Secret**: The data or process in this classification must be considered as secret and highly protected. Any unauthorized access may cause significant damage. For example: Loss of strategic military information.

5. **Top Secret**: The data or process in this classification must be considered as highest secret. Any unauthorized disclosure will cause grave damage. For example: A country’s national security information.

In a classified information system, all data has an owner and the owner is responsible for defining the sensitivity of the data depending on the organizational policies. If the owner is not sure about the sensitivity level, then the information must be classified as “3 - Confidential.” The owner is also responsible for security of the data as per the organization security policy pertaining to the classification and for defining who can access the data. Classification also depends on organizational requirements related to information confidentiality. Organizations must define their classification terms and definitions.

**Security Labeling**

Security labels represent the sensitivity level of data or processes. They denote the type of classification assigned. During runtime access, the labels are verified
and validated in accordance with an organization’s security policy. To adopt classification and labeling of data processes, it is necessary to choose a highly secure operating system (for example, Trusted Solaris Operating system) that offers labeling of data and processes based on discretionary and mandatory access-control policies throughout the operating system, including all users, files, directories, processes, services, and applications. The label, once assigned, cannot be changed other than by an owner or authorized person in the classification hierarchy with higher privileges.

Classification and labeling requirements must be identified during the design phase. Classification and labeling must be adopted when an application or service is required to manage highly sensitive data or processes and the business or organization dictates classification requirements for its information with higher confidentiality.

**Application Security Assessment Model**

Before architects and developers decide on and adopt any security design strategies or patterns, they usually perform an assessment of the application security architecture and any security mechanisms in use. Typically, external security consultants or specialized security architects review the overall security requirements and the current security design in use. Based on their assessment, they will recommend a list of suggested security mechanisms to meet their application security goals as short-term and long-term implementations.

The assessment checklist has five columns in total. The first two columns enumerate the security services and the security mechanisms that provide them. The next three are checkboxes denoting if the mechanism is suggested for the architecture (that is, recommended for adoption based on best practices), if it is implemented in the current design, and/or whether it is planned for implementation in the future.

Let’s consider an application architecture that delivers a Web-based business-to-consumer portal that integrates a variety of back-end applications. The application security architecture adopts a basic authentication using username and password for authenticating the user, authorizes the user as a customer or administrator to perform further operations, and captures all events and actions using a logging mechanism for accountability. The back-end applications running on heterogeneous platforms make use of a shared security context to provide single sign-on access and to participate in portal-initiated transactions.

Table 8–11 shows a simple assessment list for an example—a Web-based application architecture that has adopted a simple authentication using username
Chapter 8  The Alchemy of Security Design

and password mechanisms. The application is found to not be sufficient to address the security requirements because it is experiencing denial of service attacks using fake requests. After assessment, it is suggested to incorporate client-certificate–based mutual authentication to verify the originating source and to restrict forged requests from further processing. In terms of authorization mechanisms, the application architecture currently allows granting access based on user groups; the assessment suggests granting access based on roles, such as Web administrator, system administrator, and business manager.

Table 8–11  Simple Assessment Checklist for Application Security Architecture

<table>
<thead>
<tr>
<th>Security Service</th>
<th>Security Mechanism</th>
<th>Suggested</th>
<th>Current</th>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>Username and password</td>
<td>✔ ✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Client-certificate</td>
<td>✔ ✔ Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authorization &amp;</td>
<td>Group grants</td>
<td>✔ ✔ Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access Control</td>
<td>Role grants</td>
<td>✔</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Delegation</td>
<td>✔</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Access control lists</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Policy objects</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rules</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secret Key</td>
<td>RSA</td>
<td>1024-bit</td>
<td>✔</td>
<td>Yes</td>
</tr>
<tr>
<td>Public Key</td>
<td>RSA - MD5</td>
<td>1024-bit</td>
<td>✔</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RSA - SHA-1</td>
<td>1024-bit</td>
<td>✔</td>
<td>Yes</td>
</tr>
<tr>
<td>Accountability</td>
<td>Audit logs</td>
<td>✔ ✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Centralized audit log</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Encrypted checksums on log records</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Encrypted log records</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Digital signature (non-repudiation)</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reality Checks

In the design process, architects and developers may have chosen multiple security patterns to meet the application security requirements. They may need to make certain compromises in order to meet other application requirements, such as application performance throughput or cost constraints. However, any trade-off should not sacrifice the overall business security requirements without proper sign-off by the business owners.

Table 8–12 shows application-specific security reality checks intended for ensuring production-quality security protection measures prior to deploying the J2EE-based applications or Web services to production. It is not meant to be exhaustive, but can be very handy for self-assessment.

Table 8–12  Security Reality Checks

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>Are there any documented security policies for J2EE-based or Web-services–based applications?</td>
<td></td>
<td>A written security policy is a key ingredient in a well-formed security architecture. The security policy document should clearly define the application, its users, and environment-specific security policies for the security design, implementation, and deployment. It should also then document associated procedures for securing the applications or the underlying infrastructure until its retirement.</td>
</tr>
</tbody>
</table>

(continues)
### Chapter 8  The Alchemy of Security Design

Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>Does the existing security policy cover the depth of application security that is associated with the following?</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Minimizing and hardening of the target operating system that runs the target applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Securing the application servers that run applications developed in J2EE or Web-services technologies</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Securing the business logic components and data objects</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Securing the production environment and the data center infrastructure.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td>Does the security policy cover the organizational standards, procedures, and design processes for data encryption, cryptographic algorithms, and key management?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td>Does the security policy cover any escalation procedure to manage security threats in case of security intrusion to the J2EE-based applications or Web services?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td>Do you have a business continuity plan that includes the business continuity of the application infrastructure to protect the applications and the associated risks?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continues)
Policy Check Item

How is the Security Policy communicated throughout the organization?

Options are e-mail, bulletin board system, newsletter, training sessions, and so forth. Security functions should not be treated as only security personnel’s job. It should be considered as everyone’s responsibility.

Is senior management aware of and supportive of the security policy? This includes regulatory requirements such as SOX, FISMA, and so forth.

If not, security policy is doomed to be ignored. If management is not supportive of the current policy, but look to you to provide a secure architecture, chances are you will be blamed for everything bad down the road that occurs because of their reluctance to enforce security. It is important to make management aware of their responsibility in enforcing the policy.

Is there an application and data access-control security policy? Are access policy roles and groups clearly documented?

These are the policies for establishing roles and groups. A simple system, for example, will have users and administrators. A more complex system requires remote access management, personnel management, access to server administration, and so forth.

Table 8-12 Security Reality Checks (continued)
Chapter 8  The Alchemy of Security Design

Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>Are allowed and denied services and protocols documented? Is the network topology enabled with a firewall to protect the DMZ environment, including the hosts and applications?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td>Are the locations of physical management points to support policy accessed through routers, gateways, and bridges identified on network topology documentation and how often is documentation updated?</td>
<td></td>
<td>These stress the importance of topology documentation, which is the guide to identifying where possible breaches in data access security could occur.</td>
</tr>
<tr>
<td>Policy</td>
<td>Does control of data access conform to policy? Is there a privacy/encryption policy?</td>
<td></td>
<td>Encryption is typically required for services involving personal information, for example, online banking. Although efforts are usually made to only encrypt sensitive payloads in order to avoid the computing overhead that encryption incurs, it is often (especially for Web-based applications) the case that the encryption overhead is minimal compared to the communications overhead. SSL carries noticeable performance issues; to counter these overheads, it should make use of hardware acceleration with key storage.</td>
</tr>
</tbody>
</table>

(continues)
Policy | How is data privacy ensured? | Y/N | Online privacy is a very broad topic, but in our discussion we focus on the communication aspect, which is usually accomplished with encryption. It is important to pinpoint where the encryption is done, who is responsible for the product doing the encryption, and what technology/algorithm is being used. It is also important to know where the keys are stored and how they are managed. There are different methods for encrypting data between the client and a Web server, such as HTTP/SSL, which is transport-layer (or channel) encryption, and application-level encryption, which is encrypting data directly in the application. Data integrity can be accomplished through checksums or message digest algorithms. This is built into HTTPS, but at the application-level it must be implemented.

Policy | Are users aware of the need to ensure data privacy and follow information and data handling procedures. Are they aware of importance of using data sensitivity levels? | Y/N | (continues)
The Alchemy of Security Design

Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>Is there an authentication and authorization policy?</td>
<td></td>
<td>These policies are more currently referred to as trust management. How is the trust established for a principal so that he or she is given authentication credentials and permitted authorization for certain functions? How is the trust maintained and terminated? Who is given the ability to give out these privileges? This portion of the security policy should lay these out in a step-by-step fashion.</td>
</tr>
<tr>
<td>Policy</td>
<td>Does policy prohibit sharing of authentication credentials, shared secrets, and so forth?</td>
<td></td>
<td>As stated previously, many headaches and finger-pointing episodes can be avoided by assuring a one-to-one relationship between each individual user and his or her authentication information. All actions and events should be traceable back to a unique credential (except for public access).</td>
</tr>
<tr>
<td>Policy</td>
<td>Does control of authentication and authorization conform to policy?</td>
<td></td>
<td>Password changes should be enforced regularly if a secure form of authentication such as S/KEY, smart cards, or biometrics are not used. This minimizes unauthorized users from borrowing or stealing passwords to access the system.</td>
</tr>
</tbody>
</table>

(continues)
The Alchemy of Security Design

### Table 8–12 Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>Is there a change management policy?</td>
<td>N</td>
<td>There should be a process in place for change management on both a system and a data level.</td>
</tr>
<tr>
<td>Policy</td>
<td>Is access provided on a “need to know” or “need to have” basis?</td>
<td>N</td>
<td>If accessing different databases, authentication should be performed on a per-transaction level. Access Control Lists are one way to accomplish this task.</td>
</tr>
<tr>
<td>Policy</td>
<td>Is access to data controlled so that users can only change that which they should have access to?</td>
<td>N</td>
<td>In many systems, the middle tier accesses all the back-end databases as one user. A sufficiently savvy client with access to one database may try to access another database or someone else’s records. This is why access control should be performed on a per-transaction basis with sufficiently fine-grained control.</td>
</tr>
<tr>
<td>Policy</td>
<td>Do procedures for changing production systems and data exist?</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td>Are access controls sufficiently flexible for users to do their jobs without compromising data confidentiality and integrity?</td>
<td>N</td>
<td>This can be accomplished in the ACL by specifying different levels of access (that is, read, write, append, modify, and create).</td>
</tr>
<tr>
<td>Policy</td>
<td>Does change management control conform to policy?</td>
<td>N</td>
<td>There should be a process in place to ensure that change management activities conform to a policy. This can be accomplished through routine reviews of log files.</td>
</tr>
</tbody>
</table>
### Table 8–12 Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration</td>
<td>Do you have a secure protection mechanism and well-defined procedures for key management (storing the key pairs used for authentication or generating digital signatures)? How are the key pairs managed? Where are they stored? Who can create, update, and manage key pairs?</td>
<td></td>
<td>Segregation of security design and administration is one of the security control best practices.</td>
</tr>
<tr>
<td>Administration</td>
<td>Are the security design and administration control personnel separated?</td>
<td></td>
<td>There are new security threats discovered from time to time, especially for some operating systems and Web browsers.</td>
</tr>
<tr>
<td>Administration</td>
<td>Do you have a regular security patch management process that applies to J2EE application servers, back-end application resources, and Web browsers?</td>
<td></td>
<td>Authorization databases may be mirrored; how is this process protected if separate systems are at separate locations or co-located using primary/secondary servers? Are the activities managed via delegated administration?</td>
</tr>
<tr>
<td>Administration</td>
<td>How are security administration activities coordinated between locations?</td>
<td></td>
<td>Are repeated login failures detected and recorded? If repeated login failures are logged, there should be tools or mechanisms to monitor the logs and alert security personnel to possible hacking attempts.</td>
</tr>
</tbody>
</table>

(continues)
Administration Are security monitoring processes in use? Efficient monitoring requires the use of intrusion detection systems (IDS) and filtering software that flags potential problems in accordance with the security policy in place. It is important that the IDS and filtering mechanisms are not performed during the creation of the audit trail; when problems arise, the more detail available from the audit trail, the better.

Administration Are audit trails of authentication and authorization generated and reviewed? Audit trails should record as much as possible and be reviewed with a healthy dose of filtering. No one will catch any problem in the middle of 1000 pages of normal access logs. It is important that any changes to authorization of existing users/groups be scrutinized.

Administration What is the content of the audit trails? At a minimum, an audit trail record should contain activity identification, the time of the activity, user identification, requested transaction, and results. Audits of administration changes should ideally contain old and new data.

Table 8–12  Security Reality Checks (continued)
Chapter 8  The Alchemy of Security Design

Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration</td>
<td>Are audit trail mechanisms protected?</td>
<td></td>
<td>Most audit trail mechanisms, other than those that are directly related to a security product, are not protected from tampering. An attacker “covers his tracks” by removing incriminating entries from the audit trail and can foil detection. An authenticated audit trail can detect tampering; an audit trail to a write-only device (printer or CD-ROM) prevents tampering.</td>
</tr>
<tr>
<td>Quality of Services</td>
<td>Does your application security design support high availability of the security services, including the authentication of user credentials? In other words, have you included any design considerations to secure the application infrastructure, processes, and communications from loss or damage from disasters, accidents, or security attacks?</td>
<td></td>
<td>High availability of application security may include the use of hardware or software clustering of directory servers, Web Agents, Message Router Intermediaries, or any security service components.</td>
</tr>
<tr>
<td>Quality of Services</td>
<td>Do you have a recovery plan if your security components (such as Intercepting Web Agent and Secure Message Router Intermediary) are being compromised or fail to function? Do you have validation methods for verifying the integrity of those deployed components?</td>
<td></td>
<td>The recovery design of the security service component should be part of the design process. It should be represented in the infrastructure and/or application level.</td>
</tr>
</tbody>
</table>

(continues)
Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality of Services</td>
<td>Does your application security design include any plan to predetermine the tolerance of application security failure?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality of Services</td>
<td>Does your application design include a recovery of the security services and provide an alternative infrastructure for the recovery while restoring the security services?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality of Services</td>
<td>Do you have a checklist of Java objects, Web services, XML messages, or any application design elements for which you need to evaluate or identify security threats?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality of Services</td>
<td>Do you have any risk management plan or any recovery strategies for each type of security breach?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client Device Tier</td>
<td>Do you check for any suspicious “footprint” in the client devices (for example, PDA)? Hackers may leave a suspicious footprint in the client devices for future “replay” or “exploit.”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client Device Tier</td>
<td>Are the key pairs stored securely in the local client device? How is the security assured?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continues)
Chapter 8  The Alchemy of Security Design

Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation Tier</td>
<td>How does the J2EE-based application design support login authentication?</td>
<td></td>
<td>Clear text password is strongly not recommended. It is considered highly insecure because the password can be sniffed on the network, but it may be sufficient with adequate protection that guarantees there is no danger of interception; at any rate, architects need to weigh those risks. Encrypted password via Kerberos tickets or SSL mechanisms. One-time passwords using a token device (for example, SecureID). Certificates (used with SSL). Browsers using SSL normally support server authentication via certificates. Client authentication using passwords over an HTTP/SSL connection is often used, but using client-side certificates are highly recommended.</td>
</tr>
</tbody>
</table>
### Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tier</td>
<td>How is the login information carried throughout the session execution: cookies, URL rewriting, or use of a security token?</td>
<td></td>
<td>Cookies in the clear-text form can be a source of attack. They should be encrypted, hashed, and timestamped to avoid session hijacking. URL rewriting must be protected using SSL and URL encoding/decoding mechanisms. Security tokens are set within the confines of an established security protocol such as SSL.</td>
</tr>
<tr>
<td>Web Tier</td>
<td>Does the session require encryption?</td>
<td></td>
<td>This is a measure of the importance of J2EE-based applications being dealt with. A positive answer means the bar has been raised for other security mechanisms, such as authentication, session state, and so forth.</td>
</tr>
<tr>
<td>Web Tier</td>
<td>Where is user authentication information stored?</td>
<td></td>
<td>Typically, there is a user lookup mechanism (database, LDAP, and so forth), which also holds authorization information about what the user is allowed to do now that we are confident he is who he says he is. However, certificate-based authentication may rely strictly on the certificate signature to ascertain authentication as well as authorization privileges.</td>
</tr>
</tbody>
</table>

(continues)
### Chapter 8  The Alchemy of Security Design

#### Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tier</td>
<td>Are local resources accessed use signed JARs?</td>
<td>Y</td>
<td>A trusted applet or application (from a signed JAR) may perform actions outside of the normal Java sandbox, such as writing to the local machine’s hard drive. Take care that the client sanctions these actions and make sure there is obfuscation that hides the business logic to protect the middle-tier business abstraction.</td>
</tr>
<tr>
<td>Web Tier</td>
<td>How is authentication done? How is authorized access controlled and how is authentication and authorization administered?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Tier</td>
<td>Is encryption hardware or software in use?</td>
<td>Y</td>
<td>Hardware-based encryption is often more secure and has better performance. It provides a tamper-resistant solution. Software-based encryption is easier to install and change as necessary, but may be compromised if an attacker attains “root” access on the host machine.</td>
</tr>
<tr>
<td>Web Tier</td>
<td>How is encryption technology used?</td>
<td></td>
<td>It should be addressed in the application level, the network level, and at the host-environment level.</td>
</tr>
</tbody>
</table>

(continues)
### Table 8–12 Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tier</td>
<td>Does encryption technology use standard encryption algorithms widely recognized as being effective (for example, FIPS approved)?</td>
<td></td>
<td>There is a standard set of accepted encryption algorithms, many of which are in the public domain, but there are security products that use unproven encryption technology. Standard algorithms include TripleDES, RSA, Diffie-Hellman, IDEA, RC2, RC4, and Blowfish. The newsgroup sci.crypt regularly publishes a FAQ that identifies features to look out for when reviewing an encryption security product.</td>
</tr>
<tr>
<td>Web Tier</td>
<td>Are there U.S. export or international laws to be considered while using encryption?</td>
<td></td>
<td>U.S. federal law currently restricts export of products using encryption technology. For an intranet environment where U.S. businesses have a presence overseas, this is not an issue. For a U.S. company offering services to overseas clients, this is an issue.</td>
</tr>
</tbody>
</table>

(continues)
Key management involves making sure each member of a communication has the correct key value to either encrypt or decrypt a data stream.

Current encryption products involve the use of public-key technology, usually in the form of X.509 certificates. These are the certificates used by Web browsers from Netscape and Microsoft. The big problem today is finding out when a certificate has been revoked. A certificate always has an expiration date, but to date no standard method is in wide use for how to resolve premature certificate revocation; that is, revoking a fired employee’s certificate.

Table 8–12 Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tier</td>
<td>How is key management done?</td>
<td></td>
<td>Key management involves making sure each member of a communication has the correct key value to either encrypt or decrypt a data stream. Current encryption products involve the use of public-key technology, usually in the form of X.509 certificates. These are the certificates used by Web browsers from Netscape and Microsoft. The big problem today is finding out when a certificate has been revoked. A certificate always has an expiration date, but to date no standard method is in wide use for how to resolve premature certificate revocation; that is, revoking a fired employee’s certificate.</td>
</tr>
</tbody>
</table>

(continues)
### Web Tier

#### How are secret keys kept private?

Some options: token device (for example, smart card), or password-encrypted file. Use of a plaintext file to store a secret key is risky because the key is easily compromised if the machine is successfully attacked, but such a measure is necessary for machines that need the ability to cycle without human intervention (typing a password or inserting a smart card). If an LDAP database is used, encryption between the LDAP server and the authenticating party should be considered.

#### What other authentication mechanisms are in place (smart cards, biometrics)?

Multifactor authentication combining smart card and biometric authentication is often considered as a reliable personal authentication option.

---

#### Table 8–12 Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tier</td>
<td>How are secret keys kept private?</td>
<td></td>
<td>Some options: token device (for example, smart card), or password-encrypted file. Use of a plaintext file to store a secret key is risky because the key is easily compromised if the machine is successfully attacked, but such a measure is necessary for machines that need the ability to cycle without human intervention (typing a password or inserting a smart card). If an LDAP database is used, encryption between the LDAP server and the authenticating party should be considered.</td>
</tr>
<tr>
<td>Web Tier</td>
<td>What other authentication mechanisms are in place (smart cards, biometrics)?</td>
<td></td>
<td>Multifactor authentication combining smart card and biometric authentication is often considered as a reliable personal authentication option.</td>
</tr>
</tbody>
</table>

(continues)
Web Tier  Are the Java security policy files and configuration files protected in the application server?  

Is so, they should be protected with OS specific ACLs and residing on a read-only file system.

Web Tier  Do the application log files show the key-pair values and timestamps for troubleshooting?  

Web Tier  Are the log files and audit trails stored and secured in isolated systems and accessible by authorized personnel only?  

(continues)
### Table 8–12 Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Tier</td>
<td>Are the XML schemas (or DTDs) used to validate the data quality as well as to detect any invalid data or suspicious actions?</td>
<td></td>
<td>Someone could potentially send a valid schema with a petabyte of data in it. This could cause more trouble for the application than a small file that was malformed. The schemas should include restrictions on the amount of data being sent to prevent this type of attack.</td>
</tr>
<tr>
<td>Business Tier</td>
<td>Is authentication and authorization used to control access to particular applications? Is authentication and authorization used to control access to particular data within applications?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Tier</td>
<td>Do users need to reauthenticate themselves for each type of access? Does the application make use of a shared security context propagation?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Tier</td>
<td>Does user authentication expire based on inactivity? How are unauthenticated users prevented from accessing network facilities? Is an authenticated session terminated after a period of inactivity?</td>
<td></td>
<td>This helps pinpoint the extent of resources a malicious, unauthorized user can use up.</td>
</tr>
<tr>
<td>Business Tier</td>
<td>Are the authentication and authorization databases properly protected?</td>
<td></td>
<td>For password-based authentication, encryption of passwords is prudent. Certificate-based authentication databases need only contain public keys, that are by nature secure.</td>
</tr>
</tbody>
</table>

(continues)
## Chapter 8  The Alchemy of Security Design

Table 8–12  Security Reality Checks *(continued)*

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Tier</td>
<td>Are the EJB transfer objects obfuscated?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Tier</td>
<td>Do you use the system ID (or one superuser ID) to update or access business data?</td>
<td></td>
<td>Applications should not use superuser IDs. In required circumstances, it should make use of role-based access control mechanisms to get access to what they need.</td>
</tr>
<tr>
<td>Business Tier</td>
<td>Is the JDBC communication with remote databases protected? Does it use encrypted communication to transmit JDBC statements?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Tier</td>
<td>Do you tightly couple the business logic with the security processing rules in the same application logic code?</td>
<td></td>
<td>N-tier application architecture design allows loose coupling of the business logic with security processing rules for better scalability.</td>
</tr>
<tr>
<td>Business Tier</td>
<td>Do you make use of role-based access to connect with back-end resources?</td>
<td></td>
<td>Role-based access is more secure, flexible, and scalable than user-based access.</td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Are there any sensitive data encapsulated in a SOAP message?</td>
<td></td>
<td>Sensitive data in XML text encapsulated in a SOAP message can be easily snooped. Use of XML Signature and XML Encryption mechanisms to sign and encrypt sensitive payload in SOAP messages is often recommended.</td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Are unused ports, OS services, and network devices disabled?</td>
<td></td>
<td>RPC ports are easily exploited for malicious attacks.</td>
</tr>
</tbody>
</table>

*(continues)*
### Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration Tier</td>
<td>Do you have security appliances to scan and inspect SOAP payload content and attachments for suspected malicious action?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Are SOAP messages containing sensitive data or financial transaction encrypted and digitally signed during transit and storage?</td>
<td></td>
<td>Intermediaries must make use of XML signatures to prove the authenticity and privileges to modify the message contents.</td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Can the intermediary (SOAP proxies) modify the message contents in SOAP messages? Do intermediaries make use of XML Signature?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Do you protect all direct access to the WSDL?</td>
<td></td>
<td>No public access to WSDL should be permitted unless the requesting entity is authenticated and authorized to download them.</td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Does your application mandate selected encryption of the contents of sensitive business data in the SOAP messages?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Do you encrypt SOAP messages that contain sensitive business data between SOAP proxies that route the messages?</td>
<td></td>
<td>SOAP proxies or intermediaries that route SOAP messages should be tamper-proof, and unauthorized access should not change the data contents. One way to provide data integrity and confidentiality is the use of XML Signature and Encryption.</td>
</tr>
</tbody>
</table>

(continues)
Chapter 8  The Alchemy of Security Design

Table 8–12  Security Reality Checks (continued)

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration Tier</td>
<td>Have you properly set up individual user IDs for accessing the UDDI or ebXML service registry?</td>
<td></td>
<td>Some sites do not enforce access user IDs for the UDDI or ebXML service registry. Thus, hackers can easily hack in the service registry.</td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Are your service registries highly available?</td>
<td></td>
<td>UDDI or ebXML service registries can be made highly available by hardware clustering or software clustering (for example, using vendor-specific replication features).</td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Do you allow all users dynamic look-up of WSDL files and dynamic invocation of services?</td>
<td></td>
<td>WSDL can be dynamically looked up and then application can be invoked. The implication is that hackers may easily locate all Web services endpoints easily for future security attacks.</td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Do you timestamp all SOAP messages?</td>
<td></td>
<td>Using timestamps allows identifying and preventing forged messages from further processing. It is also important to synchronize time throughout your environment.</td>
</tr>
<tr>
<td>Integration Tier</td>
<td>Do you set up any time-to-live token for all service requests?</td>
<td></td>
<td>Using time-to-live tokens helps detect DoS attacks using abnormal payloads and malicious messages requiring parsing with endless loops.</td>
</tr>
</tbody>
</table>
One of the most important and most frequently overlooked areas of application development is security testing. While we pay much heed to functional testing, it is surprising how little security testing we do. This may be attributed to many factors:

- Lack of understanding (of the importance of security testing)
- Lack of time
- Lack of knowledge (of how to do security testing)
- Lack of tools

Regardless of the reasons, it is not being done and it poses a serious security risk to the application.

Security testing is a time-consuming and tedious process, most often even more so than functional testing. It is also spread across a variety of disciplines. There are the functional business security requirements that the regular test team will perform. However, there are also non-business functional, or operational, tests that must be performed as well. These can be broken down into two categories, Black Box Testing and White Box Testing.

<table>
<thead>
<tr>
<th>Areas</th>
<th>Security Reality Check Item</th>
<th>Y/N</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration</td>
<td>Do you associate access control and rights for all requesting resources?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tier</td>
<td>Have you performed any security tests such as penetration tests or a regular host security</td>
<td></td>
<td>It is important to ensure that each of the host machines and network</td>
</tr>
<tr>
<td></td>
<td>scan for all intermediaries?</td>
<td></td>
<td>appliances is scanned for any suspicious footprint or unusual security-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>related activities. A security-host OS hardening and minimizing must</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>be performed.</td>
</tr>
</tbody>
</table>

Security Testing

Table 8–12  Security Reality Checks (continued)
Black Box Testing

Zero knowledge or “black box” testing assumes no knowledge of the application. Black box testers approach the application as an attacker would, probing for information about the internals of the application and then attempting a mixture of different exploits based on that information. For example, if a URL of the application contained a “.cgi” extension, it might be inferred that the application was developed in CGI and therefore may be vulnerable to many well-known attacks.

Black box testers will employ a variety of tools for scanning and probing the application. There are hundreds of different tools out on the Internet for hacking into Web applications. These tools attempt everything from port scanning to attempting exploits for multiple implementation languages (Perl, CGI, C, PHP, and Java). Many such tools are quite sophisticated and provide comprehensive vulnerability scanning of any type of Web application. In fact, many operational groups will run such scans on their applications periodically to ensure that they are not vulnerable to the latest automated attacks.

Black box testing will generally not uncover application-code–specific vulnerabilities. It will, however, ferret out unanticipated vulnerabilities arising from the infrastructure that the application was built on. This could range from host or network misconfiguration (routers, services, and OS patches) to problems with the implementation language or version of the virtual machine that it runs on. All of these types of vulnerabilities are not easily foreseen by the developers or business owners and are usually only discovered during black box testing.

White Box Testing

The converse of black box testing is white box testing, where you have full knowledge of the application being tested. In this case, the testers will have access to all configuration information and the source code itself. They will perform code reviews, looking for possible weaknesses in the code and write test harnesses to try to take advantage of those weaknesses. White box testers are generally ex-developers or testers that have enough development background to understand the application’s programming language and know the nuances of it.

White box testers have a toolbox of their own, albeit different from the black box testers. Their toolkit will contain tools that scan the code and can probe it internally. Debuggers and source code scanners are the most prevalent. These allow the testers to find very application-specific bugs and vulnerabilities. Problems such as race conditions and input parameter checking are specific to that application and will not be discovered through black box testing tools (hop
Adopting a Security Framework

Adopting a Security Framework

Most architects and developers adopt a security framework in order to build secure applications because it abstracts the complexity from many underlying security components and is built on proven best practices. The term **security framework** sometimes refers to a set of tools or mechanisms meant for designing and building secure applications. The J2EE platform security architecture builds on J2SE Security including Java Authentication and Authorization Service, Java Cryptography Architecture, and Java Crypto Services APIs (refer to Chapters 3 through 5 for details), do architects and developers really need a security framework in order to build secure Web applications? This section analyzes the logical components of a vendor-independent security framework that are common to many vendor products for building secure Web applications using J2EE-based applications and Web services.

Why do some architects and developers still want a security framework while there are Java API mechanisms? Different Java security APIs are designed to address specific or related problems; they do not necessarily intend to cover all aspects of application security. In addition, the Java security APIs define a large variety of classes and methods but provide no overall instructions about which subset to use in a particular application. An end-to-end security framework for building J2EE-based applications and Web services should be able to simplify the security code development complexity using an abstraction security layer instead of calling individual security components respectively. This should shorten the development life cycle and ease any debugging effort for numerous security components. It should encompass existing J2EE security components in order to meet the security requirements of authentication, authorization, traceability, data confidentiality, availability, data integrity, and non-repudiation. The security framework should also ensure messaging and transaction security across architecture tiers or layers instead of providing security for a stand-alone application or on a stand-alone host. More importantly, it should provide a structured approach to coding the messaging and transaction security using security patterns and best practices. Thus, a security framework is essential for building secure applications.

There are security framework software products available today that can provide security for J2EE-based applications and Web services. But this does not
Chapter 8  The Alchemy of Security Design

mean that we can choose one silver bullet to deliver end-to-end security that addresses all risks and vulnerabilities. Nevertheless, they are usually product-specific and may not provide a platform-independent structured methodology to build end-to-end application security. Thus, some architects and developers still prefer to pick a vendor-independent security framework first to address their security requirements rather than choosing a vendor-specific software product. Some enterprises that have a large IT development team may desire home-grown or custom-made security framework components based on open-source products and/or security vendor products. One possible reason is that they want to avoid vendor lock-in by proprietary security extension features, which may lead to compatibility issues with security standards or potential security risks if these security features are later exploited. Additionally, a customized security framework can be more agile in addressing specific security requirements or legacy system integration requirements, particularly when developers need to support both proprietary and open standards security specifications simultaneously during the transition. Most of these security framework software products or home-grown security frameworks share some similarities. Figure 8–10 depicts a generic logical security framework available from an application security infrastructure provider.

Figure 8–10  Logical security framework of an application security provider
Adopting a Security Framework

Application Security Provider

The Application Security Provider infrastructure consists of single-system application security components that address security functions of organizational security requirements. They do this in terms of security infrastructure services and identity and policy management services. Let’s take a look at the key components offered by an application security infrastructure provider.

Security Infrastructure Services

Security Services. This denotes a set of common security services that are reusable by many applications and system infrastructures. They include authentication, authorization, monitoring, auditing, WS-Security, and so forth. Authentication refers to the verification of the user identity and credentials using a mixture of security mechanisms. Authorization refers to the entitlement that allows a service requester to access resources as per the predefined security policies and access rights. Monitoring refers to the capability to observe and manage application-level security events or exceptions, especially suspicious security activities. Auditing refers to the capability to trace and track application events and exceptions for security auditing or troubleshooting. Web Services Security refers to the security functions required for protection of XML Web services messages—from message creation, routing, and message acknowledgement to the execution of SOAP-based service requests and responses. Key management denotes how key pairs are generated, maintained, or retrieved for authentication or producing digital signatures. Public Key Infrastructure is an example of key management service. Privacy denotes the management of personal or sensitive data by applying data policies to protect them from unauthorized access.

Protocols. The protocols denote the security protocols that are commonly used for security processing (for example, Online Certificate Status Protocol, or OCSP, for verifying user credentials with a Validation Authority in real-time), or securing the user session in the data transport layer with the server (for example, HTTPS).

Handlers. The handlers refer to the interface that interacts with each Java object or component for the access of the common security services or to the service provisioning layer (that ensures appropriate resources are allocated to each service request). Thus, developers can access any of the security services by making use of components such as JSP/Servlets, EJB, RMI/IIOP, SAAJ, or JAX-RPC calls.
Chapter 8  The Alchemy of Security Design

Service Provisioning. This denotes the capability to provision and manage the resources to each authenticated service requester that has the proper access rights and security policies to access the resources. It implies that the resources are available and secure to each service requester and are properly provisioned according to the predefined security policies within a single system or the enterprise. This can be incremental (per usage), segmented (per usage), or transactional (per message or call), depending on the security infrastructure set-up or application system configuration.

Identity and Policy Management Services

Identity Management. This refers to the ability to define and manage user credentials and attributes, and to administer user identity by asserting the user credentials or attributes to the system resources. Single sign-on is a key objective of identity management. Typically, identity management includes a variety of authentication and authorization support mechanisms, including LDAP-based directory server and various security tokens such as user ID and password, Java card, and so forth.

Policy Management. This refers to the ability to define and manage user roles and security policies, and to execute security policies within a single system or across different security domains. It also allows administrators to map the users, groups, roles, and security policies to the protected system resources. In some implementations, they offer a policy management rule engine that facilitates security policy administration and execution.

Although there are vendor implementation differences among application security providers, architects and developers should choose a security framework that adheres to the J2EE security standards and open standards. It should also be able to easily migrate to the new security standards when available (for example, migrating legacy XML-based Security implementation to OASIS’s WS-Security). Some common questions to consider are:

- Do they use proprietary implementation or security-standards–based implementations?
- Are they imposing a vendor-specific architecture, which may restrict the flexibility to apply security patterns and best practices?
- Does the security framework include a structured design methodology to secure applications right from the development phase?
More importantly, the chosen security framework should be flexible and follow security design best practices and patterns, and the vendor product architecture should not dictate how architects and developers tailor the security design. Instead, the framework should make use of the architecturally significant security use cases. A good security framework should enable building vendor-independent security solutions that adopt standards-based technologies, structured security methodology, security patterns, and industry best practices.

Refactoring Security Design

This book is mainly about patterns. It describes a set of security patterns and how, when, and why to use them. It also discusses a methodology for building secure applications using patterns. Often, though, you are not starting from scratch when attempting to secure an application. Furthermore, you rarely choose all of the right patterns and strategies in the design the first time around. To address this, we need to adopt Refactoring.

Refactoring is, as defined by Martin Fowler, “a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior” [Fowler1]. So, why do we want to make a change to the software if we do not need to change its behavior? The answer is that we want to make the system easier to understand and maintain so that we can eliminate any unknown vulnerabilities hiding in the obscurity of a complex design or code implementation. Refactoring leads to simplicity, and security always favors simplicity over complexity.

As Joshua Kerievsky put it, “When you refactor, you relentlessly poke and prod your code to improve its design” [Kerievsky1]. He also goes on to state that the only way to refactor safely is to continuously test the changes. In the case of security pattern refactoring, this involves the functional, white box, and black box testing. Once the refactoring is complete and well-tested, you are left with a better overall design that will be more future-proof. Many of the security holes we see today are not the artifacts of a poor initial design or implementation. Instead, they are the result of series of add-ons and work-arounds that led to overly complex code and inevitably to bugs that are attributable to that complexity. Continuous refactoring protects applications from becoming buggy by continuing to simplify and refine the design and code.
Service Continuity and Recovery

No matter how elegant the security architecture is, if the application cannot sustain security attacks, or fails to recover to continuous business service, the application security design is still crippled. A robust and reliable security design should include design strategies for service continuity and recovery.

A preventive security design will also predetermine the tolerance level of potential security threats. This is usually done by estimating the capacity or sizing of the unexpected security threats and factoring them into the security design process. For example, security architects can benchmark (via simulation or system test) the tolerance level for handling a high influx of simultaneous authentication requests that may be a malicious denial of attack. They can then add any detection and exception-handling process logic in the authentication service security design. A robust security design will add processing logic to handle the service recovery scenarios when a security service is attacked and then restored to previous working condition. For example, the Secure Session Façade pattern should be able to handle session recovery after the application server is restored to normal working condition after security attacks or other non-security-related downtime. This security recovery design strategy typically goes hand in hand with implementing high availability of the security infrastructure for service continuity and requires a procedure to handle the service recovery in the IT security policy documentation.

Conclusion

Security must be omnipresent throughout your infrastructure in order for you to begin to feel your application or service is secure. In order to accomplish this, it is imperative that you follow a structured methodology. In this chapter, we looked at why this methodology must be baked into the development process right from the beginning. Security spans every aspect of your system—from the network perimeter to the service—as shown in the Security Wheel.

To incorporate security into the software development process, we extended the Unified Process to include several new security disciplines. These disciplines define the roles and responsibilities of the different security participants within the software life cycle. The Secure UP (as we called it) ensures that our security methodology can be supported within a software development process. Any security methodology must include this process or one like it. A secure methodology should also include how to adopt security patterns based on security use case
requirements and design analysis as well as how to apply them in appropriate business scenarios.

In summary, we looked at what goes into a good security design. It starts with a methodology, leverages patterns and frameworks, and gets baked into the software development process from the ground up to deliver “Security by Default.”

References

Unified Process

[Unified Process] Enterprise Unified Process:
http://www.enterpriseunifiedprocess.info

Security Principles

[NIST] NIST Security principles:
http://csrc.nist.gov/publications/nistpubs/

[Sun Blueprints] Trust Modeling for Security Architecture Development

Security Patterns

http://www.giac.org/practical/GSEC/Alfred_Amos_GSEC.pdf


Chapter 8  The Alchemy of Security Design


http://www.cse.msu.edu/cgi-user/Web/tech/document?ID=547


Others


[LIBERTY1] Liberty Alliance. Liberty Trust Models Guidelines, Version 1.0


