Security Design Refinement through Mapping Tactics to Patterns

Jungwoo Ryoo, Penn State
Rick Kazman, SEI/University of Hawaii
SATURN, San Diego – May 5, 2016
Goals of this Session

- To provide a *quick* look at
  - Software security basics
  - Secure software design
  - Architectural Analysis For Security (AAFS)

- To get some hands-on understanding of the relationships between security tactics and patterns
Plan of Attack

<table>
<thead>
<tr>
<th>Time</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30 – 8:35 AM</td>
<td>Introduction</td>
</tr>
<tr>
<td>8:35 – 8:45 AM</td>
<td>Software Security Basics</td>
</tr>
<tr>
<td>8:45 – 8:50 AM</td>
<td>Discussion</td>
</tr>
<tr>
<td>8:50 – 9:15 AM</td>
<td>Secure Design and AAFS</td>
</tr>
<tr>
<td>9:15 – 9:45 AM</td>
<td>Hands-on Mapping Exercise</td>
</tr>
<tr>
<td>9:45 – 10:00 AM</td>
<td>Discussion</td>
</tr>
</tbody>
</table>
Software Security: A Design-Centric Approach

- Introduction
- Secure Design
- AAFS
Software Security: A Design-Centric Approach

Introduction

Secure Design

AAFS
Why Software Security?

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

$3.8 Million

$154

A Breakdown and Analysis of the December, 2014 Sony Hack

Another incredibly far-reaching in-depth compromise of Sony Pictures has happened, this time by a group known as the Guardians of Peace (GOP). The new compromise has all of the excitement of the old events and more, as blaming North Korea for the attack in retaliation to a movie being released by Sony Pictures is all the rage. Risk Based Security has been keeping an updated timeline of the breach, analyzing the leaked documents, and providing links to additional information.
Software Defects

- Manifestation of design flaws or implementation bugs

- Exposed under certain conditions
  - Naturally
  - *Artificially*
    - Usually by exploits
Software Engineering Practice Today

- Lacks 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 function properly under any circumstances
  - Including malicious attacks

- Prevention of software malfunction 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 (Gödel).

- The proof is not feasible, and is cost-prohibitive.

- Your goal is to make a hacker’s job as tough as possible to avoid becoming a victim.
Defects, Bugs, and Flaws

- Defects = Bugs + Flaws

- Defect (the most comprehensive)
  - Design vulnerabilities
  - Implementation vulnerabilities

- Bug
  - Simple implementation errors
    - e.g., using a function call vulnerable to an attack

- Flaw
  - Deeper than bugs and usually involves design mistakes
    - e.g., too many access points, inconsistent input filtering, ...
We estimate risk as **Risk Exposure (RE)**: $RE(x) = P(x) \times S(x)$

- $P(x)$: the **probability** that defect $x$ will impact the mission of the software
- $S(x)$: the **size** of the loss associated with defect $x$
The Three Security Goals

- **Confidentiality**
  - Computing resources/data/information (processed) *accessible only* to the authorized users

- **Integrity**
  - Computing resources/data/information *modifiable or removable only* by the authorized users

- **Availability**
  - Computing resources/data/information *accessible when needed* by the authorized users
Software Security: A Design-Centric Approach

- Introduction
- Secure Design
- AAFS
To design in security you first need to understand the risks your system faces.

For example, you might use the STRIDE threat model to enumerate risks.

STRIDE stands for: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
Each threat type violates a security property, e.g.
- Spoofing attacks violate the Authentication property
- Tampering attacks violate the Integrity property
- Repudiation attacks violate the Non-Repudiation property
- Information disclosure attacks violate the Confidentiality property
- Denial of service attacks violate the Availability property
- Elevation of privilege attacks violate the Authorization property
Designing in Properties

- To design in a security property, we need an architectural strategy.
- We advocate *patterns* and *tactics*.
Architectural / Design Patterns

- Patterns are proven (conceptual) solutions to recurring design problems.
- Many patterns exist (*thousands*), and they are documented across many pattern catalogs.
- It is difficult to draw a boundary between “design” and “architectural” patterns.
Security Patterns

- Well-known solutions to
  - Recurring security problems
- Refined and instantiated from
  - Security tactics
- Closer to code
And hundreds of security patterns have been cataloged.
Security Tactics

Tactics are more primitive than patterns.

Patterns are constructed from tactics.
Software Security: A Design-Centric Approach

Introduction

Secure Design

AAFS
Architectural Analysis

- Structured way of discovering
  - **Design decisions** in software
    - Present or
    - Absent
  - Quality attribute goals of stakeholders
    - Security,
    - Modifiability,
    - Performance,
    - Usability,
    - Etc.
Significance of Architectural Analysis

- During early design
  - Recommended

- During maintenance
  - After the system is built
    - A basis for refactoring

- Consequences
  - Disruptions
  - Cost overruns
  - Risks
    - Poor security
Motivations

- **Not** too many
  - *Well established* architectural analysis methods
  - Example
    - Architectural Tradeoff Analysis Method (ATAM)

- Not to mention
  - Architectural analysis method specializing in security

- Dire need for *Architectural Analysis for Security (AAFS)*
  - Security: Costly and risky → dominant concern
Our Approach

- Make use of design constructs
  - Helps reason about security
  - Act as an analysis lens to zoom in

- Architectural Analysis For Security (AAFS)
  - Contains
    - Tactic-oriented Architectural Analysis (ToAA)
    - Pattern-oriented Architectural Analysis (PoAA)
    - Vulnerability-oriented Architectural Analysis (VoAA)
  - Uses
    - Interviews
Tactics and Patterns

- Design Technique
  - Used to satisfy a single quality attribute requirement

- “Aha” moment
  - Why not for *architectural analysis*?
Vulnerabilities

- Software Weaknesses
  - Exploitation by attackers
  - Code level

- Vulnerability databases
  - Common Vulnerabilities and Exposures (CVE)
  - Common Weakness Enumeration (CWE)

- Relationship with architectural solutions
  - Missing tactic or pattern
CVE vs. CWE

- **CVE**
  - Individual security incident reports
  - More than 70,000 and still counting

- **CWE**
  - Categories of the incident report
  - 940 entries
Our Approach Provides a Holistic View of Security

- The ultimate goal
  - To identify
    - The **absence or presence** of a design decision
      → ToAA and PoAA
    - The **misinterpretation or violation** of a design decision in the source code
      → VoAA
Steps of Our Methodology

- **Step 1**
  - Tactic-oriented Architectural Analysis (ToAA)
    - Fast

- **Step 2**
  - Pattern-oriented Architectural Analysis (PoAA)

- **Step 3**
  - Vulnerability-oriented Architectural Analysis (VoAA)
Case Study

- OpenEMR
  - Electronic Medical Record (EMR) System
  - Open Source
    - Released in 2001
    - 531,789 LOC
    - Big user base

- Factors in choosing a subject
  - Easy access to architect and source code
ToAA Phase

- Interview an architect
  - Where
  - How

- Identify design
  - Rationale
  - Assumptions

2. With respect to security, what are the approaches that you have taken to address this quality attribute?

<table>
<thead>
<tr>
<th>Tactic</th>
<th>How is it achieved?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detect Intrusion</td>
<td>- Use of logging</td>
</tr>
<tr>
<td></td>
<td>- There is no capability to detect specific intrusion attempts such as SQL injection</td>
</tr>
<tr>
<td>Detect Service Denial</td>
<td>- Not supported by OpenEMR</td>
</tr>
<tr>
<td>Verify Message Integrity</td>
<td>- Partially supported by OpenEMR by means of standardized library function calls specializing in sanitizing user inputs</td>
</tr>
<tr>
<td>Detect Message Delay</td>
<td>- Not supported by OpenEMR</td>
</tr>
<tr>
<td>Identify Actors</td>
<td>- Implemented as part of the OpenEMR business logic</td>
</tr>
<tr>
<td>Authenticate Actors</td>
<td>- Implemented as part of the OpenEMR business logic</td>
</tr>
<tr>
<td>Authorize Actors</td>
<td>- Implemented as part of the OpenEMR business logic</td>
</tr>
<tr>
<td>Limit Access</td>
<td>- Implemented as part of the OpenEMR business logic</td>
</tr>
<tr>
<td></td>
<td>- Use of database views and role-based access control</td>
</tr>
<tr>
<td>Limit Exposure</td>
<td>- Not supported by OpenEMR</td>
</tr>
<tr>
<td></td>
<td>- OpenEMR is not modular at all. Therefore, the impact of a compromise can quickly spread throughout the system.</td>
</tr>
<tr>
<td>Encrypt Data</td>
<td>- Not supported by OpenEMR</td>
</tr>
<tr>
<td>Separate Entities</td>
<td>- Not supported by OpenEMR</td>
</tr>
</tbody>
</table>
PoAA Phase

- Relate ToAA results to Patterns
  - ‘Verify message integrity’ ® ToAA

- Check tactic realization
  - Intercepting Validator
    - Verifies user inputs before they are used
    - Performs filtering to all requests or user inputs
      - According to validation rules
    - Forwards full, partial, or no input to the target
      - Depending on the validation results
VoAA Phase

- Relate PoAA results to CWE categories
  - Ties the suspicion to a piece of code

- CWE entries related to
  - ‘Verify message integrity’ tactic
  - ‘Intercepting validator’ pattern

- CWE 89: Improper neutralization of special elements used in an SQL command
- CWE 87: Improper neutralization of alternate XSS syntax
OpenEMR Analysis Sample Results

- **ToAA**
  - ‘Verify message integrity’
    - Partially supported by
      - Standard library functions for sanitizing user inputs

- **PoAA**
  - No intercepting validator

- **VoAA**
  - CWE 89: *Ad hoc* and incomplete coverage
  - CWE 87: No coverage
Verification

- Vulnerability analysis by IBM AppScan
  - OpenEMR
    - 3.1.0
    - 4.1.2

- SQL injection
  - Improving but still problematic

- XSS
  - Highly problematic
Hands-on Session Part 1

SECURITY TACTICS DEFINITION EXERCISE
Preparation

- Form a team of 4-6 people
- Select
  - Spokesperson
  - Scribe
Team 1 Group Task

- Review the tactics below and write down an example of each
  - Detect attacks
    - Detect intrusion
    - Detect service denial
    - Detect message delay
    - Verify message integrity
Review the tactics below and write down an example of each

- Resist attacks
  - Identify, authenticate, and authorize actors
  - Limit access
  - Limit exposure
Review the tactics below and write down an example of each

- Resist attacks
  - Encrypt data
  - Validate input
  - Separate entities
  - Change default settings
Review the tactics below and write down an example of each

- React to attacks
  - Revoke access
  - Lock computer
  - Inform actors

- Recover from attacks
  - Maintain audit trail
Team Assignments

Security Tactics
- Detect Attacks
  - Detect Intrusion
  - Detect Service Denial
  - Verify Message Integrity
  - Detect Message Delay
- Resist Attacks
  - Identify Actors
  - Authenticate Actors
  - Authorize Actors
  - Limit Access
  - Limit Exposure
  - Encrypt Data
  - Validate Input
  - Separate Entities
  - Change Default Settings
- React to Attacks
  - Revoke Access
  - Lock Computer
  - Inform Actors
- Recover from Attacks
  - Maintain Audit Trail
  - Restore
  - See Availability

System detects, resists, reacts, or recovers
Discussion

- Present and reflect on your findings
Hands-on Session Part 2

MAPPING EXERCISE
Team 1 Group Task

- Read the definition of the following patterns and identify one or more tactic they refine
  - Security session
  - Single access point

- Document your justification
Team 2 Group Task

- Read the definition of the following patterns and identify one or more tactic they refine
  - Checkpointed system
  - Audit interceptor

- Document your justification
Team 3 Group Task

- Read the definition of the following patterns and identify one or more tactic they refine
  - Compartmentalization
  - Role-based access control

- Document your justification
Team 4 Group Task

- Read the definition of the following patterns and identify one or more tactic they refine
  - Single sign on
  - Message replay detection

- Document your justification
Mapping Assignments

- **Team 1**
  - Security session
  - Single access point

- **Team 2**
  - Checkpointed system
  - Audit interceptor

- **Team 3**
  - Compartmentalization
  - Role-based access control

- **Team 4**
  - Single sign on
  - Message replay detection
Discussion

- Present and reflect on your findings
  - How was your exercise?
  - What were your challenges?
  - Was there any missing tactics?