Software Security Initiatives for Information Security Officers

Marco Morana
OWASP Cincinnati Chapter

ISSA Cincinnati Chapter Meeting
July 14th 2010

Copyright © 2010 - The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License.

The OWASP Foundation
http://www.owasp.org
Who Am I ?

- Graduated from University of Padova, Italy in 1987 with Dr. Engineering Degree
- Worked as aerospace engineer in Italy between 1990-1994
- Graduated with Master in Computer System Engineering from NPU, California in 1996
- Worked as Software Engineer in Silicon Valley
- Started my security career working at a secure email project for NASA where I developed one of the first commercial applications based upon S/MIME
- As software engineer I developed commercial security tools for ISS (SafeSuite Decisions) in 1998-2000 and for Sybase in 2002-2003
- Project managed a joint-venture security start-up in Italy, Thyreaus (2001)
- Founded my own consulting company, CerbTech LLC in (2002), architected security applications for VISA (2004) and CompuCredit (2005)
- Worked as Sr. Security Software Consultant and software security instructor for McAfee/Fondstone (2005-2006)
- Joined Citigroup in 2007 as Technology Information Security Officer and founded the Cincinnati OWASP Chapter
Agenda For Today’s Meeting

1. Introduction to Software Security Initiatives
2. Building the Business Cases For Software Security
3. The Roadmap Toward Software Security
4. How to Integrate Security into the SDLC
5. Metrics and Measurements
Introduction to Software Security Initiatives
Software Security Initiative: People, Process, Technology

- People: Who manages software security risks
- Process: What where and how security can be build in the SDLC
- Tools: How processes can be automated

Security = Commitment * (People + Tools + Process^2)
Application Security and Software Security

Y1: Security applied later by patching applications

Y2: Security that looks at external symptoms

Y3: Security that is reactive using SIRT or in response to audit and compliance

Y1: Security built into each phase of the project/SDLC

Y2: Security that looks at root causes

Y3: Security that is proactive using design reviews, threat analysis, defensive coding
Assurance of Software Security During the SDLC: Security Toll Gates

- Secure Requirements
- Threat Modeling
- Secure Code Reviews
- Security Testing
# Software Security Frameworks

## Development Process

- Deliver product to customer
- Define project scope
- Analyze impact
- Build product
- Test
- Design
- Development
- Secure Software Best Practices
- Support
- Initiation

## SDLC Phases

<table>
<thead>
<tr>
<th>SDLC Phases</th>
<th>Requirements</th>
<th>Design</th>
<th>Development</th>
<th>Testing</th>
<th>Deployment and Operations</th>
</tr>
</thead>
</table>

## Ongoing S-SDLC Activities

- **Metrics and Measurements, Training, and Awareness**

## S-SDLC Activities

|-------------------|---------------------------|-----------------------------|----------------------------------------|--------------------------------------|----------------------------|-----------------|----------------------------------------|---------------------|----------------|-----------------|----------------|----------------|----------------|----------------------|------------------|

## Other Disciplines

<table>
<thead>
<tr>
<th>Other Disciplines</th>
<th>High-Level Risk Assessments</th>
<th>Technical Risk Assessment</th>
<th>Incident Management</th>
<th>Patch Management</th>
</tr>
</thead>
</table>

## Event Probability

- Mitigate or Reduce Risk
- Avoid the Risk
- Accept Risk
- Share or Transfer Risk

## Event Impact

- High

---

OWASP
Building the Business Cases For Software Security Initiatives
Four (4) Effective Business Cases Around Secure Software
The Case #1 is about compliance with standards such as with the PCI-DSS

- **[PCI-DSS] 6 Develop and Maintain Secure Systems and Applications**
  - All vulnerabilities must be corrected.
  - The application must be re-evaluated after the corrections.
  - Requirement 6.6 options:
    - Manual review of application source code
    - Proper use of automated source code analyzer (scanning) tools
    - Manual web application security vulnerability assessments
    - Proper use of automated web application security vulnerability assessment (scanning)
    - Web Application Firewall (WAF)

- **[PCI-DSS] 11 Regularly Test Security Systems and Processes**
  - Requirement 11.3.2: External application layer penetration test. For web applications, the tests should include, at a minimum, testing for OWASP T10 vulnerabilities
The Case #2 is about reducing the cost to manage security defects

Most of my vulnerabilities are coding and design issues

But are mostly found during pen test in UAT

The cost of fixing them in UAT is 10 X during coding (unit tests)

Source: Applied Software Measurement, Capers Jones, 1996
The Case # 3 is about cybercrime attacks that exploit software vulnerabilities.

170 million card and ATM numbers

Exploited application vulnerabilities such as SQL injection and uploaded sniffers.
The Case #4 is about following what the analysts say about software security

1) “75% of security breaches happen at the application” - Gartner

2) “Over 70 percent of security vulnerabilities exist at the application layer, not the network layer”

3) “If only 50 percent of software vulnerabilities were removed prior to production ... costs would be reduced by 75 percent”

1,2,3 Sources: Gartner
The Roadmap Toward Software Security
A Feasible Plan For Software Security Initiative in 4 steps:

1. **Assess the maturity level** of the software security processes within your organization/company

2. **Start by introducing software security activities as part of the SDLC**
   1. Security Requirements
   2. Secure Design Reviews and Threat Modeling
   4. Security Testing

3. **Measure and manage vulnerabilities and software security risks**

4. **Integrate software security processes with other information security and risk management processes**
Old School Security-enhanced lifecycle process (S-SDLC): MS-SDL, Cigital TP and CLASP
New School Standard Software Security Maturity Models: SAMM, BSIMM

The Software Security Framework (SSF)

<table>
<thead>
<tr>
<th>Governance</th>
<th>Intelligence</th>
<th>SSDL Touchpoints</th>
<th>Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy and Metrics</td>
<td>Attack Models</td>
<td>Architecture Analysis</td>
<td>Penetration Testing</td>
</tr>
<tr>
<td>Compliance and Policy</td>
<td>Security Features and Design</td>
<td>Code Review</td>
<td>Software Environment</td>
</tr>
<tr>
<td>Training</td>
<td>Standards and Requirements</td>
<td>Security Testing</td>
<td>Configuration Management and Vulnerability Management</td>
</tr>
</tbody>
</table>
# Code Review Activities And Capability Levels: BSIMM

<table>
<thead>
<tr>
<th>SSDL TOUCHPOINTS: CODE REVIEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use of code review tools, development of customized rules, profiles for tool use by different roles, manual analysis, ranking/measuring results.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objective</th>
<th>Activity</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1.1 know which bugs matter to you</td>
<td>create top N bugs list (real data preferred) (T: training)</td>
<td>1</td>
</tr>
<tr>
<td>CR1.2 review high-risk applications opportunistically</td>
<td>have SSG perform ad hoc review</td>
<td></td>
</tr>
<tr>
<td>CR1.3 spread software security around without any process</td>
<td>establish coding labs or organize ad hoc review</td>
<td></td>
</tr>
<tr>
<td>CR2.1 drive efficiency/consistency with automation</td>
<td>use automated tools along with manual review</td>
<td>2</td>
</tr>
<tr>
<td>CR2.2 drive behavior objectively</td>
<td>enforce coding standards</td>
<td></td>
</tr>
<tr>
<td>CR2.3 find bugs earlier</td>
<td>make code review mandatory for all projects</td>
<td></td>
</tr>
<tr>
<td>CR2.4 know which bugs matter (for training)</td>
<td>use centralized reporting (close knowledge loop, drive training) (T: strategy/metrics)</td>
<td></td>
</tr>
<tr>
<td>CR2.5 make most efficient use of tools</td>
<td>assign tool mentors</td>
<td></td>
</tr>
<tr>
<td>CR3.1 drive efficiency/reduce false positives</td>
<td>use automated tools with tailored rules</td>
<td>3</td>
</tr>
<tr>
<td>CR3.2 combine assessment techniques</td>
<td>build a factory</td>
<td></td>
</tr>
<tr>
<td>CR3.3 handle new bug classes in an already scanned codebase</td>
<td>build capability for eradicating specific bugs from entire codebase</td>
<td></td>
</tr>
</tbody>
</table>
Capability Maturity Model Levels

- Initial (1): Disciplined process
- Repeatable (2): Standard, consistent process
- Defined (3): Predictable process
- Managed (4): Continuously improving process
- Optimizing (5): Continuously improving process
Software Security Maturity Stages and Levels

■ Maturity Innocence (CMM 0-1)
  ▸ No formal security requirements
  ▸ Issues addressed with penetration testing and incidents
  ▸ Penetrate and patch and reactive approach

■ Maturity Awareness (CMM 2-3)
  ▸ All applications have penetration tests done before going into production
  ▸ Secure coding standards are adopted as well as source code reviews

■ Maturity Enlightenment (CCM 4-5)
  ▸ Threat analysis in each phase of the SDLC
  ▸ Risk metrics and vulnerability measurements are used for security activity decision making
How to Integrate Security Activities into the SDLC
S-SDLC Security Tollgates

Risk = Threat x Vulnerability x Cost

What do we need to test, And how

Code review tools

Iterative approach

Security requirements

Threat and risk Modeling

Secure Design Review

Risk-based security tests

Static analysis (tools)

Penetration testing

Requirements and use cases

Design

Test plans

Code Review

Code

Test results

Field feedback

OWASP
A Prerequisite For A Successful Software Security Program is to Acquire People with the Right Skills
The Initial Step Toward Software Security: From Black Box To White Box Testing

- Manual Penetration Testing
- Manual Code Review
- Automated Vulnerability Scanning
- Automated Static Code Analysis
Automated Source Code Analysis

Static Analysis

Source Code

Raw Issues

Human Review

Findings
MITRE found that all application security tool vendors’ claims put together cover only 45% of the known vulnerability types (over 600 in CWE). They found very little overlap between tools, so to get 45% you need them all (assuming their claims are true).

Beware of the silver bullet security mentality and false sense of security given by tools!
Application Threat Modeling: Data Flow Diagrams

https://www.owasp.org/index.php/Application_Threat_Modeling
Threats, Vulnerabilities and Countermeasures

https://www.owasp.org/index.php/Application_Threat_Modeling
The Holistic Step: Application Threat Modeling

I. Spoofing
II. Repudiation

Access Level Internal

Internal (Web Server/App & DB Server Boundary)

Application Server

Access Level Restricted

(App & DB Server/Financial Server Boundary)

Financial Server

Restricted Network

Customer Financial Data

Account/Transaction Query Calls

I. AuthN, Encryption
II. Digital signatures, HMAC, TS, AuthZ
III. Encryption, AuthZ
IV. Filtering, AuthN
Software Risk Analysis

■ Evaluate The Risk Factors Of Software:
  ▪ Threat (e.g. the cause)
  ▪ Vulnerability (e.g. the application weakness)
  ▪ Technical Impact (e.g. the loss of service/data)
  ▪ Business Impact (e.g. financial loss, fraud, unlawful compliance etc)

■ Calculate The Overall Risk on Insecure Software:
  ▪ Qualitative: Likelihood x Impact (H, M, L)
  ▪ Quantitative: ALE = SLE X ARO
  ▪ Threat Source (STRIDE) x Severity (DREAD)
  ▪ Threat X Vulnerability X Impact (OWASP)
Security Requirements Definition

- Include both *functional requirements* for security controls and *risk derived requirements* from the abuse case scenarios

- Define Security Requirements in Standards
  - Which controls are required (e.g. authentication, authorization, encryption etc)
  - Where should be implemented (e.g. design, source code, application, server)
  - Why are required
    - Compliance and auditing (e.g. FFIEC, PCI, SOX etc.)
    - Mitigation for known threats (e.g. STRIDE)
  - How should be implemented and tested
Risk Driven Security Requirements: Use and Misuse Cases

Requirements Driven Security Testing

The OWASP Testing Guide

- Testing Principles
- Testing Process
- Custom Web Applications
  - Black Box Testing
  - Grey Box Testing
- Risk and Reporting
- Appendix: Testing Tools
- Appendix: Fuzz Vectors

- Information Gathering
- Business Logic Testing
- Authentication Testing
- Session Management Testing
- Data Validation Testing
- Denial of Service Testing
- Web Services Testing
- Ajax Testing
Metrics and Measurements
Vulnerability Management Metrics

- Is code validated against security coding standards?
- Is design of developers trained, using organizational security best practice technology, architecture and processes?
Essential Software Security Metrics

- **Define where:**
  - Tracking security defects throughout the SDLC

- **Define what qualitatively:**
  - Root causes: requirements, design, code, application
  - Type of the issues (e.g. bugs vs. flaws vs. configuration)
  - Severity (Critical, High, Medium, Low)
  - SDLC Lifecycle stage where most flaws originating in

- **Define how quantitatively:**
  - % of Critical, High, Medium, Lows for application
  - % of vulnerabilities closed/open
  - Vulnerability density (security bugs/LOC)
Defect Taxonomy in Support of Root Cause Analysis and Defect Containment Objectives

Analysis to support focused remediation, risk prioritization and tracking:

- **Security Design Flaws**
  - Introduced because of errors in design
  - Can be identified with threat modeling and manual code reviews

- **Security Coding Bugs**
  - Coding errors that result in vulnerabilities
  - Can be identified with secure code reviews and/or tools

- **Security Configuration Issues**
  - Introduced after tests because of a change in secure configuration of either the application, the server and the infrastructure components
  - Can be identified by testing the application close to production staging environment
Examples of Software Security Metrics

**Process Metrics**
- Evidence that security-check points are enforced
  - Secure code analysis
  - Vulnerability assessments
- Evidence that source code is validated against security standards (e.g. OWASP ASVS)?
- Evidence of security oversight by security officers, SME:
  - Security officers signing off design documents
  - SME participate to secure code review
  - Security officer complete risk assessments
- Training coverage on software security

**Management Metrics**
- % of security issues identified by lifecycle phase
- % of issues whose risk has been accepted vs. % of security issues being fixed
- % of issues per project over time (between quarter to quarter)
- % of type of issues per project over time
- Average time required to fix/close vulnerabilities during design, coding and testing
- Average time to fix issues by issue type
- Average time to fix issue by application size/code complexity
Security Metrics Goals The Good and The Bad

- **Good**: if goals when are “SMART” that is Specific, Measurable, Attainable, Realistic, Traceable and Appropriate
  - Example: reducing the overall number of vulnerabilities by 30% by fixing all low hanging fruits with source code analysis during construction

- **Bad**: if the goals justify the means to obtain the goals
QUESTIONS & ANSWERS
Thanks for listening, further references

- Gartner 2004 Press Release

- Software Assurance Maturity Model
  - http://www.opensamm.org/

- The Software Security Framework (SSF)
  - http://www.bsi-mm.com/ssf/

- SEI Capability Maturity Model Integration CMMi
  - http://www.sei.cmu.edu/cmmi/

- The Microsoft Security Development Lifecycle
Further references con’t

- A CISO’s Guide to Application Security

- The Seven Touchpoints of Software Security
  - http://www.buildsecurityin.com/concepts/touchpoints/

- OWASP CLASP

- ITARC Software Security Assurance
Further references con’t

- OWASP Education Module Embed within SDLC
  - http://www.owasp.org/index.php/Education_Module_Embed_within_SDLC

- Producing Secure Software With Software Security Enhanced Processes

- Security Flaws Identification and Technical Risk Analysis Through Threat Modeling