HACMS : Formal methods are practical and effective for building and verifying high-assurance aerospace systems
The goal of CASE is to develop the necessary design, analysis and verification tools to allow system engineers to design-in cyber resiliency and manage tradeoffs as they do the other non-functional properties when developing complex embedded computing systems.

- Cyber resiliency means that the system is tolerant to cyberattacks in the same way that safety critical systems are tolerant to random faults – they recover and continue to execute their mission function.
- Cyber security requirements are addressed today by penetration testing late in the development, resulting in expensive rework.
- Cyber requirements are often “shall not” statements about the system, and so are not testable (formal methods required).
CASE TARGETS

• Phase 2 Experimental platform: AFRL UxAS

• Phase 3 Demonstration platform: CH-47 Common Avionics Architecture System (CAAS)
APPROACH

• Start with initial design, new or legacy
  • Federated avionics system
• Generate new cyber requirements
  • Possibly based on modified system architecture
• Tool-assisted transformations of system architecture
  • Satisfy cyber requirements
  • Manage other design trade-offs
  • Insertion/synthesis of high-assurance components may be needed
  • Insertion of verification infrastructure
• Verification of cyber resiliency
• Generate system from architecture model
ARCHITECTURE ANALYSIS AND DESIGN LANGUAGE (AADL)

- SAE AS5506 standard
- Embedded, real-time, distributed systems
- Physical hardware
  - processors, buses, memory, devices
- Application software
  - software functions, data, threads, processes
- Extendable syntax (annex)
- Open source tools, supported by SEI
  - Open Source AADL Tool Environment (OSATE)

- Sufficiently rigorous semantics to support analysis
- Correct level of abstraction (supports construction)
- Syntax allows addition of new capabilities
Tools generate cyber requirements based on initial system model (AADL) and (possibly) functional requirements (CRA & VU/Doll). User evaluation is still needed to determine applicability and add additional details. Assurance case provides a mechanism to receive, implement, and manage cyber requirements and attach them to relevant parts of the design model. Allows us to specify exactly what evidence is necessary to satisfy each cyber requirement. Detect if subsequent design changes invalidate evidence.

```
"attack": replay_attack

"mitigation":
Context: MissionComputer.impl
preventSpoofing(c6)
preventSpoofing(c: connection) <=
** "spoofing of communication on c is prevented" **
true

**************

"attack": command_injection

"mitigation":
Context: MissionComputer.impl
well_formed(UARTDriver,WaypointManager,c6)
permitWellFormedData(s1: system, s2: system, c: connection) <=
** "connection c only permits well-formed data to flow from s1 to s2" **
true
```
An assurance case is:

- Structured argument consisting of a tree of claims (goals), each supported by evidence, or subgoals and arguments
- Mechanism for capturing and combining all the analyses and verification performed on the architecture and its components

Resolute is a logic and tool for embedding assurance cases in AADL models

- Directly linked to architecture
- Includes different types/sources of evidence
- Reviewable by domain experts
- Adds precision to informal reasoning
2 : CYBER RESILIENT ARCHITECTURE PATTERNS

• Library of general, tool-assisted *architecture model transformations* that mitigate vulnerabilities or address cyber requirements
• Automatic insertion and verification of transform properties as assume-guarantee contracts and assurance case claims
• Examples
  • Filter ✓
  • Attestation ✓
  • Isolation (VM) ✓
  • Monitor/Simplex ✓
  • Distributed Action (e.g., Zeroize)
  • seL4 implementation ✓
COMPOSITIONAL REASONING

ASSUME GUARANTEE REASONING ENVIRONMENT (AGREE)

- Each subsystem has a contract consisting of assumptions and guarantees
- The contract of a component abstracts the behavior of its implementation
- Contracts at each layer must be satisfied by contracts of its components
- Leaf component contracts must be satisfied by implementation
- Compositional analysis provides scalability
Many architecture transformations require the introduction of new special-purpose components.

These must be high-assurance components:
- Library of pre-verified components, or
- Synthesized from formal specification (with proof)

Generate high-assurance components using CakeML:
- AGREE specification -> CakeML specification
- Provably correct synthesis
- Additional infrastructure to create component interface

In addition, we can build systems using the formally verified seL4 kernel and its build system (CAmkES).
**HIGH ASSURANCE IMPLEMENTATION**

- Goal: End-to-end verification-based assurance
  - AADL properties -> running system properties

**AADL Architecture** | **CAmkES architecture** | **seL4-based implementation**

- Assurance: separation properties hold for AADL
- Assurance: separation properties hold for CAmkES architecture
- Assurance: separation properties hold for seL4 implementation

Verification of mapping
AADL to CAmkES

Verification of mapping
CAmkES to seL4
PUTTING IT ALL TOGETHER

- UAV model
  https://github.com/loonwerks/CASE/tree/master/CASE_Simple_Example_V3
REQUIREMENTS ADDED TO AADL MODEL
TRANSFORMATION: FILTER ADDED TO AADL MODEL
COMPONENT SYNTHESIS

• Verified component built using CakeML
• Generated from regex filter spec
• Proof that regex implements AGREE property
• Generate CAmkES component for insertion into system
1. Cyber requirements generated from initial system design

2. Tool-assisted architectural transform

3. Insert new component
   Generated from spec
   Proof of correctness

4. Verification of legacy component

5. Transformed system satisfies cyber requirements
   Assurance case integrates evidence
   Implementation generated from model with proof of equivalence
Goal: Automate much of the manual engineering effort from HACMS

- Identify software to be isolated
  - Thread, thread group, or process
- Apply isolation transform
  - Creates virtual processor
  - Converts software to process
  - Converts connections as needed
  - Binds process to virtual processor
- Apply seL4 implementation transform
INTEGRATED TOOL ARCHITECTURE

OSATE

AADL UAV model

HAMR plugin

AGREE

CASE TA2 plugin

Resolute

Reqts plugin

Binary analysis plugin

Component builder

SPLAT

C compiler

CAmkES input model

CAmkES

Component implementation and glue

Component implementation and glue

Component implementation and glue

CakeML files

*.c files

System Build evidence

C compiler

UAV exe

Linux

UAV exe

seL4

capDL

Support for multiple operating systems

Collins Aerospace
Code, papers, videos available at:

Loonwerks.com

github.com/Loonwerks