Software Assurance

Level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle and that the software functions in the intended manner.

From "National Information Assurance Glossary" (26APR2010)

Vulnerability: **A flaw in *software* code or a system that leaves it open to the potential for exploitation. Vulnerabilities can be introduced at any point in the supply chain.**

Software Assurance Definition

Emerging Software Assurance

Requirements and Challenges

Software Development Processes & Lifecycle employed

Software Assurance: **Enabling Security and Resilience throughout the Software Lifecycle (NIST)**

MITRE’s CWE, CVE

The National Institute for Standards and Technologies (NIST) leads USG activities related to software assurance, including:

Maintenance of a database of common weaknesses (the CWE)

Notifications of vulnerabilities found in commonly used software products

Tools and Processes for detecting and eliminating the both the unintentional and intentional flaws in the software

Supply Chain Risk Management (SCRM)

Required Components of the program’s SCRM plan

A Risk Assessment is performed to identify the critical components in the information flow of the system

Key critical parts identified receive special SCRM attention to ensure the source is reliable and not counterfeit

Contractor has a SCRM program in place and ensures SCRM requirements are levied to all sub-contractors

Key critical parts are free of malware or malicious code

Software used in the system has a known pedigree (i.e. assured chain of custody)

Software used in the system is free of malicious code

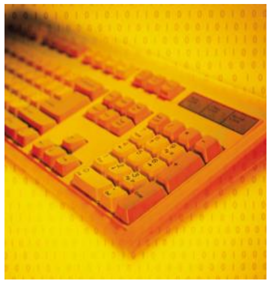
Reference: INFORMATION ASSURANCE PLATFORM INFORMATION TECHNOLOGY GUIDEBOOK
http://www.altus.af.mil/shared/media/document/AFD-111213-023.pdf

Best Practices for Cybersecurity – Static Code Analysis

IEEE
Advancing Technology for Humanity

Testing vs. Static Code Analysis

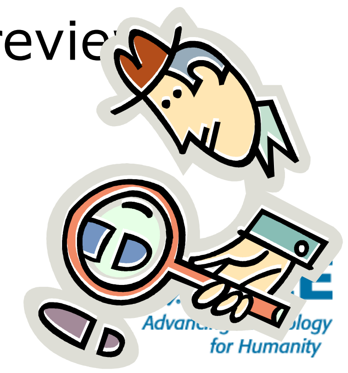
- Testing requires code that is relatively complete
- Static analysis can be performed on modules or unfinished code
- A static analysis tool is a program written to analyze other programs for flaws
  - Such analyzers typically check source code
  - A smaller set of analyzers can check byte code and binary code
- Manual analysis, or code inspection, can be very time-consuming, and inspection teams must know what security vulnerabilities look like in order to effectively examine the code
- Static analysis tools are faster and don't require the tool operator to have the same level of security expertise as a code inspector



Advancing Technology for Humanity

What Code Do You Analyze?

- How do you prioritize a code review effort when you have thousands of lines of source code, and perhaps object code to review?
- From a software assurance perspective, looking at attack surfaces is not a bad place to start
  - A system’s attack surface can be thought of as the set of ways in which an adversary can enter the system and potentially cause damage
  - The larger the attack surface, the more insecure the system [7]
  - Higher attack surface software requires deeper review than code in lower attack surface components.



Heuristics For Code Review – 1

- Howard proposes the following heuristics as an aid to determining code review priority [8]:
  - **Old code**
    - Older code may have more vulnerabilities than new code because newer code often reflects a better understanding of security issues
    - Code considered “legacy” code should be reviewed in depth.
  - **Code that runs by default**
    - Attackers often go after installed code that runs by default
    - Such code should be reviewed earlier and deeper than code that doesn’t execute by default
    - Code running by default increases an application’s attack surface
  - **Code that runs in elevated context**
    - Code that runs in elevated identities, e.g. root in *nix, for example, also requires earlier and deeper review because code identity is another component of attack surface.
  - **Anonymously accessible code**
    - Code that anonymous users can access should be reviewed in greater depth than code that only valid users and administrators can access
  - **Code listening on a globally accessible network interface**
    - Code that listens by default on a network, especially uncontrolled networks like the Internet, is open to substantial risk and must be reviewed in depth for security vulnerabilities

Heuristics For Code Review – 2

- **Code listening on a globally accessible network interface**
  - Code that listens by default on a network, especially uncontrolled networks like the Internet, is open to substantial risk and must be reviewed in depth for security vulnerabilities.
- **Code written in C/C++/assembly language**
  - Because these languages have direct access to memory, buffer-manipulation vulnerabilities within the code can lead to buffer overflows, which often lead to malicious code execution
  - Code written in these languages should be analyzed in depth for buffer-overflow vulnerabilities
- **Code with a history of vulnerabilities**
  - Code that's had a number past security vulnerabilities should be suspect, unless it can be demonstrated that those vulnerabilities have been effectively removed.
- **Code that handles sensitive data**
  - Code that handles sensitive data to should be analyzed to ensure that weaknesses in the code do not disclose such data to untrusted users.
- **Complex code**
  - Complex code has a higher bug probability, is more difficult to understand, and may likely have more security vulnerabilities.
- **Code that changes frequently**
  - Frequently changing code often results in new bugs being introduced
  - Not all of these bugs will be security vulnerabilities, but compared with a stable set of code that's updated only infrequently, code that is less stable will probably have more vulnerabilities in it

Advancing Technology for Humanity

A Three-Phase Code Analysis Process – Phase 1

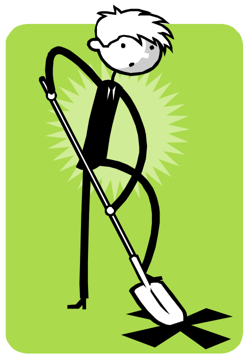
- Howard [6] also suggests a notional three-phase code analysis process that optimizes the use of static analysis tools.
- Phase 1 – Run all available code-analysis tools
  - Multiple tools should be used to offset tool biases and minimize false positives and false negatives
  - Analysts should pay attention to every warning or error
    - Warnings from multiple tools may indicate that the code that needs closer scrutiny (e.g. manual analysis).
  - Code should be evaluated early, preferable with each build, and re-evaluated at every milestone.

A Three-Phase Code Analysis Process – Phase 2

- Phase 2 – **Look for common vulnerability patterns**
  - Analysts should make sure that code reviews cover the most common vulnerabilities and weaknesses, such as integer arithmetic issues, buffer overruns, SQL injection, and cross-site scripting (XSS)
  - Sources for such common vulnerabilities and weaknesses include the Common Vulnerabilities and Exposures (CVE) and Common Weaknesses Enumeration (CWE) databases, maintained by the MITRE Corporation and accessible at: http://cve.mitre.org/cve/ and http://cwe.mitre.org/
  - MITRE, in cooperation with the SANS Institute, also maintain a list of the "Top 25 Most Dangerous Programming Errors" (http://cwe.mitre.org/top25/index.html) that can lead to serious vulnerabilities
  - Static code analysis tool and manual techniques should at a minimum, address these Top 25

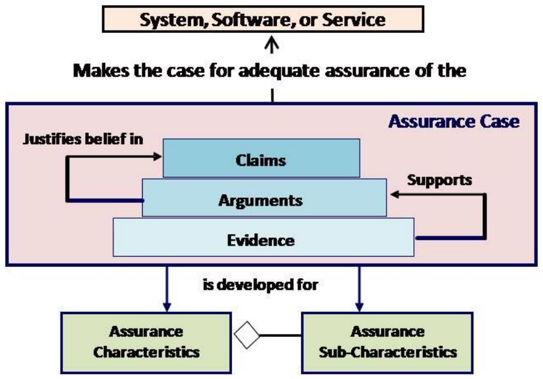
A Three-Phase Code Analysis Process – Phase 3

- **Phase 3 – Dig deep into risky code**
  - Analysts should also use manual analysis (e.g. code inspection) to more thoroughly evaluate any risky code that has been identified based on the attack surface, or based on the heuristics on Slides 34 and 35
  - Such code review should start at the entry point for each module under review and should trace data flow though the system, evaluating the data, how it’s used, and if security objectives might be compromised



The Assurance Case – Capturing the Results of Static Code Analysis as Evidence for Assurance Claims

- An Assurance Case is a set of structured assurance claims, supported by evidence and reasoning that demonstrates how assurance needs have been satisfied [9]
  - It shows compliance with assurance objectives
  - It provides an argument for the safety and security of the product or service.
  - It is built, collected, and maintained throughout the life cycle
  - It is derived from multiple sources
- The Sub-parts of an assurance case include:
  - A high level summary
  - Justification that product or service is acceptably safe, secure, or dependable
  - Rationale for claiming a specified level of safety and security
  - Conformance with relevant standards and regulatory requirements
  - The configuration baseline
  - Identified hazards and threats and residual risk of each hazard and threat
  - Operational and support assumptions
- An Assurance Case should be part of every acquisition in which there is concern for IT security
  - Should be prepared by the supplier
  - Should describe
    - The assurance-related claims for the software being delivered,
    - The arguments backing up those claims,
    - The hard evidence supporting those arguments, including static code analysis results


System, Software, or Service
Makes the case for adequate assurance of the
Assurance Case
Justifies belief in
Claims
Supports
Arguments
Evidence
is developed for
Assurance Characteristics
Assurance Sub-Characteristics

Challenge – Procurement and Maintenance of Tools

- The better static code analysis tools are expensive
  - Use multiple tools used to offset tool biases and minimize false positives and false negatives can quickly become cost prohibitive for a single program
  - In addition, maintenance agreements to ensure a tool is up to date with respect to the spectrum of threats, weaknesses, and vulnerabilities add long term costs
- Buy it once, use it often provides the most bang for the buck
- Pooled-resources analysis labs may make economic sense.

Challenge – Training

- Static code analysis is not for sissies, although it may be for CISSPs® (Certified Information System Security Professionals)
  - This tongue-in-cheek statement belies the difficulty in using static code analysis tools to their best advantage
  - Chandra, Chess, and Steven [10] point out that when static code analysis tools are employed by a trained team of code analysts, false positives are less of a concern; the analysts become skilled with the tools very quickly; and greater overall audit capacity results.
- In order to determine the validity of static code analysis results, it is important for PMs to understand
  - The level of training that code analysts have had with the tools employed for static code analysis
  - Their understanding of code weaknesses and vulnerabilities

Useful Links

- NIST SAMATE Static Analysis Tool Survey
  - The National Institutes for Science and Technology (NIST), Software Assurance Metrics and Tool Evaluation (SAMATE) project, provides tables describing current static code analysis tools for source, byte, and binary code analysis
  - More information on SAMATE can be found at http://samate.nist.gov/
- DHS Build Security In Web Site
  - A wealth of software and information assurance information, including white papers on static code analysis tools
  - More information on Build Security In can be found at https://buildsecurityin.us-cert.gov/daisy/bsi/home.html

IEEE Advancing Technology for Humanity

NIST SAMATE – Source Code Analysis Tools

http://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html

Tool	Language(s)	Avail.	Finds or Checks for	------Date------
C++test .TEST Jtest	C++ C#, VB.NET, MC++ Java	Parasoft	"defects, poor constructs, potentially malicious code and other elements"	4 Apr 2006
cadvise	C, C++	HP	many lint-like checks plus memory leak, potential null pointer dereference, tainted data for file paths, and many others	11 Mar 2009
CodeCenter	C	CenterLine Systems	incorrect pointer values, illegal array indices, bad function arguments, type mismatches, and uninitialized variables	28 Oct 2005
CodeScan	ASP Classic, PHP, ASP.Net	CodeScan Labs	specialise in inspecting web source code for security holes and source code issues.	14 Jul 2008
CodeSecure	PHP, Java (ASP.NET soon)	Armorize Technologies	XSS, SQL Injection, Command Injection, tainted data flow, etc.	16 Mar 2007
K7	C, C++, and Java	Klocwork	Access problems, buffer overflow, injection flaws, insecure storage, unvalidated input, etc.	6 July 2005
Ounce	C, C++, Java, JSP, ASP.NET, VB.NET, C#	Ounce Labs	coding errors, security vulnerabilities, design flaws, policy violations and offers remediation	19 Apr 2007
PLSQLScanner 2008	PLSQL	Red-Database-Security	SQL Injection, hardcoded passwords, Cross-site scripting (XSS), etc.	23 Jun 2008
PolySpace	Ada, C, C++	PolySpace Technologies	run-time errors, unreachable code	25 Feb 2005
PREfix and PREfast	C, C++	Microsoft proprietary		10 Feb 2006
Prevent	C, C++	Coverity	flaws and security vulnerabilities - reduces false positives while minimizing the likelihood of false negatives.	11 Mar 2005

NIST SAMATE – Byte Code Analysis Tools

http://samate.nist.gov/index.php/Byte_Code_Scanners.html

Tool	Language	Avail.	Finds or Checks for	Date
AspectCheck	Java and .NET applications, including ASP.NET, C#, and VB.NET	Aspect Security proprietary	security critical calls	24 Nov 2004
FindBugs™	Java class files	free	null pointer deferences, synchronization errors, vulnerabilities to malicious code, etc. It can be linked to Java source code to highlight the problem in the source.	23 June 2005
FxCop	.NET managed code assemblies	free	checks for conformance to the Microsoft .NET Framework Design Guidelines: more than 200 defects in: Library design, Globalization, Naming conventions, Performance, Interoperability and portability, Security, and Usage.	16 May 2008
Gendarme	.NET Applications	free	extensible rule-based tool to find problems in .NET applications and libraries.	30 Oct 2008
Smokey	.NET or Mono assemblies	jesjo...@mindspring.com	correctness, design, security, performance and other rules	13 Nov 2008
SoftCheck Inspector	Java	SofCheck	creates assertions for each module, tries to prove the system obeys assertions and the absence of runtime errors.	8 Jun 2006
XSSDetect BETA	compiled managed assemblies (C#, Visual Basic .NET, J#)	free	Visual Studio plugin to help find Cross-Site Scripting vulnerabilities (CWE 79). Ignores paths with proper encoding or filtering	10 Jul 2008

NIST SAMATE – Binary Code Analysis Tools

http://samate.nist.gov/index.php/Binary_Code_Scanners.html

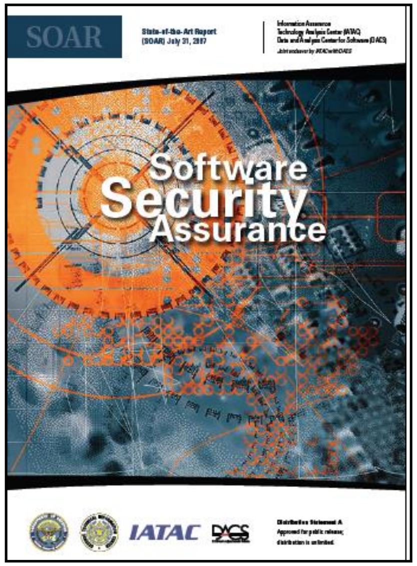
Tool	Language	Avail.	Finds or Checks for	- - Date - -
BugScam	app binaries .EXE or .DLL files	SourceForge	This a package of IDC scripts for IDA Pro to look for common programming flaws.	8 May 2003
CodeSurfer/x86	x86 executables	Grammatech	A prototype system from joint research by the University of Wisconsin and GrammaTech to provide a platform for an analyst to understand the workings of COTS components, plugins, mobile code, and DLLs, as well as memory snapshots. CodeSurfer is a source code anaylzer.	2005
IDA Pro	Window/Linux excutables	DataRescue	A disassembler/debugger that can be used to analyze security issues in binary code.	31 Jan 2008
Logiscan	J2EE, MIPS and SPARC binaries, as well as existing Intel x86 support	LogicLab	Weaknesses such as buffer overflows, SQL injection and cross-site scripting can be discovered . It also offers suggestions for appropriate security remediation via its built-in training for secure coding. Formerly BugScan.	2005
SecurityReview	Excutable of *C, C++, C#, JAVA	Veracode	Automated static binary and dynamic web application analyses to identify software flaws and vulnerabilities, absence of security features, and malcode including backdoors and other unintended functionality. SecurityReview is a security testing service provided by Veracode.	24 May 2007
Vine	x86 executables	BitBlaze	Vine is a component of UC Berkeley�s research project BitBlaze. It provides an intermediate language (ILA) that x86 code can be translated to. It also provides analysis on the ILA, such as abstract interpretation, dependency analysis, and logical analysis via interfaces with theorem provers.	20 Jan 2008

Additional Guidance for Assurance

Advancing Technology for Humanity

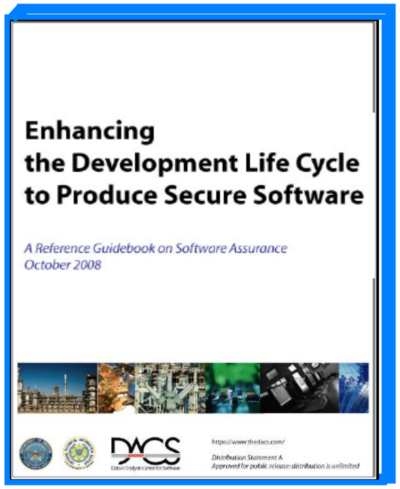
State of the Art Report on Software Security Assurance

- An IATAC/DACS report identifying and describing the current state of the art in software security assurance, including trends in:
  - Techniques for the production of secure software
  - Technologies that exist or are emerging to address the software security challenge
  - Current activities and organizations in government, industry, and academia, in the U.S. and abroad, that are devoted to systematic improvement of software security
  - Research trends worldwide that might improve the state of the art for software security
- Free via http://iac.dtic.mil/iatac/download/security.pdf



Enhancing the Development Life Cycle to Produce Secure Software

- Describes how to integrate security principles and practices in software development life cycle
- Addresses security requirements, secure design principles, secure coding, risk-based software security testing, and secure sustainment
- Provides guidance for selecting secure development methodologies, practices, and technologies



Enhancing
the Development Life Cycle
to Produce Secure Software

A Reference Guidebook on Software Assurance
October 2008

DACS
Data & Analysis Center for Software

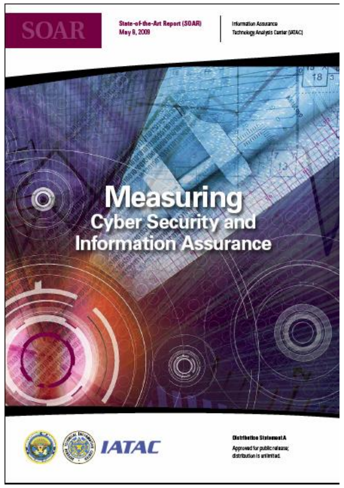
https://www.thedacs.com/

Distribution Statement A
Approved for public release: distribution is unlimited

Free via
https://www.thedacs.com/techs/enhanced_life_cycles/

Measuring Cyber Security and Information Assurance

- Provides a broad picture of the current state of cyber security and information assurance (CS/IA), as well as, a comprehensive look at the progress made in the CS/IA measurement discipline over the last nine years since IATAC published its IA Metrics Critical Review and Technology Assessment (CR/TA) Report in 2000


SOAR

State-of-the-Art Report (SOAR)
May 8, 2009

Information Assurance
Technology Analysis Center (IATAC)

Measuring
Cyber Security and
Information Assurance

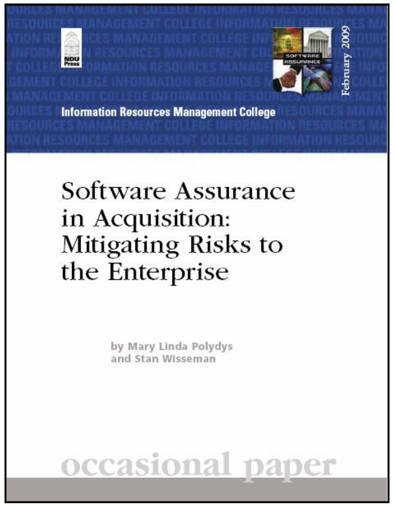
IATAC

Distribution Statement A
Approved for public release;
distribution is unlimited.

Available free via
http://iac.dtic.mil/iatac/download/cybersecurity.pdf

Software Assurance in Acquisition: Mitigating Risks to the Enterprise

- Provides information on how to incorporate Software Assurance considerations in key decisions
  - How to exercise due diligence throughout the acquisition process relative to potential risk exposures that could be introduced by the supply chain
  - Includes practices that enhance SwA in the purchasing process
    - Due diligence questionnaires designed to support risk mitigation efforts by eliciting information about the software supply chain (these are also provided in Word format so they can be customized)
    - Sample contract provisions
    - Sample language to include in statements of work
- Pre-publication version available free via
- https://buildsecurityin.us-cert.gov/swa/downloads/SwA_in_Acquisition_102208.pdf
- Final version published by National Defense University Press, Feb 2009



February 2009

Information Resources Management College

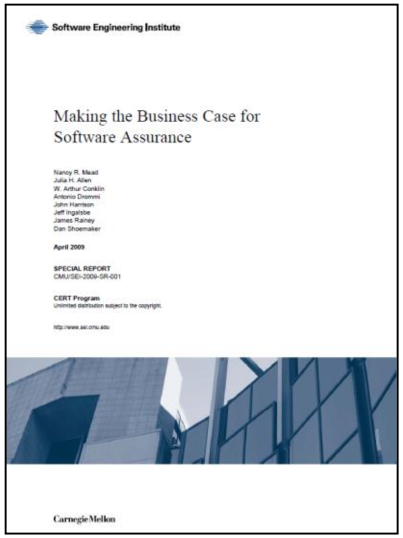
Software Assurance in Acquisition: Mitigating Risks to the Enterprise

by Mary Linda Polydys and Stan Wisseman

occasional paper

Making the Business Case for Software Assurance

- Provides background, context and examples for making the business case for software assurance:
  - Motivators
  - Cost/Benefit Models Overview
  - Measurement
  - Risk
  - Prioritization
  - Process Improvement & Secure Software
  - Globalization
  - Organizational Development
  - Case Studies and Examples
- Available free via
- http://www.sei.cmu.edu/library/abstracts/reports/09sr001.cfm



Software Engineering Institute

Making the Business Case for Software Assurance

Nancy R. Mead
Julia H. Allen
W. Arthur Conklin
Antonio Drommi
John Harrison
Jeff Ingalsbe
James Rainey
Dan Shoemaker

April 2009

SPECIAL REPORT
CMU/SEI-2009-SR-001

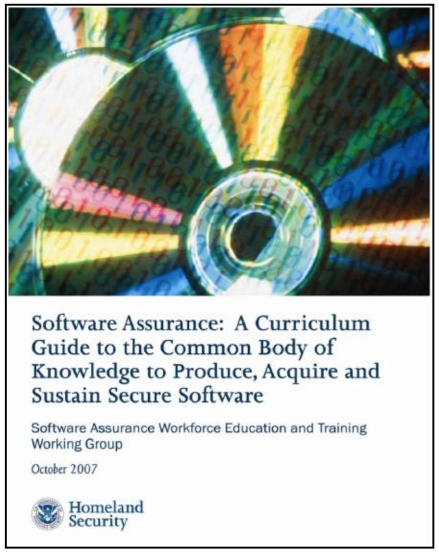
CERT Program
Unlimited distribution subject to the copyright.

http://www.sei.cmu.edu

Carnegie Mellon

Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software

- Provides a framework intended to identify workforce needs for competencies, leverage sound practices, and guide curriculum development for education and training relevant to software assurance
- Available via https://buildsecurityin.us-cert.gov/bsi/940-BSI/version/default/part/AttachmentData/data/CurriculumGuideToTheCBK.pdf


Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire and Sustain Secure Software

Software Assurance Workforce Education and Training Working Group

October 2007

Homeland Security

IEEE Advancing Technology for Humanity