SECAPPDEV 2008
Security Architectures

Riccardo Scandariato
Wouter Joosen
Users Creates

Requirements

Architect Creates

Software Architecture

Prescribes

Developers Creates

Software Product

Prescribes
Iterative Software Development

Software Concept

Requirements Analysis

Design of Architecture

Design and Implement a Version

Test and Deliver the Version

Introduces various feedback loops

An idea why we ended up with eXtreme Programming?
Software Architecture

Sign off
The play

- Act I – Prologue
  - Introduction to Software Architectures

- Act II – Security on stage
  - Security Architectures with Patterns

- Final rehearsal
  - A case study
Act I

Software Architectures
Objectives

- What is Software Architecture?
- Why is Software Architecture important?
- How to Create Software Architecture?
- How to Evaluate a Software Architecture?
Is this an architecture?

Boxes and arrows
The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.
Other Definitions

“Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution”[IEEE 1471]

Importance of architecture
Reconcile stakeholders
Importance of architecture
Impact on requirements

Twin Peaks
Creating software architectures

- Architectures are largely influenced by **software qualities** (non functional requirements)

- Software qualities
  - Performance
  - Modifiability
  - Availability
  - Security
Creating SA Quality Models

How achieve software quality?
- Understand what quality means: quality model
- Verify that quality is achieved: measure

Quality Model
- ISO9126, Boehm, etc
Creating SA Quality Model

Important High Level Quality Factors

- Usability
- Reliability
- Efficiency
- Reusability
- Maintainability
- Portability
- Testability

Low Level Criteria

- Communicativeness
- Accuracy
- Consistency
- Device Efficiency
- Accessibility
- Completeness
- Structuredness
- Conciseness
- Device Independence
- Legibility
- Self-descriptiveness
- Traceability

Product Operation

Product revision
Creating SA
Attribute-driven design

• A recursive decomposition process where, at each stage, tactics and architectural patterns are chosen to satisfy a set of quality scenarios and then functionality is allocated to instantiate the module types provided by the pattern.
Creating SA
Quality attribute scenario

Source: Developer

Stimulus: Wishes to change the UI

Environment: At design time

Artifact: Code

Response: Modification is made with no side effects

Response measure: In 3 hours

Tactics to Control response
Creating SA Tactics & patterns

Bass, Clements, Kazman

Qualities

Security tactics

Resisting Attacks
- Authenticate Users
- Authorize Users
- Maintain Data Confidentiality
- Maintain Integrity
- Limit Exposure
- Limit Access

Detecting Attacks
- Intrusion Detection

Recovering from Attacks
Restoration: (see Availability)
Identification:
- Audit trail

Pattern
Creating SA Algorithm

1. Choose the module to decompose
2. Refine the module
   a) Choose architectural drivers
   b) Choose architectural patterns (from strategy)
   c) Instantiate child modules and allocate functionality (from use cases). Document in multiple views
   d) Gap analysis
3. Repeat
Documenting SA
Architectural Views

- Views on human body 😊

- An architectural view is a simplified description (abstraction) of a system
  - From a particular perspective
  - Covering particular concerns, and
  - Omitting entities that are not relevant to this perspective
Documenting SA
Architectural Views

- At least
  - Decomposition
  - Interaction
  - Deployment

- Mapping between views
  - Important
  - Hard
Documenting SA Decomposition
Documenting SA Interaction

1. Off-Hook
2. Dial tone
3. Digit
4. Digit
5. Open conversation

Joe:Controller → Joe:Terminal → Numbering plan

Conversation
Documenting SA Deployment

- **Foo**
- **LAN**
- **Application Server**
- **Bar**
- **LAN connection**
- **Vocal gateway Cisco 3640**

**Client**

**Server**

**Device**
Evaluating SA
Motivation

- Creating the “right” system for a set of given requirements is still a general problem in software system development [SEI]
Evaluating SA
Boehm costs of change
Evaluating SA Motivation

Source of Problems in Software Development

- Architecture Design
- Requirements Engineering
- Software Implementation
- Hardware Implementation
- Evolution and Maintenance

Evaluating SA Output

- Is this architecture **suitable** for the system for which it was designed?
  - Resulting system will meet quality goals
  - System can be built using available resources

- Architectural **risks**
  - What are the weak points of the architecture?

- Architectural **trade-offs**
  - Choices are made explicit
Evaluating SA
Who’s involved?

- **Evaluation Team**
  - Team leader
  - Evaluation leader
  - Scenario Scribe
  - Proceedings Scribe
  - Timekeeper
  - Questioner

- **Customer Roles**
  - Decision Maker
  - Software Architect
  - Other stakeholders
Evaluating SA
Architectural approaches

- Examples
  - Used a layered architecture
  - Use of distributed data

- I.e., architectural styles (patterns)

- Examples in security
  - Use of interception
  - Use of process separation
  - Use of single access point
Evaluating SA
Elicit and prioritize scenarios

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>

Do these first
If time permits, do these
Do not do these
Evaluating SA
Analyze

<table>
<thead>
<tr>
<th>Scenario A8.1</th>
<th>Search, browse, and order submission is down less than 1 hour/week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Availability</td>
</tr>
<tr>
<td>Architectural approaches</td>
<td>Risk</td>
</tr>
<tr>
<td>AD9 Backup copy of database on tape (not disk)</td>
<td>R9</td>
</tr>
</tbody>
</table>

R9. Recovery from tape can take more than 1 hour in case of large amount of data
Act II
Security Architectures
Objectives

- What Are Security Patterns?
- How to systematically bridge from security requirements (problem domain) to security-aware software architecture (solution domain)?
Security patterns

Well-known (and sound) solution for a recurring security problem, whose pros & cons are known in advance

- A (security) pattern describes... [Doug Lea]
  - a single kind of (security) problem
  - the solution as a constructible software entity
  - design steps for constructing the solution

- Potential helpful tools to implement security
Example: Audit Interceptor

- Structure

Client sends Audit Interceptor forwards Target

Audit Event Catalog uses Audit Log

Diagram showing the flows and connections between the Client, Audit Interceptor, Target, Audit Event Catalog, and Audit Log.
Example: Audit Interceptor

- Sequence Diagram

```plaintext
: Client : Audit Interceptor : Audit Event Catalog : Audit Log : Target

1: request -> 2: lookup
2: lookup -> 3: log
3: log <-> 4: forward
4: forward <-> 5: reply
5: reply <-> 6: lookup
6: lookup <-> 7: log
7: log
8: reply
```
Existing inventories

Security patterns landscape
Data set

- 38 publications
- 218 patterns
- 1996-2006

Abstraction level
Overlaps
Quality issues
No structure

Graph showing:
- Number of patterns over years (1996-2006)
- Key milestones:
  - Trigger
  - Inflated expectations
  - Enthusiasm
  - Illusion
  - Enlightenment

Graph highlights:
- Yearly increments in pattern count
- Fluctuations and trends over the years
Security patterns landscape

Quality

Grade pattern elements

- Problem
- Structure
- Behavior
- Consequence
- Example

\[ Q = \sum w_i \frac{s_i}{\text{max}} \]
Problems & our approach

- Quality & quantity:
  - Not all published patterns are actual patterns
  - Overlapping/duplicate descriptions
  - Descriptions are lacking in detail
  - Essentially: too many unstructured patterns

- How to choose and implement the right pattern?
  - ... reading them all?
    - done that, not recommendable ;)

- Our approach:
  - Collect good patterns in a structured inventory
  - Integrate selection in software engineering process
Security patterns catalog
Overview

- Abstraction level
  - Categorization
- Quality
  - Template
- Overlaps
  - Grouping
- No structure
  - Inter-pattern relations

- Support for methodology
  - Security objectives
  - Trade-off labels
Security patterns catalog
Categorization

Code required

Locality principle
Security patterns catalog

Relations

- Depends on
- Benefits from
- Conflicts with
- Impairs
- Alternative
<table>
<thead>
<tr>
<th>Security patterns catalog</th>
<th>Relations – In practice</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Demilitarized Zone</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
</tr>
<tr>
<td>Secure Pipe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>Load Balancer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>Audit Interceptor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Application Architecture</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authentication Enforcer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A,C</td>
</tr>
<tr>
<td><strong>Application Design</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Limited View</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A,C</td>
</tr>
<tr>
<td>Full View with Errors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Security patterns catalog
Objectives

Confidentiality
- Data Confidentiality
  - Storage Confidentiality
  - Transmission Confidentiality
    - Authorization†
  - Application Confidentiality

Integrity
- Data Integrity
  - Storage Integrity
  - Transmission Integrity
    - Authorization†
  - Application Integrity
    - Authorization†

Accountability
- Non-repudiation
- Auditing
  - Requires authentication‡

Availability

Privacy
- Anonymity

† Authorization
‡ Authentication

 Requires authentication‡
Security patterns catalog

Objectives – In practice

- Confidentiality
  - Data confidentiality
  - Storage confidentiality
  - Transmission confidentiality
  - Application confidentiality
- Integrity
  - Data integrity
  - Storage integrity
  - Transmission integrity
  - Application integrity
- Accountability
  - Non-repudiation
  - Auditing
  - Logging
- Availability
- Privacy
- Anonymity
- Authentication
- Authorization
## Security patterns catalog

### Trade-off labels

<table>
<thead>
<tr>
<th>ISO 9126</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Dependability</td>
<td></td>
</tr>
<tr>
<td>- Portability</td>
<td></td>
</tr>
<tr>
<td>- Maintainability</td>
<td></td>
</tr>
<tr>
<td>- Performance</td>
<td></td>
</tr>
<tr>
<td>- Usability</td>
<td></td>
</tr>
<tr>
<td>- Manageability</td>
<td></td>
</tr>
<tr>
<td>- Auditability</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Confidentiality</td>
</tr>
<tr>
<td>- Integrity</td>
</tr>
<tr>
<td>- Accountability</td>
</tr>
<tr>
<td>- Availability</td>
</tr>
<tr>
<td>- Cost</td>
</tr>
</tbody>
</table>

Denote strong points and weaknesses, e.g. Audit Interceptor:  
- Performance  
+ Accountability
# Security patterns catalog

## Bringing it together

- **Purpose:** uniformly describing patterns
- **Ensures that all relevant data is included**
- **Summarizes information for quick selection**

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Intent</th>
<th>Also known as (optional)</th>
<th>Applicability</th>
<th>Security objective</th>
<th>Labels</th>
<th>Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Dependencies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Impairments</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Conflicts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Benefits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Alternatives</td>
</tr>
</tbody>
</table>

1. Problem
   - **Forces**
2. Example
3. Solution
   - **Structure**
   - **Dynamics**
   - **Participants**
   - **Collaborations**
4. Implementation (optional)
5. Pitfalls (optional)
6. Consequences
7. Related patterns
8. Known uses
Attribute-driven design

Bass, Clements, Kazman

Main Security Objective

Labels:
- Availability
- Security

Pattern

Quality

Security tactics

Resisting Attacks
- Authenticate Users
- Authorize Users
- Maintain Data Confidentiality
- Maintain Integrity
- Limit Exposure
- Limit Access

Detecting Attacks
- Intrusion Detection

Recovering from Attacks
Restoration: (see Availability)

Identification:
- Audit trail

Our approach

Pattern
Methodology Analysis

Analysis

Domain model → Functional Requirements → Security Requirements
- Using *misuse cases*
- Categorized by *security objective(s)*

Architecture
Methodology
Architecture (inspired by ADD)

Analysis

Architecture

Initial architecture → Quality attributes → A requirement

Select patterns from catalog

Tradeoff using labels

Check Conflicts, Dependencies, Benefits, ... using relationships
Methodology
Experimentation

<table>
<thead>
<tr>
<th></th>
<th>Functional components</th>
<th>MUCs</th>
<th>Patterns</th>
<th>Extra components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calendar</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>ATM</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>E-health</td>
<td>7</td>
<td>92</td>
<td>13</td>
<td>10</td>
</tr>
</tbody>
</table>

Digital Publication System: new experiment this year, with students (including evaluation)
Final rehearsal
Case study
E-Health Information Platforms

- Distributed health-care providers in Flanders
  - Hospitals, general practitioners, others
  - Large amount of data and proprietary systems
- Federated IT infrastructure
  - Enables smooth collaboration
  - Patient-centric
  - Access to data anytime, anywhere
E-HIP: example scenario
Mammo screening

1. visit

Ms. Smith

1. visit

Radiology Center

2. Upload pictures first reading

Screening Center

3. Download pictures first reading

4. Upload second reading + report

5. Download report

General Practitioner

E-HIP platform
IHE-XDS
Reference model
Methodology

Start with initial architecture

Hospital Leuven
- Hospital employee
- Hospital terminal
- Local identity provider
- Repository

Hospital Antwerp
- Hospital employee
- Hospital terminal
- Local identity provider
- Repository

E-HIP gateway
E-HIP registry
E-HIP portal
Local identity provider

MPI

Government
Security analysis
Architecture level

- Threat modeling using STRIDE
  1. Model architecture as Data Flow Diagram (DFD)
  2. Determine threats by using STRIDE

- **Spoofing**
- **Tampering**
- **Repudiation**

- **Information disclosure**
- **Denial of service**
- **Elevation of privilege**

Security analysis
Results

- 86 MUCs
- Security assumptions, architectural similarities
  - No-deletion policy
  - Reuse solution for repository (data) to registry (meta)
- 14 MUCs left

- Gap analysis (business level misuse cases)
  - Consider how XDS/EHIP functionality can be misused
- 6 additional MUCs
Memo

1. Start with initial architecture

2. Tag MUC’s with security objective(s)

3. Prioritize security objectives

4. Select security objective from prioritized list
   a. Select pattern associated with objective
      i. Trade-off based on quality labels
      ii. Take into account benefits, dependencies, impairments and conflicts
# Labeling MUCs

<table>
<thead>
<tr>
<th>Threat</th>
<th>Mitigation Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Authentication</td>
</tr>
<tr>
<td>Tampering</td>
<td>Integrity</td>
</tr>
<tr>
<td>Repudiation</td>
<td>Non-repudiation</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Confidentiality</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Availability</td>
</tr>
<tr>
<td>Elevation of Privilege</td>
<td>Authorization</td>
</tr>
</tbody>
</table>
Initial architecture

- Important qualities: manageability and auditing

- First security objective: confidentiality
  - Is composed of controlled access and secure data transmission
  - We start with controlled access
Example
E-health platform

Confidentiality

Authorization

Select Authorization Enforcer
Benefits: Secure Service Facade, AuthN Enforcer

Authentication

Select AuthN Enforcer
Benefits: Secure Service Facade
Select Secure Service Facade

Secure data transmission
Example

E-health platform

1. Need for authorization
2. Beneficial for authorization enforcer
3. Beneficial for authorization enforcer and authentication enforcer
4. Need for identification
5. Need to filter requests (different types of users)
6. Need for message integrity

Repository

Authorization Enforcer

Credential Tokenizer

Security Pipe

Registry server

Firewall facade

Identification & authorization

Application Firewall

Secure Service Facade

Example E-health platform
E-Health platform

Final architecture
SECAPPDEV 2008
Security Architectures

Riccardo Scandariato
Wouter Joosen
For further reading

○ Software Architecture

○ Documenting Software Architecture
For further reading

- **Architecture Evaluation**

- **Security patterns**
  - [Methodology] Koen Yskout, Thomas Heyman, Riccardo Scandariato, Wouter Joosen, Security patterns: 10 years later, draft paper