Software Assurance vs. Security Compliance:

Why is Compliance Not Enough?

Carol Woody, Ph.D.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Current Challenge for Security Compliance

Acquisition Life Cycle

- **A**: Material Solution Analysis
  - Material Development Decision
- **B**: Technology Development
  - Post-CDR A
- **C**: Engineering and Manufacturing Development
  - FRP Decision Review

- **Pre-Systems Acquisition**
- **Systems Acquisition**
- **Sustainment**

Certification and Authorization to Operate

Software Patch Cycle

Software Supply Chain
How is Security Compliance Addressed?

Reliability, quality, and effective systems engineering are considered sufficient to address security

Security requirements are

• Established at the system level based on concerns for confidentiality, integrity, and availability (CIA)
• Assigned to components through system engineering decomposition
• Not required until Milestone B
Security Compliance Limitations

CIA principles were developed in 1974, and much has changed since then.

Effective software engineering is not being addressed by system engineers.

Many acquisition decisions affecting security are made before Milestone B.
Origins of CIA - 1


Defined security as

“techniques that control who may use or modify the computer or the information contained in it”

Described the three main categories of concern:

confidentiality, integrity, and availability (CIA)
Origins of CIA - 2

Technology environment in 1974

- S360 in use from 1964-1978
- S370 came on the market in 1972
- COBOL & BAL programming languages in use
- MVS operating system released in March 1974
Origins of CIA - 3

What’s missing?

• Internet
• Morris worm, which occurred in November 1988
• 49,296 common vulnerabilities and exposures (CVE)
• Java, C++, C#
• Mobile computing
• Bluetooth
• Stuxnet attack on isolated supervisory control and data acquisition (SCADA) systems
• Cloud computing
• etc.
Software Assurance

Picks up where compliance leaves off

**Definition:** Software assurance

(DHS Software Assurance Curriculum Project)

Application of technologies and processes to achieve a required level of **confidence** that software systems and services function in the intended manner, are free from accidental or intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures.

7 principles must augment CIA
7 Principles for Software Assurance

1. **Risk**: Perception of risk drives all assurance decisions.

2. **Interactions**: Systems are highly inter-connected and share the risks of all connections.

3. **Trusted Dependencies**: Your assurance depends on other people’s assurance decisions and your level of trust for these dependencies.
4. **Attacker**: A broad community of attackers with growing technology capabilities can compromise any and all of your technology assets - there are no perfect protections, and the attacker profile constantly changes.

5. **Coordination**: Assurance requires effective coordination among all technology participants and their governing bodies.
7 Principles (concluded)

6. **Dynamic**: The threat is always changing. Assurance is based on governance, construction, and operation and is highly sensitive to changes in each area.

7. **Measurable**: A means to measure and audit overall assurance must be built in. If you can’t measure it you can’t manage it.
**Systems Engineering vs. Software Engineering**

**Systems Engineering Assumptions**
- Systems can be decomposed into discrete, independent, and hierarchically-related components (or subsystems)
- Components can be constructed and integrated with minimal effort based on the original decomposition
- Quality properties can be allocated to specific components

**Software Engineering Realities**
- Software components are often related sets of layered functionality (one layer is *not* inside another)
- Interactions of components (*not* the decomposition) must be managed
- Security properties relate to composite interactions (*not* to individual components)

---

**Diagram:**
- **System**
  - **Sub-system**
    - **HW**
    - **SW**

- **Interfaces to capabilities provided by a layer**
  - **applications**
  - **common software services**
  - **generic device access**
    - (e.g., LAN, device drivers)

- **Within and outside of the system**
Role of Software in Systems

From the *NRC Critical Code Report*

“Software has become essential to all aspects of military system capabilities and operations” p.19

- 1960 – 8% of the F-4 aircraft functionality
- 1982 – 45% of the F16 aircraft functionality
- 2000 – 80% of the F-22 aircraft functionality

* Committee for Advancing Software-Intensive Systems Producibility; National Research Council (NRC).

*Critical Code: Software Producibility for Defense, 2010*
Systems Engineering vs. Software Engineering

**Systems Engineering Assumptions**
- Systems can be decomposed into discrete, independent, and hierarchically-related components (or subsystems)
- Components can be constructed and integrated with minimal effort based on the original decomposition
- Quality properties can be allocated to specific components

**Software Engineering Realities**
- Software components are often related sets of layered functionality (one layer is not inside another)
- Interactions of components (not the decomposition) must be managed
- Security properties relate to composite interactions (not to individual components)

---

Systems engineering is insufficient for software-reliant security
Software Assurance Impact on C&A

Focusing on individual systems is insufficient

• Critical services used by the software are not considered
• Differences in security controls for systems tied to the same mission are not considered

Software development is increasingly in the supply chain, and security controls must be considered during acquisition

Missions, which extend beyond a single system, define the functionality that is intended

Software assurance methods are required to build effective operational security
Software Assurance Methods

Mission Thread Analysis
Supply Chain Risk Management
Security Requirements Elicitation (SQUARE)
Measurement
Software Assurance Across the Life Cycle

ACQUISITION LIFE CYCLE (ALC)

Establish Security Requirements
Assured Supply Chain Analysis
Measure Software Program Security Risks
Mission Thread Analysis for Security
Complexity Analysis

Validate Security Requirements
Monitor Evolving Threats
Measure System Product Security Risks

PLANNING REQUIREMENTS ANALYSIS DESIGN/BUILD TEST INTEGRATE IMPLEMENTATION

SYSTEM DEVELOPMENT LIFE CYCLE (SDLC)
Mission Thread Analysis
Mission Thread Analysis

Establish the role of mission success (functioning as intended) for system and software assurance
Analyze potential mission failure
Connect the software and systems to the operational mission
  • How is security defined and validated?
  • Will the mission survive a security compromise?
Tool: Survivability Analysis Framework

Preconditions

People

Actions

Resources

Postconditions

Stresses

Acceptable Outcomes

Analysis

- Potential failure conditions
- Likelihood of error conditions
- Impact of occurrences
- Recovery strategies
Analysis of Mission Failure Potential

Who identifies and manages an error?

• Human or technology control?
• Coordination of responses across multiple components (multiple contractors?)

Which faults should be reported and how?

• Logging and alerting can easily overload resources
• Will the receiver understand an error and know what to do?

How could an attack go undiscovered in the “cracks” between systems?
Building Justified Confidence

Mission Thread Analysis: building the case

Explanation of why confidence is justified

Delivered System

Released System

Acceptance Case

Building the System
Mission Thread Resources

[http://www.sei.cmu.edu/library/abstracts/reports/10tn013.cfm](http://www.sei.cmu.edu/library/abstracts/reports/10tn013.cfm)

[http://www.sei.cmu.edu/library/abstracts/reports/08tr008.cfm](http://www.sei.cmu.edu/library/abstracts/reports/08tr008.cfm)
Supply Chain Risk Management
State of Security in Software Products

MITRE has documented over 700 software errors in commercial products that have led to exploitable vulnerabilities: Common Weakness Enumeration (CWE)\(^1\). 58% of all products submitted to Veracode for testing did not achieve an acceptable security score upon first submission\(^2\). Forrester reports in *Application Security: 2011 And Beyond*\(^3\):

- 47% do not perform acceptance tests for third party software.
- 46% follow a homegrown application security methodology instead of one that had been independently validated.
- 27% do not perform security design.

1. [http://cwe.mitre.org](http://cwe.mitre.org)
2. Fall DHS SwA Forum 2010
3. [http://go.microsoft.com/?linkid=9777219](http://go.microsoft.com/?linkid=9777219)
Limits for Supply Chain Risk Mitigations

Total prevention is not feasible because of the sheer number of risks; limited development visibility; uncertainty of product assurance; and evolving nature of threats, usage, and product functionality.

Responding exploit by exploit is a losing game

- Skilled attackers know system weaknesses better than defenders
- As networks and operating systems are hardened, attackers exploit application software

Identify the risks, establish evidence for what has been mitigated, monitor gaps.
Acquisition of Products
Supply Chain Factors

Supply chain risks for a product is reduced to acceptable level

- **Supplier Capability**: Supplier follows practices that reduce supply chain risks
- **Product Security**: Delivered or updated product is acceptably secure
- **Product Distribution**: Methods used to transmit product to the purchaser guard against tampering
- **Operational Product Control**: Product is used in a secure manner
Acquisition of Systems and Components
System Security Must Be Added

Supply chain risks for **system** reduced to acceptable level

- **Supplier Capability**
  Supplier follows practices that reduce supply chain risks

- **Product Security**
  Delivered or updated product is acceptably secure

- **Product Distribution**
  Methods used to transmit the product to the purchaser guard against tampering

**System Security**
Component products are assembled for effective system security

**Operational Product Control**
Product is used in a secure manner

System design should ensure that externally developed products including legacy software are used in a secure manner
Stronger Integrator Criteria is Needed

Integrator is providing a unique product

Applying practices such as threat modeling at the system level can be more demanding than it is for a product

- **Product development**
  - long product life - incremental
  - focus on software weaknesses appropriate to that supplier’s domain and products, guided by product history
  - relatively small and stable set of suppliers

- **An integration contractor or custom system developer**
  - multiple one-off, relatively short-lived efforts
  - multiple functional domains
  - multiple sets of software products, suppliers, and subcontractors
Supply Chain Resources

http://www.sei.cmu.edu/library/abstracts/reports/10tn026.cfm

http://www.sei.cmu.edu/library/abstracts/reports/10tn016.cfm

Security Requirements Elicitation (SQUARE)
SQUARE

Methodology to help organizations build security into the early stages of the production life cycle

Addresses eliciting, categorizing, and prioritizing security requirements

Security requirements are

- treated at the same time as the system's functional requirements, *and*
- carried out in the early stages
- specified in similar ways as software requirements engineering and practices
- carried out through a process of nine discrete steps
The SQUARE Process

A robust SQUARE tool is available for download from http://www.cert.org/sse/square.html
SQUARE Resources


SQUARE Case Study Reports

http://www.sei.cmu.edu/library/abstracts/reports/08sr017.cfm

http://www.sei.cmu.edu/library/abstracts/reports/06sr003.cfm

http://www.sei.cmu.edu/library/abstracts/reports/09sr017.cfm
Security Measurement
Definitions

**Measurement**
A set of observations that reduce uncertainty where the result is expressed as a quantity\(^1\)

**Measure**
A variable to which a value is assigned as the result of measurement\(^2\)

---

Drivers

Definition
A factor that has a strong influence on the eventual outcome or result

Examples

- **Security Process**: The process being used to develop and deploy the system sufficiently addresses security
- **Security Task Execution**: Security-related tasks and activities are performed effectively and efficiently
- **Code Security**: The code will be sufficiently secure
Drivers: Success and Failure States

The objective when analyzing a driver’s state is to determine how each driver is currently acting.

- The driver is almost certainly in its success state.
- The driver is most likely in its success state.
- The driver is equally likely in its success and failure states.
- The driver is most likely in its failure state.
- The driver is almost certainly in its failure state.
## Drivers for Secure Software Development

### Programmatic Drivers
1. Program Security Objectives
2. Security Plan
3. Contracts
4. **Security Process**
5. Security Task Execution
6. Security Coordination
7. External Interfaces
8. Organizational and External Conditions
9. Event Management

### Product Drivers
10. Security Requirements
11. Security Architecture and Design
12. Code Security
13. Integrated System Security
14. Adoption Barriers
15. Operational Security Compliance
16. Operational Security Preparedness
17. Product Security Risk Management
# Evaluating Drivers

*Directions*: Select the appropriate response to the driver question.

<table>
<thead>
<tr>
<th>Driver Question</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. Does the process being used to develop and deploy the system sufficiently incorporate security?</td>
<td>Yes, Likely Yes, Equally Likely, Likely No, No</td>
</tr>
</tbody>
</table>

*Consider:*
- Security-related tasks and activities in the program workflow
- Conformance to security process models
- Measurements and controls for security-related tasks and activities
- Process efficiency and effectiveness
- Software security development life cycle
- Security-related training
- Compliance with security policies, laws, and regulations
- Security of all product-related information
Driver Profile

The driver profile provides an indication of systemic risk to the mission.

It can be used as a dashboard for program decision makers.
MRD: Focus on Mission Risk

Systemic risk to the mission (also called mission risk) is defined as the probability of mission failure (not achieving key objectives).

From the MRD perspective, mission risk is the probability that a driver is in its failure state.
Measurement Resources


Summary
Compliance Limitations

Based on principles that were developed in 1974 and much as changed since that time

Does not address security for software-reliant systems

Does not address the security risks of software acquisition decisions
Focus on Software Assurance

Ensure that systems and software function as intended and are free from vulnerabilities

Key areas for risk mitigation:
• Mission Thread Analysis
• Supply Chain Risk Management
• Security Requirements
• Measurement
NO WARRANTY

THIS MATERIAL OF CARNEGIE MELLON UNIVERSITY AND ITS SOFTWARE ENGINEERING INSTITUTE IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.