Round-trip Security Engineering Using Tactics, Patterns, and the Two-Tier Programming Toolkit

Jungwoo Ryoo, Ph.D., CISSP
Penn State University, jryoo@psu.edu

5/1/2013, Fairleigh Dickinson University Symposium on Advances in Cyber Security and Information Assurance

Outline

• Significance of Software Security
  • Secure software design
  • Security tactics and security patterns
  • Significance of and motivation for our research
  • Round-trip Security Engineering
  • Limitations and future research
  • Q&A

Why Software Security?

• Almost every aspect of our modern society
  • Depends on trustworthy software
The Threats Are Real!

- United Airlines reservation system error (August 2012)
- Knight Capital trading snafu (August 2012)
- LinkedIn Hacked (June 2012)

Software Defects

- Manifestation of design flaws or implementation bugs
- Exposed under certain conditions
  - Naturally
  - Artificially
  - Exploits

Common Software Engineering Practices

- Lack the rigorous controls necessary to minimize the introduction of defects into software
- Because security is often:
  - Not a priority
  - Time to market
  - A financial burden
  - An afterthought
The Goal of Software Security

- Ensure that software functions properly under malicious attacks
- Prevention of software malfunctions by removing or reducing the probability of
  - Design flaws
  - Implementation bugs

However, there is no such thing as perfectly secure software!

- It is impossible to produce an absolutely bug-free software especially when the software is non-trivial.
- A proof is not feasible, and is simply cost prohibitive.
- Your goal is to make a hacker’s job as tough as possible, to avoid becoming a victim.

Outline

- Significance of Software Security
- Secure software design
- Security tactics and security patterns
- Significance of and motivation for our research
- Round-trip Security Engineering
- Limitations and future research
- Q&A
What is software design?

- One of the engineering disciplines in an iterative software development lifecycle (SDLC) phase (Kruchten, 2004).
- While requirements engineering focuses on the problem domain by answering what questions, the design task concentrates on the solution domain by asking how questions.
- What do stakeholders want? vs. How to satisfy a stakeholder requirement?
- The design discipline does not involve coding (also called implementation).

Software Design Examples

Component Diagram

Class Diagram

Types of Software Designs

- Architectural:
  - Addresses global, cross-cutting concerns called quality attributes (also called non-functional requirements) in a specific problem domain
- Low-level design:
  - Deals with local, functional requirements
- Example:
  - Operating System (OS):
    - Features such as file search function (lower level design)
    - Security vulnerabilities (architectural level design)
The Status Quo

- Too much focus on testing what's already been implemented
  - Particularly static analysis
- Software security
  - Afterthought
- Consequences
  - Waste of resources
  - Suboptimal effectiveness

The Reality

- Roughly half of the defects leading to security vulnerabilities found in today’s software
  - Stem from design oversights and mistakes (McGraw 2006)

Analogy

- Which is more secure?
  - A bank building originally designed to withstand burglary attempts, complete with thick walls, security bars, CCTV, etc.
  - A residential building outfitted with the same security features as an afterthought
Roles and Goals of Secure Design

- Transition from
  - Problem domain to
    - What and why
  - Solution domain
    - Who, when, where, and how

- The earliest and best place to contemplate security countermeasures

Roles and Goals of Secure Design

- Provides
  - Guidance
  - Direction

- Establishes bases for
  - Effective enforcement

The Bottom Line

- Build security into software

- Identify and minimize (potential) security vulnerabilities

- Do the above
  - As early as possible
  - As much as possible
  - As cost-effectively as possible
The Challenges

- The dynamic nature of software security threats
  - Constantly evolving

- The best you can do is
  - Continuous risk management throughout an SDLC recursively

Outline

- Significance of Software Security
- Secure software design
  - Security tactics and security patterns
- Significance of and motivation for our research
- Round-trip Security Engineering
- Limitations and future research
- Q&A

Patterns

- Definition
  - Well-known solutions to recurring problems

- Types of patterns
  - Architectural
    - Manifests architectural designs
  - Design
    - Manifests low-level designs

- Examples
  - Role-Based Access Control (RBAC)
  - Input Validation
Architectural Patterns

- Definition
  - Addresses two or more quality attributes in a specific problem domain

- Example
  - RBAC used in Operating Systems (OSes)

- Limitation
  - Does not map directly to a particular quality attribute making it impossible to tell which aspect of an architecture addresses a specific quality attribute concern (e.g., security)

Tactics

- Address a single quality attribute
- Most primitive design building blocks used to compose an architectural pattern

Significance of Tactics

- Provides a means to
  - Justify the adoption of a pattern
  - Customize a pattern to satisfy a new quality attribute requirement
Examples of Security Tactics

Relationship between Tactic Types, Tactic Instances, and Architectural Patterns

- Tactic types
  - Categories
- Tactic instances
  - Examples of tactic types
- Architectural patterns
  - A composite of tactic instances

Outline

- Significance of Software Security
- Secure software design
- Security tactics and security patterns
  - Significance of and motivation for our research
- Round-trip Security Engineering
- Limitations and future research
- Q&A
The Problem

• Design is hard; patterns help.

• But it is common for architectural patterns to be implemented incorrectly, or for them to structurally degrade over time (e.g. Cai et al, 2011; Izurieta & Bieman, 2012).

• For some qualities, this is bothersome. For security it is potentially fatal.

Outline

• Significance of Software Security
• Secure software design
• Security tactics and security patterns
• Significance of and motivation for our research
• Round-trip Security Engineering
• Limitations and future research
• Q&A

Proposed Solution

The Design Guide

• Associate security tactics with security patterns that elaborate a guide to designers.

• Realize this library of security patterns as Codecharts (diagrams in the formal specification language LePUS3).

• Use the Two Tier Programming (TTP) toolkit to automatically ensure conformance of implementations to design specifications.
Proposed Solution - 2

- Example: the “Limit Exposure” tactic.
- Limit exposure. The Limit exposure tactic minimizes the attack surface of a system. This tactic focuses on reducing the probability of and minimizing the effects of damage caused by a hostile action. It is a passive defense since it does not proactively prevent attackers from doing harm. Limit exposure is typically realized by having the least possible number of access points for resources, data or services and by reducing the number of connectors which may provide unanticipated exposure.
- Limit exposure is realized by the Single Access Point pattern, and used (extended) in the Roles and Check Point patterns.

Proposed Solution - 3.

- The Check Point pattern can be expressed as a Codechart, in the TTP toolkit, as follows

- and conformance to implementation can be automatically checked.

Verification Process

- Pattern is defined (or reused in the toolkit):

- Pattern is manually associated with code:
Verification Process - 2

- Resulting in a specific instantiation (in this case for JAAS):

- That is verified as conforming (i.e., properly implemented):

Verification Process - 3

- Whenever a programmer makes a change to the code:

Verification Process - 4.

- It is simple to automatically verify conformance again:
**Elements of the Design Guide**

- Combination of
  - Natural language description
    - Provides the context
  - Formal specification with tool support
    - Provides precision
    - Codecharts and the TTP toolkit
  - Appendix
    - Tactic types and instances matrix
    - LePUS3 Primer

---

**Seamless Transition**

The TTP tool comes with a built-in library of security patterns specified in the design guide. Therefore, no need to transfer specifications.

---

**Patterns Specified So Far**

- Core security patterns including:
  - Single Access Point
  - Check Point
  - Roles
  - Session
  - Full View with Errors
  - Limited View
  - Security Access Layer
  - Intercepting Validator
  - Secure Logger
  - Client Input Filter
  - Password Authentication
  - Password Propagation
  - …
Systems and Patterns Verified

- Check Point in JAAS
- Secure Logger in Java Logging API
- Intercepting Validator in Apache Struts
- Full View with Errors in Lexi

Expected Outcomes

- The Design Guide
- A library of security patterns specified in LePUS3
- Demonstration implementations (positive and negative examples)
- Empirical evidence for the use of the Guide/TTP toolkit

Significance and Validity

- To reiterate: design is hard
  - Most designers get it wrong, often in subtle ways.
  - Even if designers get it right, there are always implementers...
- The Guide and Toolkit can help eliminate large classes of security design and implementation errors.
Limitations and Future Work

1) Currently the toolkit can only specify static properties and relationships.
   • We are working with a group at U. of Manchester led by David Rydeheard to address this shortcoming:
     • Behavioral obligations can be enforced using runtime verification techniques
     • A software module monitors the execution of a program and checks its conformance to constraints.

2) Pattern matching is currently done by hand.
   • We are looking into techniques for approximate pattern matching.

Acknowledgement

• Line-Funded Exploratory New Starts (LENS) program
• Team members
  • Rick Kazman, University of Hawaii/SEI
  • Amnon Eden, University of Essex

IEEE Software Security Course Series
• Foundations of Software Security
• Secure Software Design
• Secure Software Implementation
• Management of Secure Software
Questions and Answers

References - 1


References - 2.