core
SECURITY PATTERNS

CHRIS STEEL • RAMESH NAGAPPAN • RAY LAI

Security Patterns
For
J2EE Applications, Web Services, Identity Management, and Service Provisioning

Good application design is often rooted in appropriate design strategies and leverages proven best practices using design patterns. Design strategies determine which application tactics or design patterns should be used for particular application security scenarios and constraints. *Security Design patterns* are an abstraction of business problems that address a variety of security requirements and provide a solution to the known security related problem(s). 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 secure code can later be built.

Core security patterns is a collection of proven design patterns for delivering end-to-end security in J2EE applications, Web services, identity management, and service provisioning. These security patterns differ from traditional infrastructure security design patterns in terms of addressing 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.

Typical to Gang-of-four patterns, Core security patterns are 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:

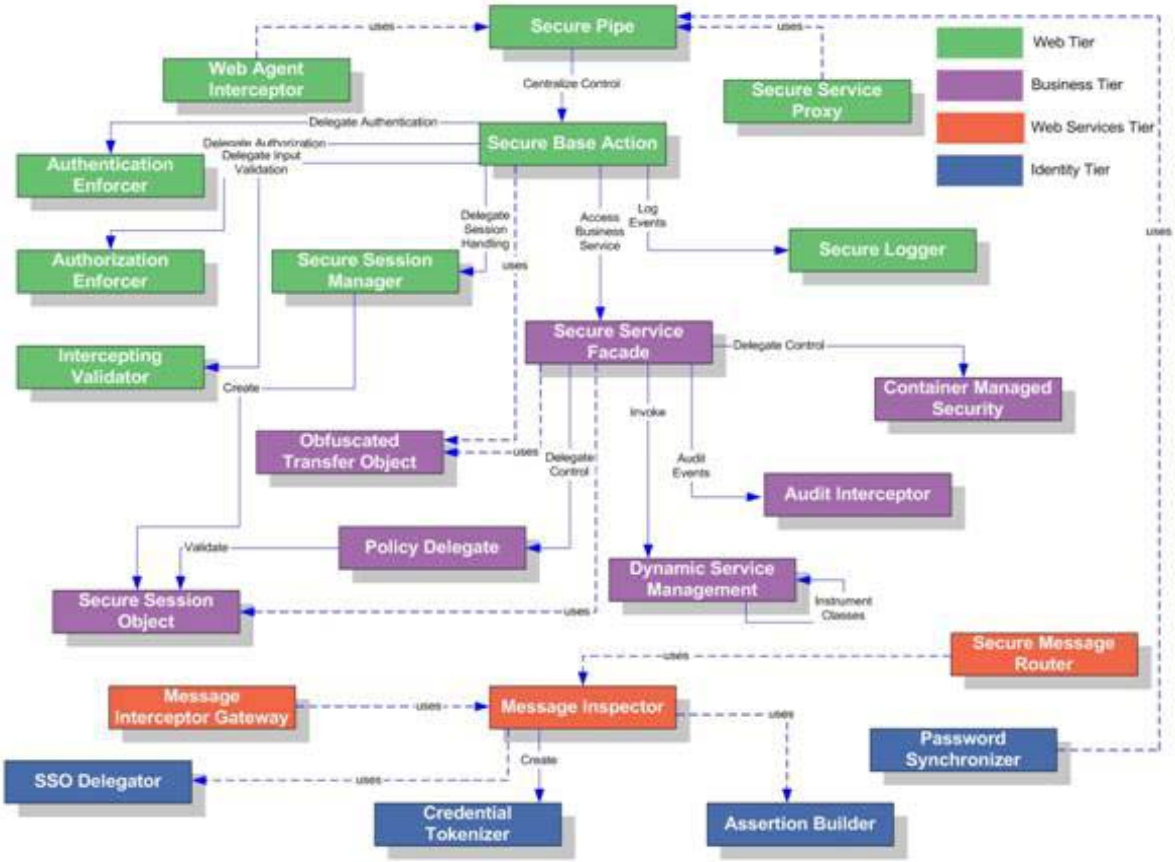
- *Problem*: Describes the security issues addressed by the pattern.
- *Forces*: Describes the motivations and constraints that affect the security problem. Highlights the reasons for choosing the pattern and provides justification.
- *Solution*: Describes the approach briefly and the associated mechanisms 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 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.

Core Security Patterns facilitate securing J2EE based application architecture by applying them in the components and logical tiers such as *Web Tier*, *Business Tier*, *Web Services Tier*, and *Identity Tier*. In the following sections, we present the security patterns catalog and briefly discuss each pattern specific to its logical tier, how they relate to each other with coexisting component tiers and finally how it contributes to the end-to-end security of an application.

Core Security Patterns Catalog



Web Tier Security Patterns

Pattern Name	Standards & Technologies	Description	Related Patterns
Authentication Enforcer	HTTPS; SSL/TLS; IPsec JAAS; JSSE; JCE; JGSS;	This pattern illustrates how a J2EE based application client should authenticate with a J2EE application. Refer to Chapter 9, "Securing the Web Tier: Design Strategies and Best Practices," for details.	**Context Object [CJP];** **Intercepting Filter [CJP]**
Authorization Enforcer	JACC JAAS; JSSE; JCE; JGSS;	This pattern illustrates how authorization should be enforced after user authentication with a J2EE application. *Refer to Chapter 9, "Securing the Web Tier: Design Strategies and Best Practices," for details.*	**Context Object;** **Intercepting Filter [CJP]**
Intercepting Validator	JSP Servlets	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	**Message Inspector; Message Interceptor Gateway**

		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, "Securing the Web Tier: Design Strategies and Best Practices," for details.*	
Secure Base Action	JSP Servlets	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, "Securing the Web Tier: Design Strategies and Best Practices,"*	**FrontController [CJP];** **Command[GoF];** **Authentication Enforcer;** **Authorization Enforcer;** **Secure Logger;** **Intercepting Validator**

		for details.	
Secure Logger	JMX; Java API for logging	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, "Securing the Web Tier: Design Strategies and Best Practices," for details.*	**Abstract Factory Pattern[GoF];** **Secure Pipe;**
Secure Pipe	HTTPS; SSL/TLS; IPsec JSSE	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	**Message Interceptor Gateway**

		authentication and establishing confidentiality or non-repudiation between trading partners. This is particularly critical for B2B integration using Web services. *Refer to Chapter 9, "Securing the Web Tier: Design Strategies and Best Practices," for details.*	
Secure Service Proxy	Servlets JAX-RPC SAAJ	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, "Securing the Web Tier: Design Strategies and Best Practices," for details.*	**Proxy [GoF]** **Intercepting Web Agent;** **Secure Message Router;** **Message Interceptor Gateway;** **Extract Adapter [Kerievsky]**

Secure Session Manager	Servlets EJB	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. *Refer to Chapter 9, "Securing the Web Tier: Design Strategies and Best Practices," for details.*	**Context Object [CJP]**
Intercepting Web Agent	JMX Web server plugin	This pattern helps protecting Web based J2EE applications through a Web Agent that intercepts requests at the Web Container and provides authentication, authorization, encryption, and auditing capabilities. *Refer to Chapter 9, "Securing the Web Tier: Design Strategies and Best Practices," for details.*	**Proxy [GoF]**

Business Tier Security Patterns

Pattern Name	Standards & Technologies	Description	Related Patterns
Audit Interceptor	Java API for Logging; Log4J	Works in conjunction with 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.*	**Secure Logger** **Intercepting Filter [CJP]**
Container Managed Security	EJB	This pattern describes when and how to declare security-related information for EJBs in a deployment descriptor. *Refer to Chapter 10, "Securing the Business Tier – Design Strategies and Best Practices," for details.*	**Secure Pipe**
Dynamic Service Management	JMX	This pattern provides dynamically adjustable instrumentation of security components for monitoring and active management of business objects. *Refer to Chapter 10,*	**Secure Pipe;** **Secure Message Router**

		"Securing the Business Tier – Design Strategies and Best Practices," for details.	
Obfuscated Transfer Object	JCE	This pattern describes ways of protecting business data represented in transfer objects and passed within and between logical tiers. *Refer to Chapter 10, "Securing the Business Tier – Design Strategies and Best Practices," for details.*	**Transfer Object [CJP];**
Policy Delegate	JACC EJB XACML	This pattern creates, manages, and administers security management policies governing how EJB tier objects are accessed and routed. *Refer to Chapter 10, "Securing the Business Tier – Design Strategies and Best Practices," for details.*	**Secure Base Action;** **Business Delegate [CJP]**
Secure Service Facade	EJB	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	**Secure Service Proxy;** **Session Façade [CJP]**

		a secure session. Session information can be also captured and tracked in conjunction with the Secure Logger pattern. *Refer to Chapter 10, "Securing the Business Tier – Design Strategies and Best Practices," for details.*	
Secure Session Object	EJB	This pattern defines ways to secure session information in EJBs facilitating distributed access and seamless propagation of security context. *Refer to Chapter 10, "Securing the Business Tier – Design Strategies and Best Practices," for details.*	**Transfer Object [CJP];** **Session Façade[CJP]**

Web Services Tier Security Patterns

Pattern Name	Standards & Technologies	Description	Related Patterns
Message Inspector	XML Encryption; XML Signature; SAAJ;	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	**Message Interceptor Gateway,** **Secure Message Router**

	JAX-RPC; WS-Security; SAML; XKMS;	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.*	
Message Interceptor Gateway	JAX-RPC; SAAJ; WS-Security XML Signature; XML Encryption; SAML XACML WS-*	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, "Securing Web Services: Design Strategies and Best Practices," for details.*	**Secure Access Point,** **Message Inspector,** **Secure Message Router**
Secure Message Router	XML Signature	This pattern facilitates secure XML communication with multiple partner endpoints that adopt	**Secure Access Point,** **Message Inspector,** **Message Interceptor**

	XML Encryption WS-Security Liberty Alliance SAML XKMS	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, "Securing Web Services: Design Strategies and Best Practices," for details.*	**Gateway**

Identity Management and Service Provisioning

Pattern Name	Standards & Technologies	Description	Related Patterns
Assertion Builder	SAML; Liberty Alliance	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.*	**Single Sign-on Delegator**
Credential Tokenizer	SAML; Liberty Alliance	This pattern describes how a principal's security token can be encapsulated, embedded in a SOAP	**Secure Session Object**

		message, routed, and processed. *Refer to Chapter 12, "Securing the Identity: Design Strategies and Best Practices," for details.*	
Single Sign-on (SSO) Delegator	SAML; Liberty Alliance	This pattern describes how to construct a delegator agent for handling a legacy system for single sign-on (SSO). *Refer to Chapter 12, "Securing the Identity: Design Strategies and Best Practices," for details.*	**Service Locator [CJP]** **Business Delegate [CJP]**
Password Synchronizer	SPML	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.*	

[CJP] Alur, Crupi and Malk: Core J2EE Patterns (Prentice Hall 2003)