19701
Leveraging Cybersecurity Tools For Software Safety

Focusing (Some) Static Analysis on Safety-Critical Software

Stuart A. Whitford
Booz Allen Hamilton
20th Annual NDIA Systems Engineering Conference
Springfield, VA
25 October 2017

Agenda

- Some Givens
- Safety versus Security
- General Static Analysis: Dealing with false positives and false negatives
- Targeted Static Analysis: Proving specific properties and assertions
- Coordinating the Efforts
- Conclusion

NOTE: Blue highlighting in this presentation is for ***emphasis.***

Some Givens

[C]ybersecurity applies to weapons systems . . . [and] is a critical priority for the DoD. . . incorporate code reviews and architecture reviews against incremental builds to reduce vulnerabilities in any custom software, including via automated scanning tools (e.g., static analysis).

[*The DoD Program Manager's Guidebook for Integrating the Cybersecurity Risk Management Framework (RMF) into the System Acquisition Lifecycle*, September 2015]

DoD will continue to assess Defense Federal Acquisition Regulation Supplement (DFARS) rules . . . to ensure they mature . . . in a manner consistent with known standards for protecting data from cyber adversaries, to include standards . . . by the National Institute of Standards and Technology (NIST).

[*The Department of Defense Cyber Strategy*, April 2015]

More Givens

Source code should be periodically reviewed using automated tools or manual spot check for common programming errors . . . as part of the software development QA process.

[NIST Special Publication 800-64 revision 2, *Security Considerations in the System Development Life Cycle*, October 2008]

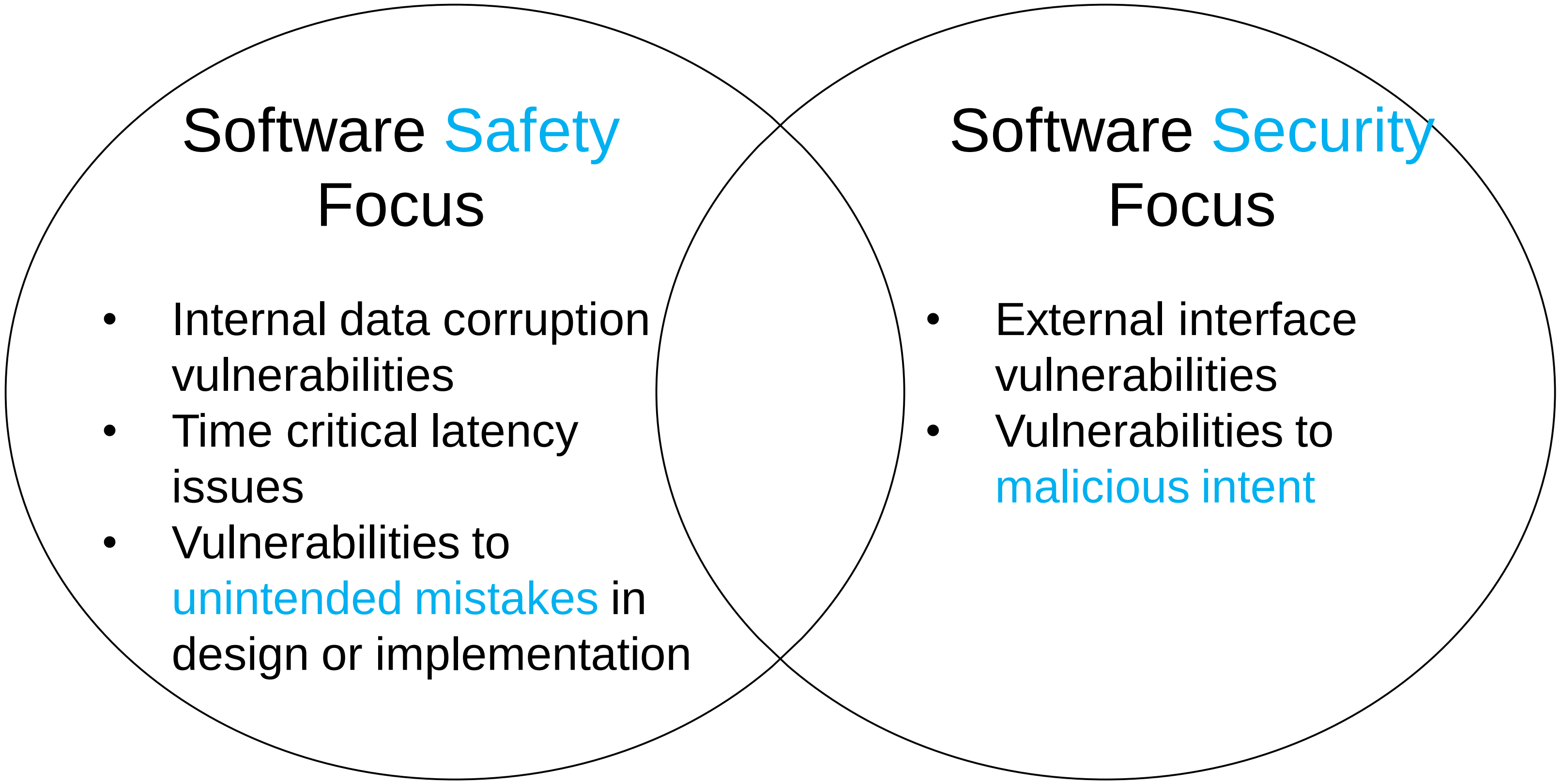
The Program Manager will integrate ESOH risk management into the overall systems engineering process for all engineering activities throughout the system's life cycle. . . The Program Manager will use the methodology in MIL-STD-882E.

[DoD Instruction 5000.02, "Operation of the Defense Acquisition System," January 7, 2015]

Level of Rigor Tasks [for Software Criticality Index (SwCI) 1/highest] . . . Program shall perform analysis of requirements, architecture, design, and code; and conduct in-depth safety-specific testing.

[MIL-STD-882E, "DoD Standard Practice for System Safety," May 11, 2012]

Software Safety versus Software Security



Software Safety Focus

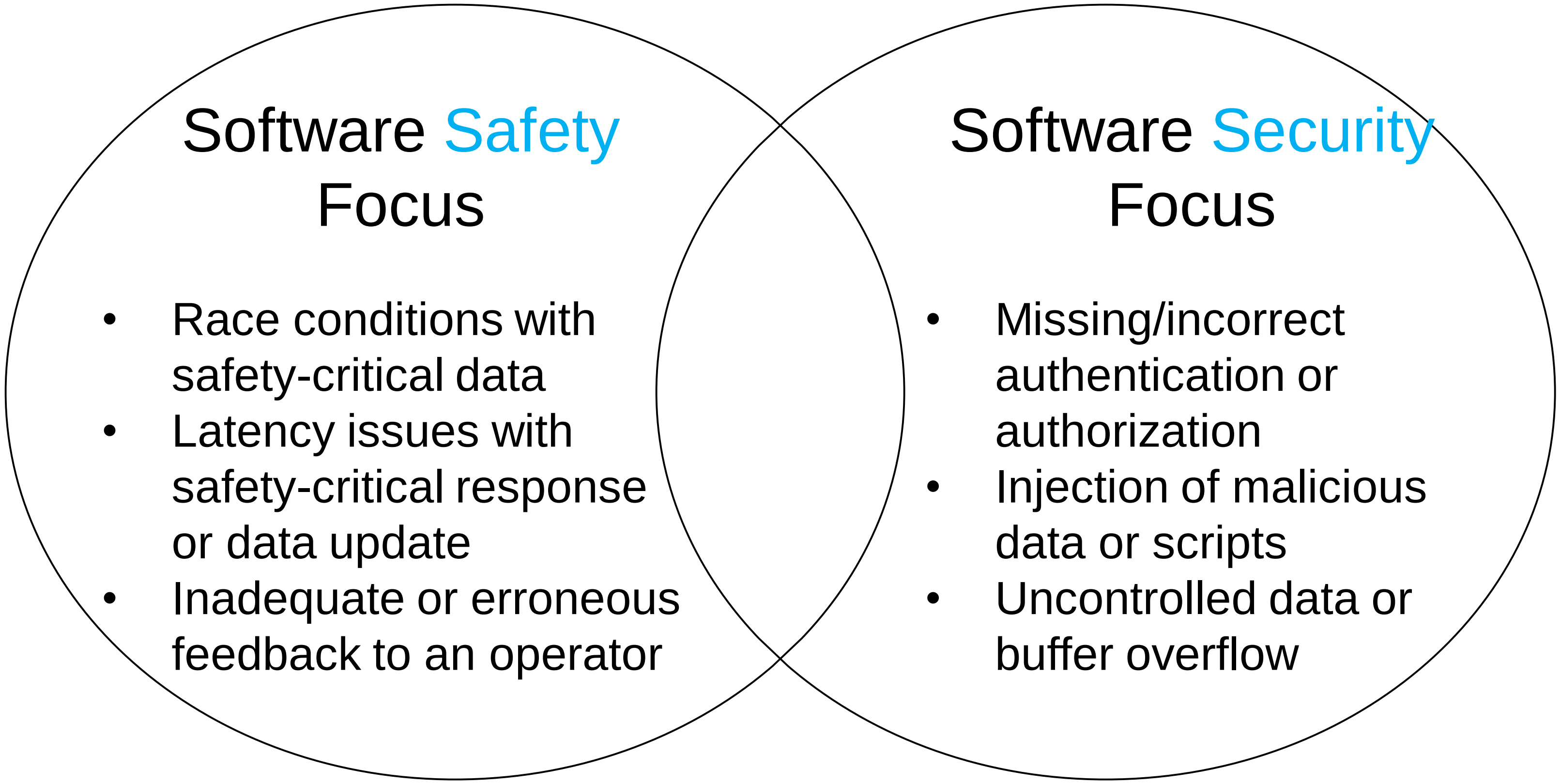
- Internal data corruption vulnerabilities
- Time critical latency issues
- Vulnerabilities to unintended mistakes in design or implementation

Software Security Focus

- External interface vulnerabilities
- Vulnerabilities to malicious intent

There is some overlap, but the priorities and focus are different.

Software Safety versus Software Security



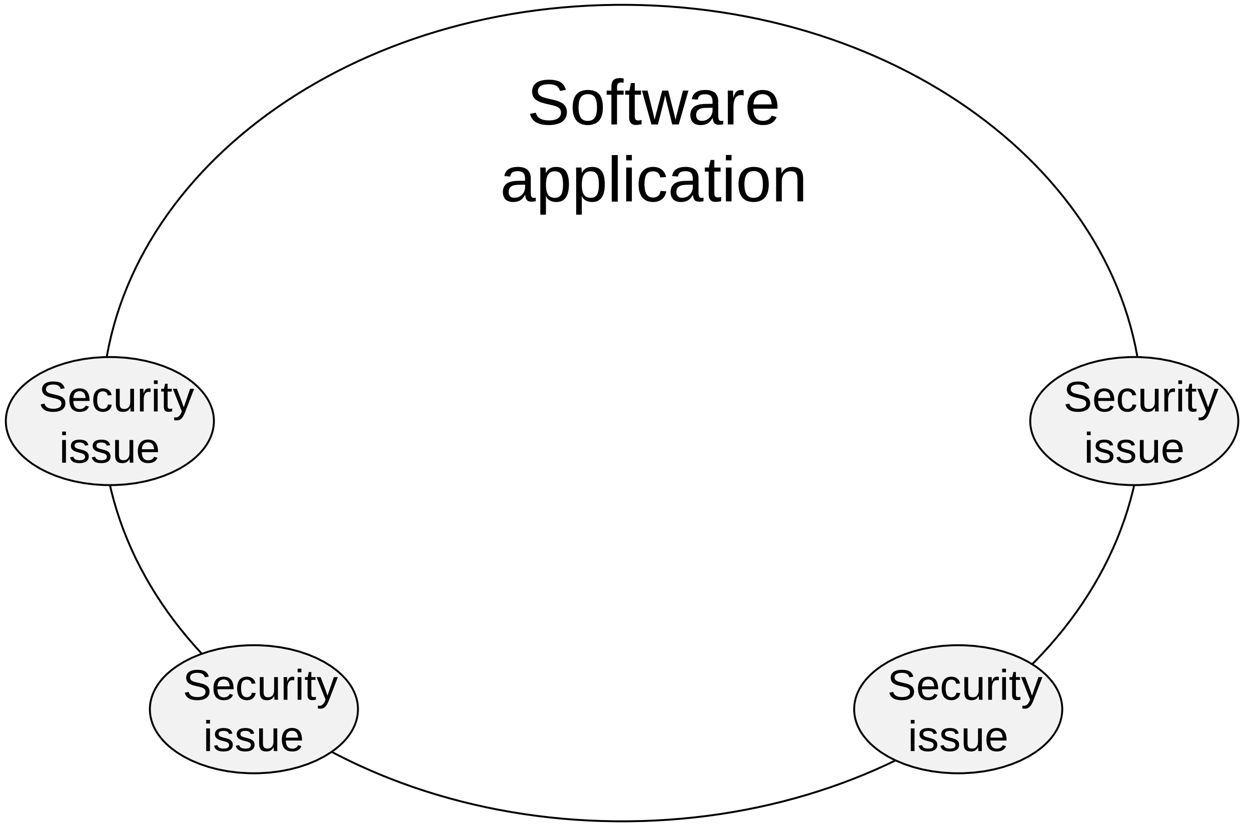
Software Safety Focus

- Race conditions with safety-critical data
- Latency issues with safety-critical response or data update
- Inadequate or erroneous feedback to an operator

Software Security Focus

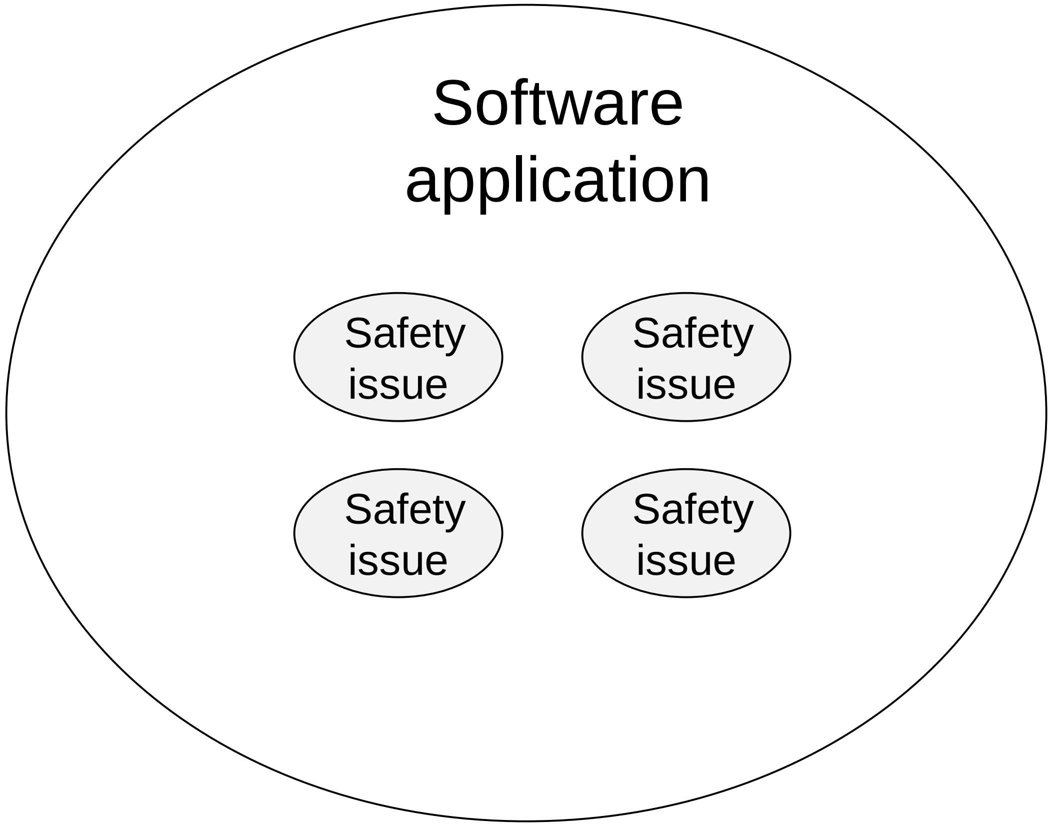
- Missing/incorrect authentication or authorization
- Injection of malicious data or scripts
- Uncontrolled data or buffer overflow

Software Safety versus Software Security



Security issues tend to be at the external interfaces of a software application.

Software Safety versus Software Security



Safety issues tend to be in the core system functionality of a software application.

General Static Analysis: Dealing with *false positives* and *false negatives*

Booz | Allen | Hamilton

General Static Analysis

- General static source code analysis
  - – Flagging programming errors
    - MITRE’s Common Weakness Enumeration (CWE)
    - False positives and false negatives

- Targeted static analysis
  - – Proving targeted assertions
  - – Counter examples
  - – Program slicing

General Static Source Code Analysis

- Flagging programming errors
  - MITRE's Common Weakness Enumeration (CWE)
  - Security CWE's
    - Open Web Application Security Project (OWASP) Top 10 CWE's
      - Injection / Broken Authentication / Cross-site Scripting / Insecure Direct Object References / Security Misconfiguration / etc.
  - Safety CWE's
    - Data corruption CWE's
      - Shared resource race condition / Buffer Overflow / Improper Validation of an Array Index / Pointer Issues / Incorrect Type Conversion / etc.

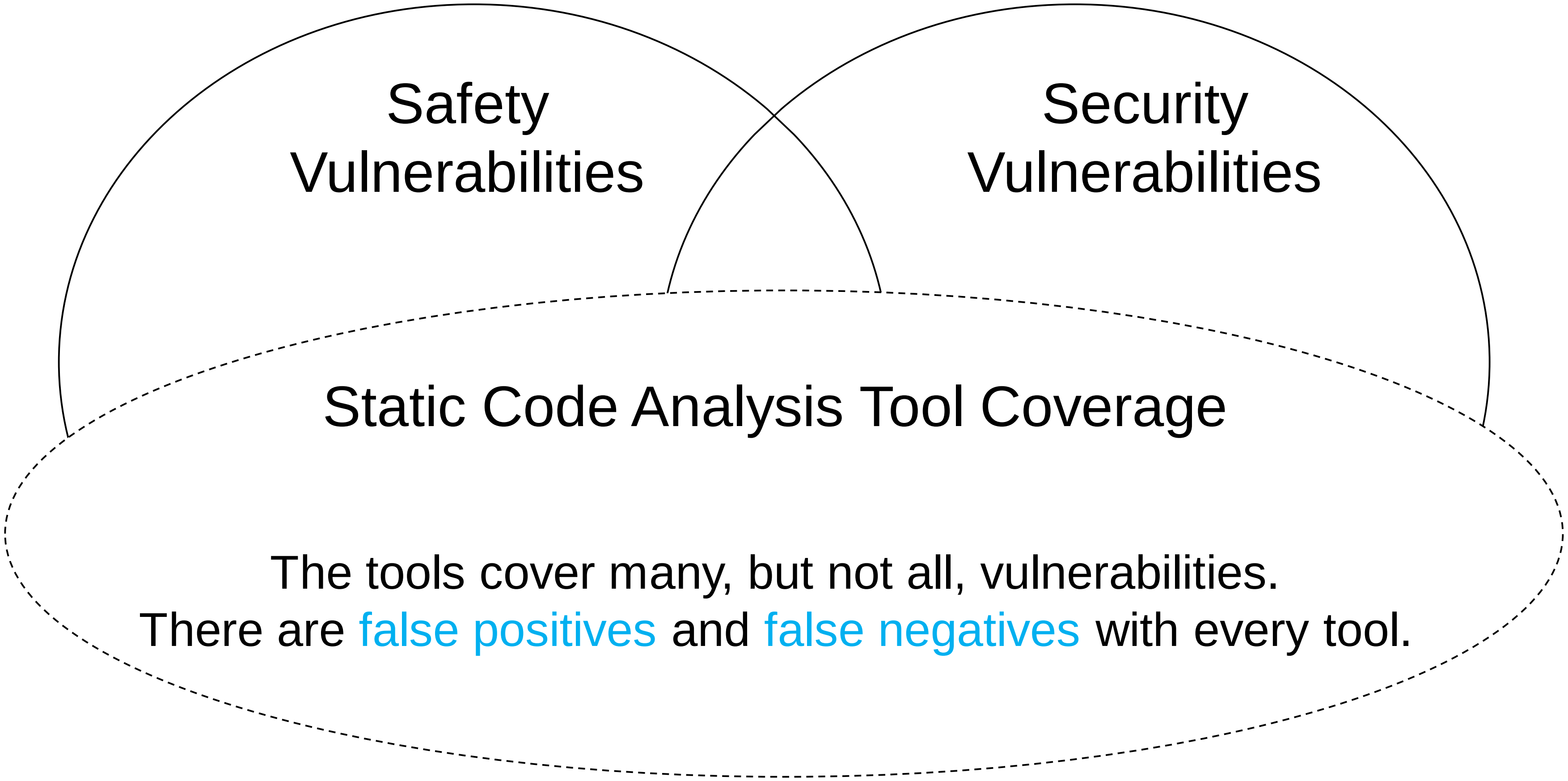
Safety Critical Data ‘Corruption’

A correctly implemented algorithm operating on corrupted or stale safety-critical data can have unintended catastrophic results.

Some sources of corrupted data:

- Noise in digital message transmission
- Physical events/upsets during data storage
- Multi-threaded shared data
- Shared data between ‘main’ and Interrupt Service Routines
- Caching of data
- Loss of transient status data in failover or ‘recovery’

General Static Code Analysis



The tools cover many, but not all, vulnerabilities.
There are false positives and false negatives with every tool.

The Opportunity for Software Safety

- Many of the programming errors detected by software static analysis tools used for cybersecurity have potential safety-critical impacts:
  - Multi-threaded race conditions
  - Mishandling of pointers
  - Incorrect casting (data type conversion)
  - Buffer overflow

- Providing access to general static analysis tools already being used for cybersecurity could greatly assist those responsible for software safety design and code analysis.
  - Need communication and coordination of effort between those responsible for security and those responsible for system safety

Static analysis tools are already in use for safety

- Food and Drug Administration (FDA):

  . . . static analysis examines the code exhaustively for certain kinds of insidious errors that are hard for human reviewers to detect.

  [http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/GeneralHospitalDevicesandSupplies/InfusionPumps/ucm202511.htm#staticAnalysis]

- Federal Aviation Administration (FAA):

  A combination of both static and dynamic analyses should be specified by the applicant/developer and applied to the software.

  [Certification Authorities Software Team (CAST) Position Paper CAST-9, January 2002]

- Motor Industry Software Reliability Association (MISRA):

  Compliance with MISRA C/C++ coding standards for safety-critical software is checked by many static analysis tools.

Some General Static Source Code Analysis Tools

- Flagging programming errors
  - Grammatech’s CodeSonar
  - Coverity’s Code Advisor
  - IBM’s AppScan
  - Clang Static Analyzer
  - CppCheck
  - Parasoft’s Static Analysis Engine
  - Redlizard’s Goanna
  - Checkmarx’s CxSAST
  - Fasoo’s Sparrow

Targeted Static Analysis:

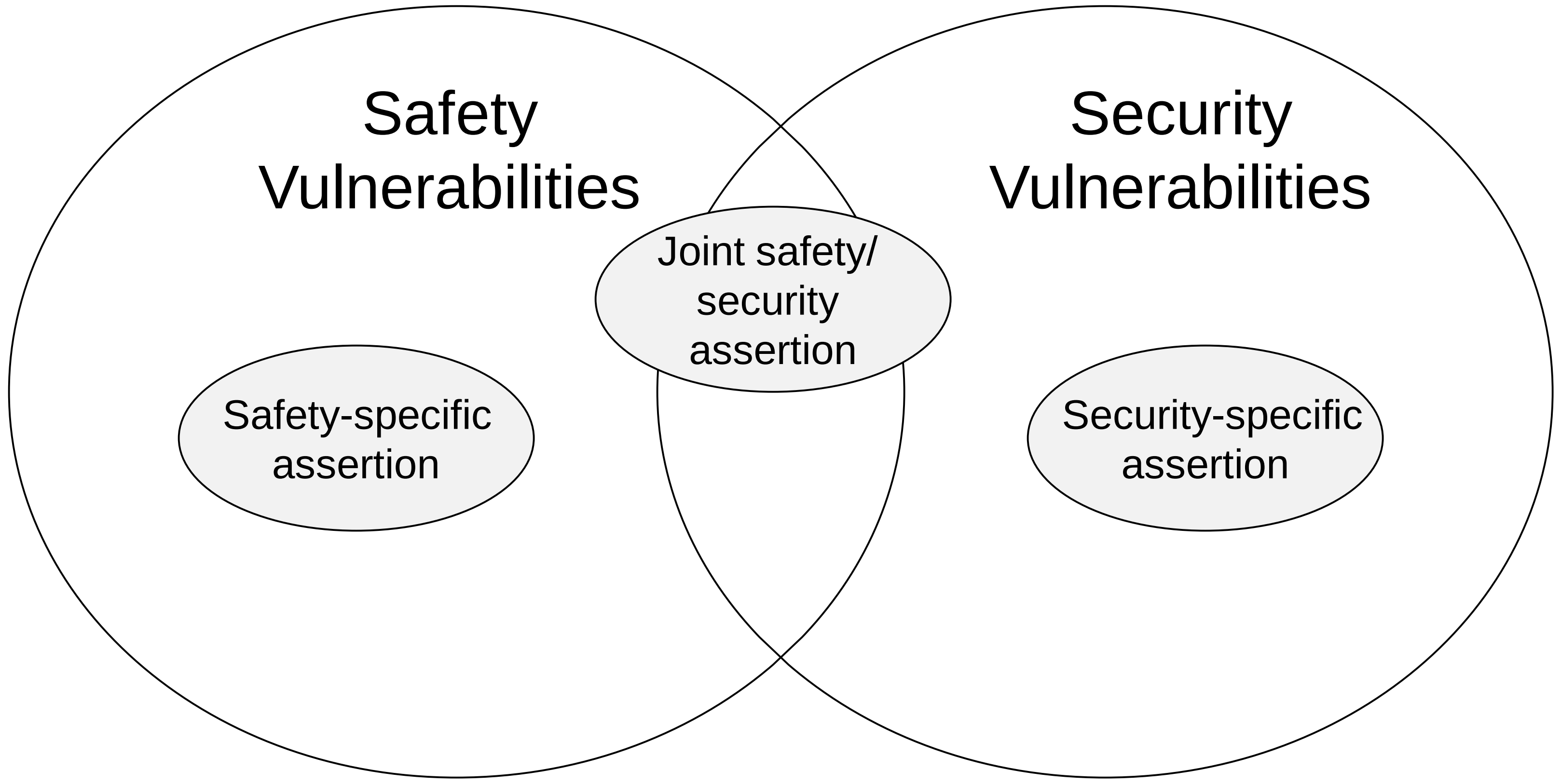
Proving *specific properties* and *assertions*

Booz | Allen | Hamilton

Targeted Static Analysis

- Targeted static analysis
  - Proving targeted assertions
  - Counter examples
  - Program slicing

Targeted Static Analysis
Abstract Interpretation/Model Checking


Safety Vulnerabilities
Security Vulnerabilities
Joint safety/ security assertion
Safety-specific assertion
Security-specific assertion

"Prove" application-specific assertions hold true
for any possible execution sequence (absence of *specific* vulnerabilities).

Soundness vs. Completeness

"[T]he essence of [abstract] static analysis is to efficiently compute approximate but sound guarantees: guarantees that are not misleading. . . . Due to the undecidability of static analysis problems, devising a procedure that does not produce spurious warnings and does not miss bugs is not possible."

["A Survey of Automated Techniques for Formal Software Verification" D'Silva, et al. IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 27, NO. 7, JULY 2008]

Soundness means that, if the tool reports a property or assertion is met, the tool can be trusted.

Undecidability means that the tool might not be able to decide for every possible property or assertion (it is "incomplete").

Programming constraints to enable sound static analysis

- Specialized programing or modeling languages
  - Esterel/Lustre
  - Signal
  - Promela (for formal analysis by SPIN)
- Language subsets
  - Escher C Verifier (verifies programs written in an annotated C subset)
  - KeY (verifies properties of programs written in a Java subset)
  - VeriFast (verifies programs written in Java or C subsets)

Safety-Critical Decision Points

- Safety-critical software has command authority over potentially dangerous system actions.
- The software is therefore responsible for making the decision to take that action.
- If the data used to make the decision is corrupted or stale, the software can make the wrong decision with catastrophic results.
- Design and code analysis of the software should be focused on the integrity of the data used at each Safety-Critical Decision Point in the software.

Programming slicing

In computer programming, program slicing is the computation of the set of programs statements, the program slice, that may affect the values at some point of interest, referred to as a slicing criterion. Program slicing can be used in debugging to locate source of errors more easily. Other applications of slicing include software maintenance, optimization, program analysis, and information flow control.

[*Wikipedia* article on “Program Slicing,” March 17, 2015]

Some Targeted Static Analysis Tools

- Proving targeted assertions (model checking)
  - Bell Lab's SPIN
  - Carnegie Mellon's NuSMV
  - Kestrel's CodeHawk (abstract interpretation)
  - MathWork's Polyspace Code Prover (abstract interpretation)
  - Microsoft-Inria TLA+ Proof System (TLAPS)
- Program slicing tools
  - VALSOFT/Joana
  - GrammaTech's CodeSurfer

Opportunities for software security/safety collaboration

[A]ll systems should be developed as safe secure systems. . . to allow for a complementary software skill set in software development (tools and language dependent). This would require a common development process rather than a skill change. . . [R]isk and hazard analysis, for both a security and safety assessment, should be conducted and therefore requires skills from both arenas . . . Independence of this skill . . . may be required though to ensure there is no bias towards contradicting risks.

["Safety-Critical Versus Security-Critical Software." Dr. Adele-Louise Carter, Version 1.0, August 2010, bcs.org.uk]

Questions?

Stuart Whitford
Senior Lead Scientist

Booz | Allen | Hamilton

Booz Allen Hamilton
1550 Crystal Dr, Suite 1100
Arlington, VA 22202
Tel (540) 903-7035
whitford_stuart@bah.com

Backup Slides

Tools to Support Software Safety Analysis

Use tools to help analyze the Safety-Significant Software in the context of the Architecture, Design, or Code (leverage those in use by the software developers or obtain):

- Software architecture and design modeling and analysis tools, such as those supporting Architecture Analysis and Design Language (AADL), Unified Model Language (UML), or Systems Modeling Language (SysML)
- Static code analysis tools that support focused design and code analyses, such as thread race/deadlock detection or program slicing
- Source code cross reference tools that support searching, cross-referencing, and navigating (forward and backward) source code trees

Some References

- Joint Software Systems Safety Engineering Workgroup. (2010). Joint Software System Safety Engineering Handbook (JSSSEH). Indian Head, MD: Naval Ordnance Safety and Security Activity.
- Anton, J. et al (2005). “Towards the Industrial Scale Development of Custom Static Analyzers.”
- NSA Center for Assured Software (2011). “On Analyzing Static Analysis Tools.”
- NIST Special Publication 800-176 (2014). *Computer Security Division Annual Report 2014*.
- Garavel, H., ed. (2013). *Formal Methods for Safe and Secure Computer Systems*. BSI Study 875.
- Carter, A. (2010). *Safety-Critical Versus Security-Critical Software*.
- Moy, Y. (2014). “Static Analysis Tools Pass the Quals.” *CrossTalk*. November/December, 2014.

Booz | Allen | Hamilton