Security Design Patterns

- Overview
- Software Development Lifecycle
- Enterprise Software Design Process and Artifacts
- Pattern Format
- Aspect Oriented Programming
Security Design Patterns

- Focus of this presentation
- Architecture-centric (AOP)
- Enterprise Focus
- Technology Agnostic
- Collaboration between Security, Business, and Development
Development Lifecycle

- Software Development Lifecycle
  - Analysis: focuses on requirements gathering and high level definitions
  - Design: drills down on technical issues, distributions, and refines requirements
  - Construction: building and testing the system
  - Transition: "going live!"
SW Security Architect Role

- Provides Leadership
- Facilitate Collaboration between disparate stakeholders
- Focus on Design Process
Analysis Phase

- "A problem, properly stated, is a problem on its way to being solved," Buckminster Fuller
- Concerned with the “what” not the “how”
- What is the business value of security?
- Artifacts
  - Functional & non-functional requirements
    - Security requirements are often “negative”
  - Use Cases
Use Case

- A specific way to capture requirements using actors and actions to show structure and relationships
- Defines both text document and diagram formats
- Use Cases drive the development process
Use Case

- Use Case Example: user transferring money on bank website system
Use Case

- Use Case Attributes
  - Goal/Context
  - Boundaries
  - Preconditions
  - End Condition: Success/Fail
  - Actor/Roles
  - Actions
Mis-Use Cases

- Look at the system from an attacker point of view
- Useful to glean security requirements
- Discussed in paper by Guttorm Sindre and Andreas Opdahl.
Mis-Use Case Example

- Attacker View of Bank Website
Mis-Use Case Benefit

- Defending Against Login Subversion
Design Phase

- Goals of this phase include
  - System, object, component design
  - Prototyping
- Design Artifacts
  - CRC Cards: Class, Responsibility, Collaboration
  - Class & Sequence Diagrams
  - Common Services: Logging/Security/Exception
Threat Modeling

- Elaborates on threats in MisUse case analysis
- Focus on distilling:
  - Threat impact level
  - Threat likelihood
  - Mitigation, management, and containment
Design Patterns

• Christopher Alexander
  – “Timeless Way of Building” & “Pattern Language”

• Pattern definition
  – "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice," Alexander
Design Patterns

• Gang of Four “Design Patterns”
  – Defined three pattern types
    • Creational
    • Structural
    • Behavioral

• Basic Pattern Template
  – Problem, Context, Solution
Security Design Patterns

- Derived from Solutions to Mis-Use Cases and Threat models
- Encompass “prevention, detection, and response” (Schneier, “Secrets and Lies”)
- Context and pattern relationships equally important as individual problems and solutions
Input Validator Pattern

- Context: distributed applications are typically built to be client independent.
- Problem: a minimum of assumptions and control of client interface creates possibility of malicious input. Malicious input can be used to gain unauthorized access to system processes and resources.
Input Validator Pattern

- Solution: Do not trust input. Validate input against acceptable value criteria.
Improving The Solution with AOP

• Aspect Oriented Programming Basics
  – AOP and OOP collaborate
  – Ability to address cross cutting concerns (like security!) in a modular way
  – Component Relationships
  – Tool Support: AspectJ, HyperJ (IBM), AspectWerks, Nanning (see www.aosd.net)
  – Not Just Java
AOP Concepts

- AspectJ Basics
  - Aspect
  - Join Point
    - Location
  - Pointcut
    - Context gathering/assembling
- Advice
- Introduction
Refactoring with AspectJ

- Login Use Case
Refactoring with AspectJ

- Additional Use Cases
Refactoring with AspectJ

- Classes with Getters

```
sdp::Login
- getUserInfo()
- setUserInfo()

sdp::AccountManager
- getAccountInfo()
- setAccountInfo()
- updateAccount()

sdp::Reporter
- getCriteria()
- getReport()
- setCriteria()
- setReport()

sdp::ContactForm
- getContacts()
- setContacts()
```

Input Validation Required
Refactoring with AspectJ

- AspectJ modularizes common behavior

```java
before(): call(void Facade+.get*(..))
|| call(void Facade+.update*(..))
{
    InputValidator.validate();
}
```
Exception Manager Pattern

- “If I wanted you to understand I would have explained it better,” Johan Cruyff
- **Context:** differentiate between exception handling and exception management
  - Java exception handling paradigm
- **Problem:** exceptions can write sensitive data, i.e. Database connection info, to logs or to user screen.
Exception Manager Pattern

Solution: Use structured exception handling, wrap exceptions, and sanitize exception information for display.
Secure Logger Pattern

- Context: balance between performance and analytical purposes
- Problem:
  - Distributed Systems
  - Centralize vs. decentralize
  - Time
  - Management
Secure Logger Pattern

- Solution: remote logging host
Secure Logger Pattern

- Solution: deployment diagram
Secure Logger Pattern

- Logging in Java
Secure Logger Pattern

- SloggerAspect.java

  before(): call(void Facade+.get*(..))
  || call(void Facade+.update*(..))
  {
    //assemble context init logger methods;
  }

  after(): call(void Facade+.get*(..))
  || call(void Facade+.update*(..))
  {
    //final logger methods;
  }
Patterns

- Modular Behavior
Construction Phase

- Concerned with building, integrating, and testing code
- Iterate
- Use unit tests like Junit (www.junit.org) and Nunit to validate your design assumptions
Build and Unit Test Process

- Separation of privileges
  - Developer Level
    - Compile
    - Unit test
  - Integration Level
    - Build
    - Configure
    - Deploy
    - Promote
Transition Phase

- "There's nothing like bringing in a herd," City Slickers
- Moving to operational mode
- Where security usually begins
- Operational plans, monitoring processes & Incident response
Questions?

- More information and free, monthly architecture newsletter at: www.arctecgroup.net/articles.htm