Integration of Security Principles with Systems Engineering Activities

Jim Acerra, CISSP, C|CISO

Agenda

- Integrating IA/Security Engineers into the organization
- What is Information Assurance (IA)?
- CNSS IA Component Definitions
- The Perceived Conflict
- Engineering Disciplines Defined
- Why "Baked In" vs. "Bolted On"?
- Integrating Security within INCOSE Systems Engineering Activities
- Example Systems Engineering Approach with Security Integrated

Integrating IA Engineers Into the Organization

Dilbert, slightly modified to reflect the Information Assurance Engineer's plight

What is Information Assurance (IA)?

- According to the Committee on National Security Systems' IA Glossary, Information Assurance is:

  "Measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. These measures include providing for restoration of information systems by incorporating protection, detection, and reaction capabilities".

CNSS IA Component Definitions

- **Confidentiality:** The property that information is not disclosed to system entities (users, processes, devices) unless they have been authorized to access the information.
- **Integrity:** The property whereby an entity has not been modified in an unauthorized manner.
- **Availability:** The property of being accessible and useable upon demand by an authorized entity.
- **Non-Repudiation:** Assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the information.
- **Authentication:** The process of verifying the identity or other attributes claimed by or assumed of an entity (user, process, or device), or to verify the source and integrity of data.

The Perceived Conflict

- Systems Engineers (SEs) often perceive Information Assurance/Cyber Security Engineers as roadblocks to achieving a functional system
- Conversely, Systems Security Engineers (SSEs) frequently perceive that SEs do not care about system security
- Both positions are incorrect, as security is a critical functionality requirement for most systems, and system functionality is aligned to customer requirements and is essential for success of the system

SE and SSE are necessary to field a viable product

Engineering Disciplines Defined

Systems Engineering: is an engineering discipline responsible for creating and executing an interdisciplinary process that ensures the customer and all other stakeholder needs are satisfied in a high quality, trustworthy, cost efficient, and schedule-compliant manner throughout a system's entire life cycle (INCOSE, 2006).

Systems Security Engineering: is a specialty engineering field strongly related to systems engineering that applies scientific, engineering, and information assurance principles to deliver trustworthy security solutions that satisfy stakeholder requirements within their established risk tolerance (CNSS, 2010).

Why Baked In vs. Bolted On?

- Security is more effective if it integrated early on
- Security integration is more cost effective in development rather than adding it after the system is built
- Reduces the risk of having to re-engineer a system in order for it to meet security requirements
- Allows for threats and vulnerabilities to be factored into the engineering risk management activities for a project

"Until recently, security used to be considered as an afterthought, as an extra measure to protect company assets against threats or risks. This approach required security to be 'bolted-on.' Today, however, we are seeing a relatively more proactive approach, whereby enterprises are favorable to the idea of making security principles an integral aspect of enterprise architecture right from the outset. This approach requires security to be 'baked-in' to the enterprise architecture".

- Niranjan Prasad, Accenture

Integrating Security within INCOSE System Engineering Activities

Preparation

Establish the IA Testing Environment

- Select method of development
- Establish security touch points
- Define SE and SSE Organizations
- Design and implement repeatable security processes
- Acquire necessary security tools
- Train IA engineers how to use the tools
- Establish metrics and criteria for success
- Implement a test Bed for that Approximates the System's Intended Operational Environment

Preparation for security activities increase the likelihood of a secure system

INCOSE Systems Engineering Activities

- State the Problem
- Investigate Alternatives
- Model the System
- Integrate
- Launch the System
- Assess Performance
- Re-evaluate
- Variations

Each of these SE activities has a security counterpart or component

Security View of INCOSE Systems Engineering Activities

- State the Security Challenge
- Investigate Security Alternatives
- Model the System Security
- Integrate
- Launch the System Securely
- Assess Security Performance
- Re-evaluate (remediate and mitigate security flaws)
- Variations (Deviations in the security baseline that accommodates functionality and environmental conditions)

SE and SSE activities are integrated with each other

State the Security Challenge/Problem

- Identify the system's IA/Security boundary
- Articulate the threat to the system being developed
- Create an abstract of the system's security to prescribe and implement adequate and appropriate security requirements:
  - Determine presence of Critical Program Information/Technologies
  - Enclave (External Network)
  - Network (Internal Network)
  - Ports, Protocols and Services (PPS)
  - System Components and Sub-Components
    - Hardware, Software, and Firmware
    - Supply Chain Risk Management

System dissection allows for insertion of security measures at key points

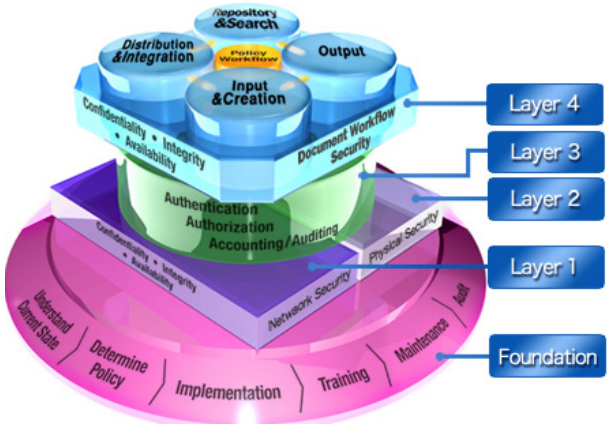
Investigate Security Alternatives

- Commercial-Off-The-Shelf (COTS) vs. Custom-developed security applications, mechanisms, and services
  - Product analysis
  - Cost benefit & suitability analysis of homegrown solutions
- Leverage upstream/lateral security (i.e., inherited and horizontal protection)
  - Is it effective to meet the system's needs?
  - Is it appropriate given the system's intended operation
- Outsource security
  - Completely?
  - Partially?
  - Not at all?

Reality is that in most cases, the result is a hybrid approach

Model the System's Security

- Begin with selection of appropriate framework applicable to the system. This serves as a foundation for your system's security and may require more than one framework. Here are a few examples:
  - DoD 8500 Series
  - COBIT
  - ITIL
  - NIST
  - ISO 27000 Series
  - OSSTMM



- Framework selection is impacted by these attributes:
  - Local, State, Federal, International Legislation
  - Proprietary Information (PI) concerns
  - Personally Identifiable Information (PII) and its associated liability issues
  - Industry specific (e.g., HIPPA, GLBA, PCI)
  - Culture and risk appetite of the organization

Framework selection impacts the system for its entire life cycle

Integration

- Does the designed and prescribed security measures break the system?
- Implement the system without harming or creating unacceptable risk to existing systems and networks
- Inherited and horizontal protection measures meet the new system's needs
- Can security be integrated in phases, allowing for layering of security measures based on cost, time, and staff?
- Conduct vulnerability assessments and penetration testing

Modular security allows for quicker isolation & troubleshooting

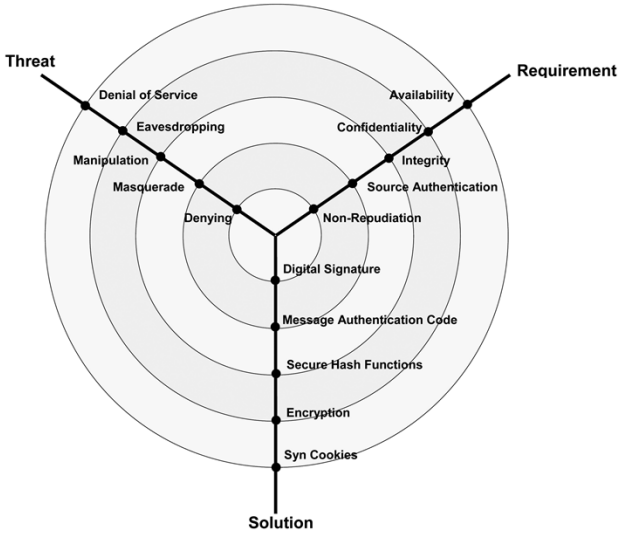
Launch Secure System (Securely)

- Have patches published since release been tested and applied to the system being fielded?
- Have final vulnerability scans been conducted to ensure that the system is still compliant?
- Are there any manufacturer workarounds that have been published since last review?
- Validate configuration of the operating systems, system services, network services, and internetworking devices as applicable
- Verify that the system does not negatively impact the security posture of interconnected systems

Attention to security during launch of a system prevents failure cascades

Assess Security Performance

- Validate security requirements have been met in the operational environment
- Security mechanisms receiving automated updates are functional and updating
- Conduct on-site vulnerability assessments and penetration testing
- Verify system redundancy (e.g., COOP)
- Ensure continuous monitoring and auditing of the new system
- Evaluate cyber hygiene regimen
- Remediate and mitigate residual vulnerabilities



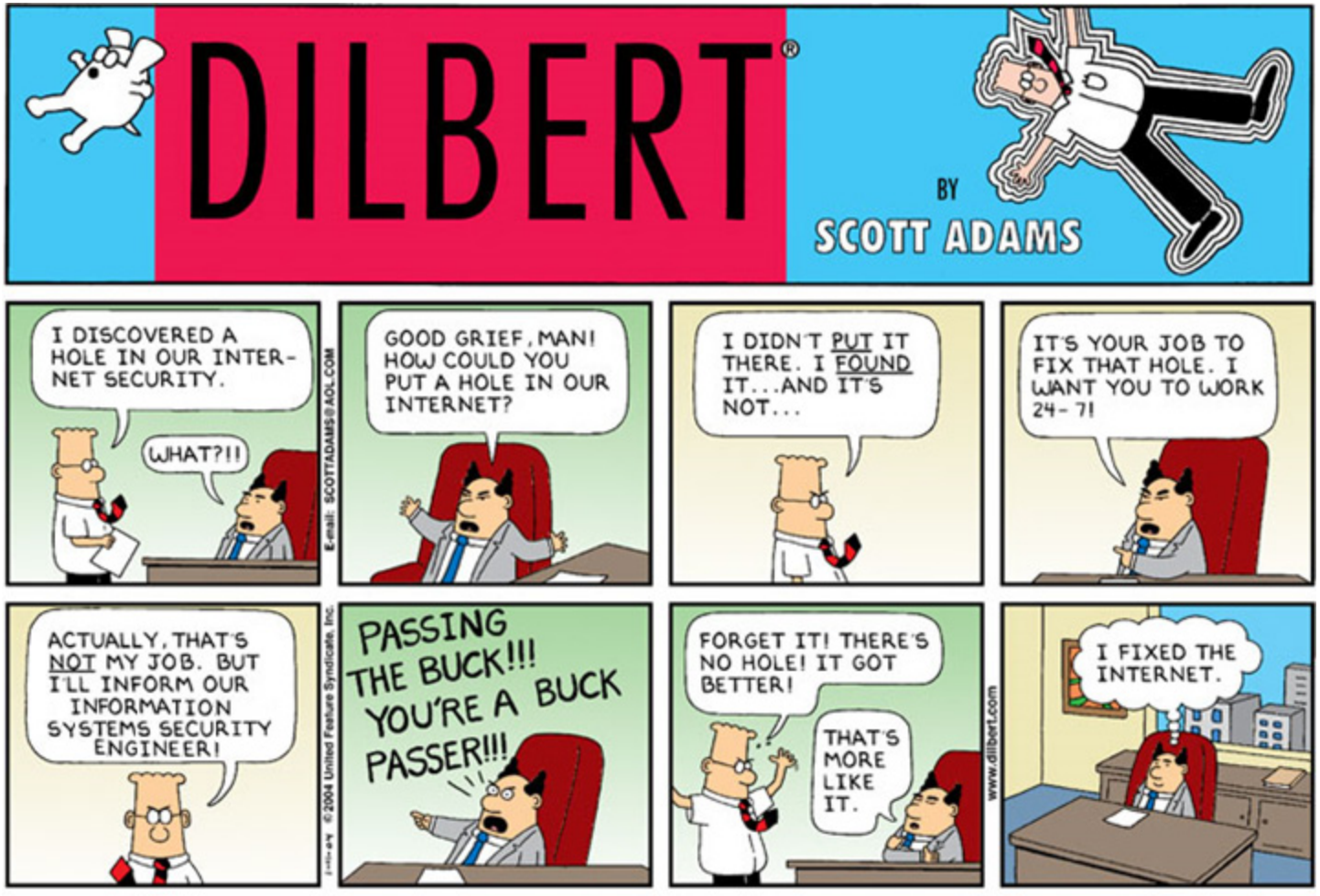
The repeatability of security processes and metric measurement is critical

Reassess Security & Baseline Variations

- Post-remediation and mitigation, does the system's security posture still meet requirements?
- Reapply security performance assessment processes to ensure stability between baselines
- Identify any requirements for deviations in the security baseline for a tailored security approach for the site
  - Environmental conditions
  - Physical security
  - Staffing
  - Maintenance availability
  - Differing user communities
  - Locale

Redundant checks help identify anomalies arising from fielding & remediation

Problem Solved?



Example Systems Engineering Approach with Security Integrated

Software Systems Engineering Using the Agile Method

Generic Agile Team with Security Embedded

- Product Owner
- Scrum Master
- Development Team Member (i.e., Systems Engineers, Programmers)
- Core Team Members
  - **SSE/IA Engineer** 

  - Quality Assurance (QA)
  - Configuration Management (CM)
  - Test & Evaluation (T&E)
  - Program Management (PM)
- Core Team Members are the last line of defense before a release candidate is advanced to external testing (i.e., outside of the scrum team)

High-Level ISSE/IA Engineer Scrum Duties

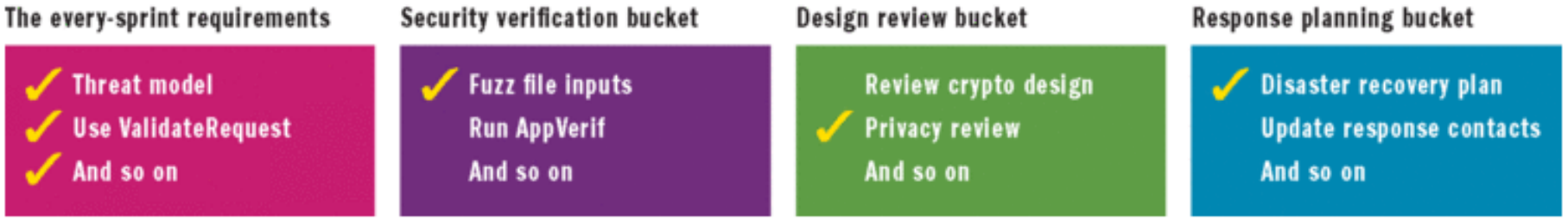
- Provides security requirements for the project
- Conducts security testing for each sprint
  - Vulnerability Scans
  - Penetration Testing (including Web, network, and system)
  - Source Code Security Analysis (i.e., Static and Dynamic)
  - Security/Vendor Patching
- Ensures the development team is following security mandates and leading industry practices in their development activities
- Remediates and mitigates project vulnerabilities
- Documents residual vulnerabilities
- Validates the pedigree of software repositories used by developers (i.e., supply chain risk management)
- Release Candidate (RC) source code signed

Microsoft Agile Security Development Life Cycle (SDL/A)

- Every Sprint Requirements: The first frequency level for SDL/A is the "every-sprint" level. These requirements are considered non-negotiable and must be completed every sprint regardless of how short the sprint length is. These requirements place the highest burden on the development teams since they're being completed so frequently, so these requirements are chosen very carefully according to two factors—importance and ease of automation. Example: Application threat modeling.
- Onboarding Requirements: The second group of requirements are referred to as "onboarding" requirements. These are requirements that product teams have to complete once at the beginning of the project and then never need to address again. Example: Configure the product's bug tracking system to track security-specific data.

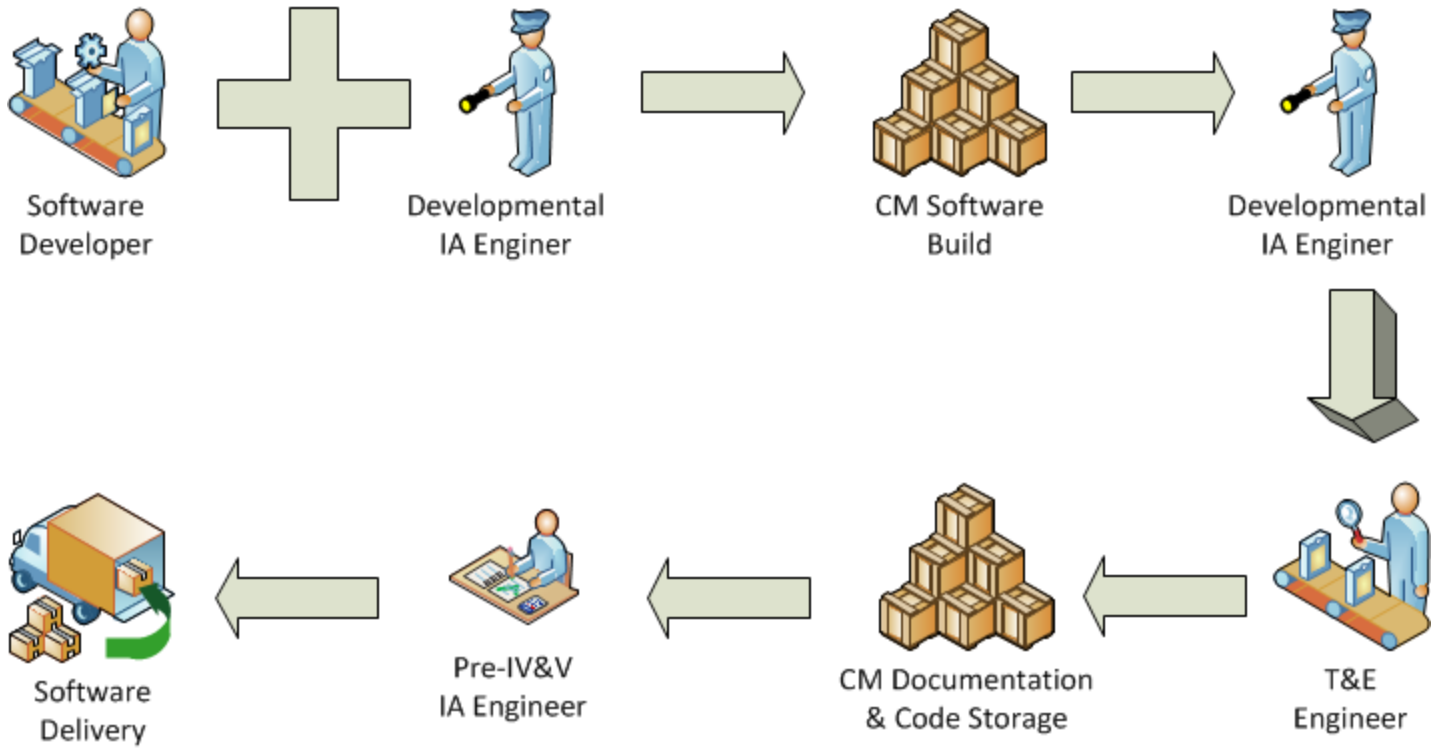
Microsoft Agile Security Development Life Cycle (SDL/A) - continued

- Bucket Requirements: All other SDL requirements that do not fall into either the every-sprint or onboarding requirement categories are placed into one of three requirement buckets: Security Verification, Design Review, and Response Plans. Unlike in classic SDL, where all of these requirements must be completed before the product can release, in SDL/A only one requirement from each bucket must be completed during each sprint. This is the concession that SDL/A makes to the shorter release schedules of Agile development projects.

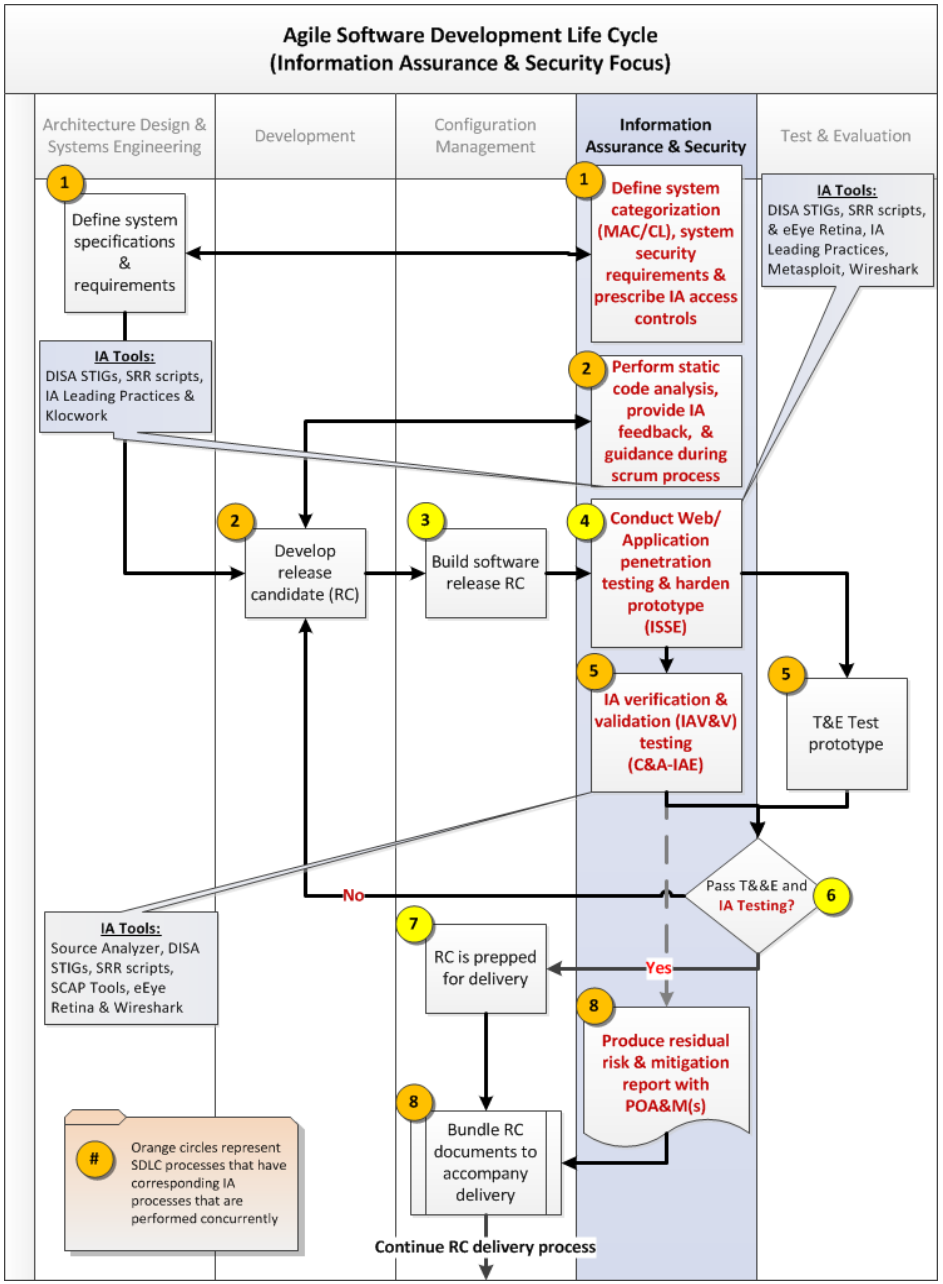


Microsoft SDL/A Life Cycle

Agile Method Basic Security Workflow



Each engineering sprint has iterative security applied and undergoes security testing in a manner that aligns with Agile practices



Summary

- SE and SSE are co-dependent disciplines when delivering and fielding secure and viable systems
- Security is cheaper and more effective when it is baked in and not bolted on
- While no process fits every situation, injecting IA rigor into a project's SE is possible
- Whatever IA approach is custom-fitted for your project, ensure that it is adequate, measurable, and repeatable
- Adoption of Agile principles has the potential to streamline security engineering and make it less painful for everyone
- Security tools, whether for hardware, software, or firmware should be varied to ensure no gaps in analysis

Questions?

Contact Information:

Jim Acerra, CISSP, C|CISO

jmacerra@outlook.com

619.300.9740

Back-Up Slides

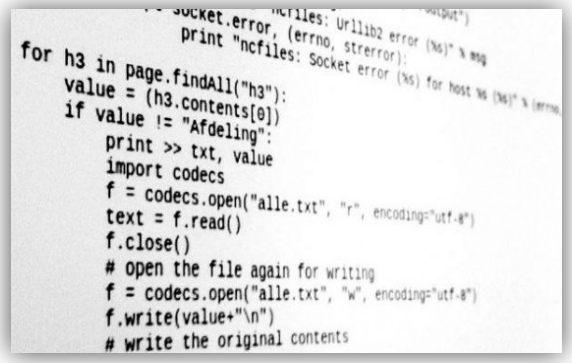
Software Security Engineering Tools

Static Source Code Analyzers: Static source code analyzers attempt to find code sequences that, when executed, could result in buffer overflows, resource leaks, or many other security and reliability problems. Source code analyzers are effective at locating a significant class of flaws that are not detected by compilers during standard builds and often go undetected during runtime testing as well.

Most static source code analyzers use the same type of compiler front end that is used to compile code. In fact, ideally, a static source code analyzer should be integrated with the everyday compiler to maximize use and reduce complexity of the tool chain. In addition, integrated checking enables source code parsing to be performed only once instead of twice. The use of a compiler front end is only natural because the analyzer takes advantage of preexisting compiler dataflow algorithms to perform its bug-finding mission.

Representative tools for static source code analysis:

- Fortify
- Coverity
- Klocwork
- Sonar



Software Security Engineering Tools (cont.)

Dynamic Source Code Analyzers: Dynamic analysis uses code instrumentation or a simulation environment to perform checks of the code as it executes. Some compilers have dynamic code analysis instrumentation available as a standard option. The development process should require that these checks be enabled at appropriate stages of development, testing, and integration.

The instrumented code performs the check and then calls a diagnostic function, provided by a library that is automatically linked to the program when using this option, which informs the user that a fault occurred as well as the type and location of the error within the source code.

Representative tools for dynamic source code analysis:

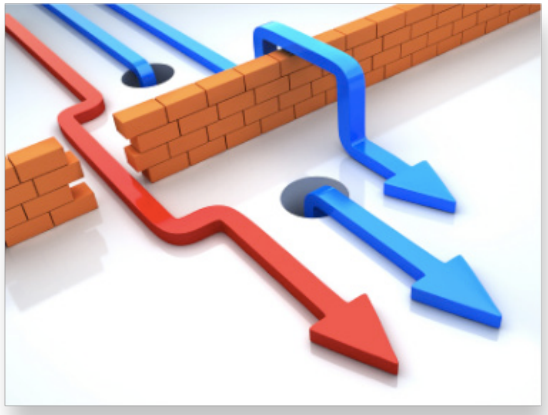
- Veracode
- Parasoft Insure++
- Klocwork

Software Security Engineering Tools (cont.)

System Penetration Testing: A penetration test, occasionally "pentest", is a method of evaluating computer and network security by simulating an attack on a computer system or network from external and internal threats. Pen-testing is implemented by simulating malicious attacks from an organization's internal and external users. The entire system is then analyzed for potential vulnerabilities. A plan that communicates test objectives, timetables and resources is developed prior to actual pen-testing.

Representative tools for system penetration testing:

- Saint
- Backtrack
- Nmap/Zenmap
- Metasploit



Software Security Engineering Tools (cont.)

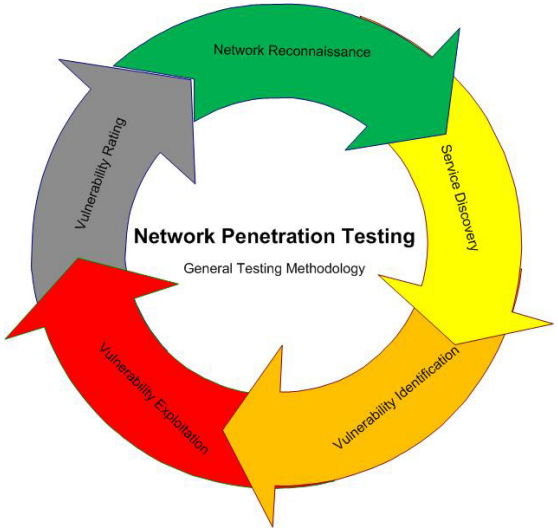
Web Application Penetration Testing: A Web application penetration test focuses on the security and potential risks present with a Web application. Using methods attackers use to infiltrate Web applications to obtain financial, personal and even medical information, these tests allow programmers to assess weaknesses in both server and client-side applications. Web application penetration tests can be performed manually or with automated software applications to identify possible security breach points, simulate the actual breach and report the final conclusion and resolutions of the test.

Representative tools for system penetration testing:

- Nikto
- Burp Suite
- w3af

Software Security Engineering Tools (cont.)

Network Penetration Testing: A method that evaluates the security of a network system by conducting an analysis and subsequent examination of potential gaps or "holes" within security operations, flaws within hardware or software configuration or other operational weaknesses that may exist. Once completed these tests expose security vulnerabilities, assess the impact should a potential security threat occur and follow with a proposal of a technical solution. The outcome of these tests allows businesses to make sound security decisions under certain circumstances such as setting up a new office, deployment of a new network infrastructure or upgrading existing or new applications.



Representative tools for network penetration testing:

- Wireshark
- Ettercap
- Tcpdump

Software Security Engineering Tools (cont.)

Vulnerability Scanners: A vulnerability scanner is a computer program designed to assess computers, computer systems, networks or applications for weaknesses. There are a number of types of vulnerability scanners available today, distinguished from one another by a focus on particular targets. While functionality varies between different types of vulnerability scanners, they share a common, core purpose of enumerating the vulnerabilities present in one or more targets. Vulnerability scanners are a core technology component of vulnerability management.

Vulnerability scanners are tools that generally perform the following:

- Cataloging assets and capabilities (resources) in a system.
- Assigning quantifiable value (or at least rank order) and importance to those resources
- Identifying the vulnerabilities or potential threats to each resource
- Mitigating or eliminating the most serious vulnerabilities for the most valuable resources

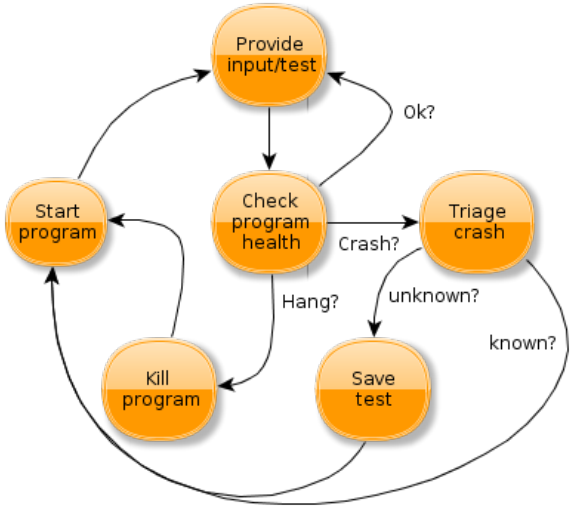
Representative tools for vulnerability scanning:

- eEye Retina
- NESSUS
- OpenVAS

Software Security Engineering Tools (cont.)

Fuzz Testing: Fuzz testing is often employed as a black-box testing methodology in large software projects where a budget exists to develop test tools. Fuzz testing is one of the techniques that offers a high benefit-to-cost ratio[1]. The technique can only provide a random sample of the system's behavior, and in many cases passing a fuzz test may only demonstrate that a piece of software can handle exceptions without crashing, rather than behaving correctly. This means fuzz testing is an assurance of overall quality, rather than a bug-finding tool, and not a substitute for exhaustive testing or formal methods. Representative tools for network penetration testing:

- Fuzz
- Peach Fuzzer Platform
- Untidy



1. Source: Justin E. Forrester and Barton P. Miller. "An Empirical Study of the Robustness of Windows NT Applications Using Random Testing".