Software Security (Part II): Practices, Trends, Risks

Rattikorn Hewett
NSF SFS Workshop
August 12-16, 2013

Outline
• Software security: interpretation & issues
• Software security practices
  • In software development
  • New trends
  • Software security in Information System Design

Software security: Interpretations
• **Software security** refers to:
  • Software that provides security (e.g., anti-virus software)
  • Security in any software

**Goal:**
To build software that protects
• confidentiality,
• integrity, and
• availability
of resources under the control of authorized users even under malicious attacks
Software security: Issues

• Software is highly connected and exposed
  → diverse & large sets of users/attackers
• Software is highly complex
  → software defects
  → security ramifications, e.g.,
    • Implementation bugs leading to buffer overflow
    • Design security flaws
• Software is highly extensible to large-scale
  → infeasibility of defect-free software
• Software defects with security ramifications are likely to remain ...... regardless how well we learn to develop secure software

Outline

• Software security: meaning & issues
• Software security practices
  • In software development
  • New trends
• Software security in Information System Design

Current principles

• No longer non-functional add-on security features
  • Deal with the whole life-cycle of software development
• No longer secure components by isolation
  • Deal with the whole system and not just software
**How to build Security in Software**

**Touch Point Model:** Incorporate security considerations throughout the Software Development Life Cycle

- Requirements and use cases
- Design
- Test Plan
- Code
- Test Results
- Field Feedback
- Abuse cases
- Security requirements
- Risk-based security
- Risk analyses
- Penetration testing
- Security breaches


**Outline**

- Software security: meaning & issues
- Software security practices
  - In software development
  - New trends
- Software security in Information System Design

**Additional principles**

- No longer non-functional add-on security features
- Deal with the whole life-cycle of software development
- No longer secure components by isolation
- Deal with the whole system and not just software
- No longer one perspective
- Deal with performance vs. usability and risks
- No longer a single concern of securing codes
- Cultivate awareness of software security practice in organizations
Security practices must …

• Be iterative → building blocks
• Enable risk-based customized to the organization
• Provide enough details for non-security-people
• Be simple, well-defined, and measurable

SAMM: Software Assurance Maturity Model*

---

SAMM Security Practices

• Each business function has three security practices covering areas relevant to software security assurance
• Each practice is a ‘silos’ for improvement

---

Each Security Practice

• Defines elements: objectives, activities, results, metrics, cost etc.
• Each element has three successive levels of practices (for improvement over time):
  1. Initial understanding/ad hoc provision
  2. Increase efficiency and/or effectiveness
  3. Comprehensive mastery of the practice at scale
Each business function has three security practices covering areas relevant to software security assurance.

Each practice is a ‘silo’ for improvement.

---

**EG1 – Objective**
Offer development staff access to resources around the topics of secure programming and deployment.

**EG1 – Activities**
- Conduct technical security awareness training.
- Build and maintain technical guidelines.

**EG1 – Results**
- Increased developer awareness on the most common problems at the code level.
- >50% development staff briefed on security issues within past 1 year.

**EG1 – Cost**
- Training course build out or license.

**EG1 – Personnel**
- Developers (1-2 days/yr)
- Architects (1-2 days/yr)

---

**Example of security practice level 1**

---

**Conducting assessments**

---

**SAMM includes assessment worksheets for each Security Practice**

---
SAMM

Provides a model for security practices that help:
- Build a balanced software security assurance program in well-defined iterations
- Define and measure security-related activities throughout an organization
- Demonstrate concrete improvements
- Evaluate an organization’s software security practices

Outline

- Software security: meaning & issues
- Software security practices
  - Development
  - New trends
  - Software security in Information System Design

Motivations

- Today’s modern organizations increasingly rely on web-based information systems to provide:
  - critical functions (e.g., power grid control)
  - day-to-day operations and business services (e.g., banking transactions)
- Security breaches in such applications could have devastating consequences
  → Security has become a necessary requirements
Challenges

• Advances in Communication & IT have made web-based systems more vulnerable
  • pervasive, ubiquitous (e.g., distributed, mobile, wireless) computing ⇒ High exposure & huge diverse users
  • Distinctive features of web-based development:
    • Rapid change and growth ⇒ Uncertainty
    • Living with long life cycles ⇒ Long term effects
    • Increase scope & complexity ⇒ Impossible for large-scale applications to be defect-free

Issues

• Design flaws are estimated to cost about 50% of security problems [Verdon & McGraw, 04]
  • Fixing security after a system was built can be costly, unprincipled, or infeasible
  • Existing security risk methodologies and tools at the design level mostly rely on
    • Best practice standards (e.g., checklists and guidelines) ⇒ not robust to custom designs
    ⇒ vulnerable to unforeseen threats
  • Subjective expert judgments ⇒ tend to be ad-hoc & unstructured ⇒ results are inconsistent, unrepeatable & hard to scale

Current security engineering

Software Development Life Cycle

• Existing risk methodologies at design levels:
  • standard-based e.g., NIST’s ASSET, SEI’s OCTAVE
  • tool-based e.g., Insight’s CRAMM, Microsoft’s STRIDE
Our research objectives

- To develop an analytical framework for building security into web-based information system design
- To enhance risk quantification in high-level design to alleviate current deficiencies

Model-based risk quantification

Risk concepts

- **Risk** refers the possibility of some undesirable event along with the likelihood of its occurrence.
- **Traditional risk quantification**: Given an undesirable event (hazard),
  \[ \text{Risk} = \text{Hazard Likelihood} \times \text{Subsequent Severity} \]
- **Security risk quantification**: Given a threat (attacker’s goal), we can quantify security risk above.

Proposed Risk Quantification

- High-level web-based information system design
- Use case scenarios
- Activity/Workflow/Process Diagrams
- Likelihood of component usage
- Scanned known vulnerability of each HW component
- Number of vulnerabilities
- Exposure Degree
- Vulnerability Ratio
- Quantified Severity
- Quantified attack Likelihoods
- Security Risk
Case study

• A web-based system for material requirements planning (MRP) for manufacturing enterprise (to minimize delay in production and product delivery):
  • takes customer orders through the B2B web front
  • updates and predicts the daily planned orders (can be overwritten by the manager)
  • simulates real-time scheduling and production
  • determines materials required
  • sends material orders to supplier

Use case scenarios & Assumptions

We assume a simplified B2B
• Three use cases
  • customer orders products
  • manager adjusts demands/requirements
  • supplier acknowledges material order
• All payments are off-line transactions
• Access policy:
  • customers & suppliers must login via a web front
  • a manager can do remote login to the MRP system

High-level Software Design
Estimate Exposure Degrees

Customer order products scenario

Estimating component exposure based on its likelihood of usage

Component Dependency Graph

Hardware Deployment

Customer orders products (0.4)
0.5×0.4 = 0.2

Manager adjusts demands/requirements (0.2)
0.33×0.2 = 0.06

Supplier acknowledges material order (0.4)
0.5×0.4 = 0.2

Hardware Deployment

Manager
Customer
Supplier
Client Servers
Web Server
Application Servers
Database Servers

Password/Data
Product Info/Production Plan
Customer & Supplier Info
Product and Material orders

Hardware Deployment

Manager
Customer
Supplier
Client Servers
Web Server
Application Servers
Database Servers

Password/Data
Product Info/Production Plan
Customer & Supplier Info
Product and Material orders

Hardware Deployment

Manager
Customer
Supplier
Client Servers
Web Server
Application Servers
Database Servers

Password/Data
Product Info/Production Plan
Customer & Supplier Info
Product and Material orders
Estimate Component Vulnerability

Assess known vulnerabilities from public databases (e.g., Bugtraq) or running a security scanner (e.g., Nessus)

<table>
<thead>
<tr>
<th>Host</th>
<th>Vulnerability Sources</th>
<th>Vulnerability Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>WH</td>
<td>Apache-Chunk, PHP 4.2</td>
<td>2</td>
</tr>
<tr>
<td>A1</td>
<td>Jboss, JRE 1.4.2, Windows, Tomcat-3.2.1</td>
<td>4</td>
</tr>
<tr>
<td>A2</td>
<td>Jboss, JRE 1.4.2, Windows</td>
<td>3</td>
</tr>
<tr>
<td>A3</td>
<td>Telnetd</td>
<td>1</td>
</tr>
<tr>
<td>D1</td>
<td>Oracle, TNS Listener</td>
<td>2</td>
</tr>
<tr>
<td>D2</td>
<td>Oracle, TNS Listener</td>
<td>2</td>
</tr>
<tr>
<td>D3</td>
<td>Oracle, TNS Listener</td>
<td>2</td>
</tr>
<tr>
<td>D4</td>
<td>Oracle, TNS Listener</td>
<td>2</td>
</tr>
</tbody>
</table>

Vulnerability Ratios

<table>
<thead>
<tr>
<th>Vuls</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>Total</th>
<th>Vul. Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.14</td>
<td>0.16</td>
<td>0.16</td>
<td>0.23</td>
<td>0.09</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Component Dependency Graph

Annotate each component with its vulnerability ratio

The chances of C1 being attacked (via S→C1) is estimated by

S is secured & S transits to C1 safely & C1’s vulnerability

Chances = 1×0.4×0.09 = 0.04

Ignore cycle since it gives lower chance
Component Dependency Graph

Annotate each component with its vulnerability ratio.

The chances of MRP being attacked via CI is estimated by

\[
S \text{ is secured & } S \text{ transits to CI safely & } S \text{ is secured & CI transits to MRP safely & Vulnerability of MRP = 0.23}
\]

\[
\text{Chance} = 1 \times 0.4 \times 0.75 \times 0.23 \approx 0.09
\]

Chances via MI = 0.02

Chances via SI = 0.04

Chances via JOB = 0.01

Chance of MRP being attacked = 0.14

Likelihood Estimates

Severity Analysis

<table>
<thead>
<tr>
<th>Component</th>
<th>Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI</td>
<td>0.04</td>
</tr>
<tr>
<td>SI</td>
<td>0.06</td>
</tr>
<tr>
<td>MI</td>
<td>0.04</td>
</tr>
<tr>
<td>MRP</td>
<td>0.14</td>
</tr>
<tr>
<td>MPS</td>
<td>0.07</td>
</tr>
<tr>
<td>BOM</td>
<td>0.08</td>
</tr>
<tr>
<td>JOB</td>
<td>0.13</td>
</tr>
<tr>
<td>T</td>
<td>0.665</td>
</tr>
</tbody>
</table>

CT (catastrophic: 0.95)
CR (critical: 0.75)
MG (marginal: 0.5)
MN (minor: 0.25)
Exercise

You are to assess security risk of a high level design of a simple online shopping system with given three components with corresponding functions as follow:

- **Product Browser (B):** browse items and add/update customer’s basket
- **Payment Manager (P):** checks customer’s financial credentials and allows submission accordingly. A customer may change his mind to quit or go back to shopping again
- **Record Manager (R):** records shipping information and purchase order. A customer may continue shopping after this

Exercise (cont)

- Step 1 – Design a workflow diagram of the e-shopping process as described
- Step 2 – Construct component dependency model from transitions in one use case
- Step 3 – Compute vulnerability ratio of each component from a given hardware deployment and the number of vulnerabilities resulted from a scanner (in green) below
Exercise (cont)

- Step 4 – Compute attack likelihood of each component based on vulnerability ratio and transitions
- Step 5 – Assess severity, i.e., damages of attack of each component (using standard ISO 2002)
- Step 6 – Compute security risk of each component

Conclusions

- Provide concepts of software security practices
- Present one aspect of research in software security
  - How to evaluate software design with respect to security risks
  - Can be used to determine which components need to be protected from attack risks