Naval Ordnance Safety & Security Activity Software Security Assessment Tools Review

Safety-critical systems

- Main functions: monitoring, diagnosis, control of physical systems
- Extraordinary conditions = hazards (accidents)
- Consequences: potentially catastrophic, even fatal
- Examples:
  - Embedded: onboard vehicle controllers/computers, medical devices, process controllers, robots
  - Non-embedded: SCADA and DCS, air traffic control systems, telematic monitoring/diagnostic/control (OnStar, etc.)
  - Hybrid: weapons systems

Security critical systems

- Main functions:
  - sensitive/privacy information processing, transmission, storage;
  - network communications;
  - security (detection, protection, response) for data, software, networks, physical facilities
- Extraordinary conditions = threats (attacks/ exploits; errors with exploitable results)
- Consequences: depend on nature of purpose, users, data, resources
- Examples:
  - Embedded: network controllers, facility security sensors/alarm systems, Trusted Platform Modules (TPMs)
  - Non-embedded: operating system kernels and file systems, virtual machine monitors, information systems/applications, communications systems/applications, computer and network security systems and sensors
  - Hybrid: networking devices, security appliances, cryptographic devices, ATM machines

Using tools to begin integrating safety and security

Terminology

- Extraordinary condition = Any condition deviating from those under which software is designed to operate

Divergences

- What constitutes an extraordinary condition
- What is at stake if software fails due to an extraordinary condition
- Level of tolerance for failure

Convergence

- Shared need for software to remain dependable under extraordinary conditions

Security of safety-critical software

Must be addressed at three levels

1. Functional
   - ↗ threats to software's own availability (denial of service) and integrity (corruption, tampering, malicious code)
2. Data
   - ↗ threats to integrity of inputs, outputs (tampering, substitution, rerouting, deletion, malicious code insertion, disclosure)
   - ↗ threats to information processed, stored, transmitted (same as inputs/outputs)
3. Execution environment
   - ↗ threats to availability, integrity of environment components
   - ↗ threat of resource theft

Security of safety-critical software cont'd

System vs. software level security

- System: focus is on
  - external interfaces/interactions
    - between system components
    - between system and other systems
    - between system and users

- Software: focus is on
  - internal workings *and*
  - external interfaces/interactions
    - between software components/units/modules
    - between software and execution environment
    - between software and users or process

Tool Categories

- The categories of tools evaluated and detailed in this paper are:
- Static Analysis
- Source Code Fault Injection
- Dynamic Analysis
- Architectural Analysis
- Pedigree Analysis
- Binary Code Analysis
- Disassembler Analysis
- Binary Fault Injection
- Fuzzing
- Malicious Code Detectors
- Bytecode Analysis

Methodology

- Open source information
  - High-level vendor-provided data
- Which stage of the software development life cycle targets
  - Discussion of additional stages if applicable
- Required skills
  - Higher maturity level of tools indicates less of a reqiured skillset
- Benefits and drawbacks

Summary of Evaluation **Key:** X*- To be most beneficial X+ - In some cases X% - When possible X# - e.g., compilation	Static Analysis Code Scanning	Source Code Fault Injection	Dynamic Analysis	Architectural Analysis	Pedigree Analysis	Binary Code Analysis	Disassembler Analysis	Binary Fault Injection	Fuzzing	Malicious Code Detector	Byte Code Analysis
When to Use											
Requirements				X							
Design				X*							
Implementation	X*	X	X	X	X*	X	X	X	X	X	X*
Testing	X	X*	X*	X	X	X	X	X*	X*	X	X
Production	X	X	X	X	X	X	X	X	X	X	X
Acquisition	X	X	X	X	X	X*	X*	X	X	X*	X
Required Skills (Understanding of…)											
Underlying source code		X									
Underlying development methodology			X#	X					X		
Implementing language	X	X					X%				
Binary						X+	X				
Bytecode											X
Testing methodology		X	X	X			X		X	X	X

Summary of Evaluation **Key:** X*- To be most beneficial X+ - In some cases X% - When possible X# - e.g., compilation	Static Analysis Code Scanning	Source Code Fault Injection	Dynamic Analysis	Architectural Analysis	Pedigree Analysis	Binary Code Analysis	Disassembler Analysis	Binary Fault Injection	Fuzzing	Malicious Code Detector	Byte Code Analysis
Benefits											
Reduces cost over system life	X	X	X	X	X	X	X	X	X	X	X
Educates developers about secure programming	X			X	X						
Rechecks legacy code	X				X	X	X	X	X		X
Automates repetitive and tedious aspects of source code security audits	X				X						
Checks for good programming style	X										X
Increased test coverage		X						X	X		
Increased accuracy		X							X		
No need for source code			X	X		X	X	X	X	X	X
Improved accuracy and coverage		X	X					X	X		
Reduces the amount of testing necessary					X						
No disassembly	X	X	X	X	X	X		X	X		
Guaranteed the analysis is performed on the actual product			X+	X		X	X	X	X	X	X

Summary of Evaluation **Key:** X*- To be most beneficial X+ - In some cases X% - When possible X# - e.g., compilation	Static Analysis Code Scanning	Source Code Fault Injection	Dynamic Analysis	Architectural Analysis	Pedigree Analysis	Binary Code Analysis	Disassembler Analysis	Binary Fault Injection	Fuzzing	Malicious Code Detector	Byte Code Analysis
Drawbacks											
No architectural-level flaws	X		X		X	X	X			X	X
Thorough understanding of the software			X	X				X	X		
Required expertise		X		X			X	X	X		
Requires use of open source software					X						
Lack of tool availability		X				X				X	
Licensing concerns							X				
Reliance on a primary vendor							X				
Additional analysis		X		X			X	X	X		X
Additional preparation		X	X	X					X		
Limited to a single language											X

Further Work

- The Tools Report is available at:
  - https://buildsecurityin.us-cert.gov/swa/procwg.html
- Safety and Security Considerations for Component-Based Engineering of Software-Intensive Systems
  - https://buildsecurityin.us-cert.gov/swa/procwg.html
- In-depth review of specific security tools
  - Review the capabilities of tools
  - Review their applicability to the safety community
- Safety and Security Test Toolkit

QUESTIONS?

- Available for public comment at:
  - https://buildsecurityin.us-cert.gov/swa/procwg.html