Architecture-Level Security Concerns in a Safety Critical System

Sam Procter
Aside: This talk vs. My paper

Paper topic: An *operationalized* taxonomy of system errors

Not covering this topic directly in this talk, but I’m happy to answer questions about it.
Agenda

0. AADL Primer
1. Safety in AADL
2. Security in AADL
3. Safety + Security
AADL: The language used for this work

AADL focuses on interaction between the three elements of a software-reliant mission and safety-critical systems

**The Physical System**
- Aircraft, Car, Train
- Physical Interface Platform Component

**The Software System**
- Embedded Operational Avionics & Mission Software
- SW Design & Runtime Architecture
- Deployed on Utilizes

**The Computer System**
- Computer System Hardware & OS
What does AADL actually look like?

Semi-formal semantics

Only architectural elements

Annexes add functionality:
- Error Modeling
- Behavior
- Code Generation

```plaintext
79  -- Basic/naive version that abstracts all the valves with
80  -- a selector subsystem. This selector subsystem hide
81  -- the physical logic behind the selector, shut off and meter/anti-skid
82  -- valves.
83  system implementation wbs.basid extends wbs.generic
84  subcomponents
85  bscu : refined to system impl::bscu::bscu.basic;
86  -- The selector subsystem
87  selector : refined to system impl::valves::selector_basic{Classifier_Substitution_Rule => Type_E}
88  wheel : refined to system impl::wheel::wheel_one_input.i{Classifier_Substitution_Rule => Type_E}
89  connections
90  blue_to_selector : bus access blue.pump.pressure_output <=> selector.blue_input;
91  green_to_selector : bus access green.pump.pressure_output <=> selector.green_input;
92  bscu_sel_to_selector : port bscu.Select Alternate -> selector.select_alternate;
93  bscu_cmdnor_to_selector : port bscu.cmd.norm -> selector.cmd.norm;
94  bscu_cmdalt_to_selector : port bscu.cmd.alt -> selector.cmd.alt;
95  selector_to_wheel : port selector.output -> wheel.input;
96 end wbs.basid;
```
AADL excels at analyzing component-based systems by
• integrating annotated components
• running system-level analyses
The benefit of a “Single Source of Truth”

Change of Encryption from 128 bit to 256 bit

Higher CPU Demand

Increased Latency

One change drives multiple system issues!

Potential New Hazard

Affects Temporal Correctness

Potentially New Hazard
Agenda

0. AADL Primer
1. Safety in AADL
   1. Background
   2. ALISA + EMV2
   3. Why generate reports?
2. Security in AADL
3. Safety + Security
Safety Background: Fault Tree Analysis

Bell Labs, 1962

Looks for contributory causes to undesired events

Doesn’t really have a notion of “component” or use system structure
### Safety Background: Failure Modes and Effects Analysis

**FMEA: US Military, 1949**

- Analyses impacts of individual components
- Doesn’t clearly address component-interaction problems

<table>
<thead>
<tr>
<th>System: PCA Interlock Scenario</th>
<th>Subsystem: Pulse Oximeter Device</th>
<th>Mode/Phase: Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td><strong>Failure Mode</strong></td>
<td><strong>Fail Rate</strong></td>
</tr>
<tr>
<td>Provide SpO₂</td>
<td>Fails to Provide</td>
<td>N/A</td>
</tr>
<tr>
<td>Provides late</td>
<td>Provides late</td>
<td>N/A</td>
</tr>
<tr>
<td>Provides wrong</td>
<td>Provides wrong</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Analyst: Sam Procter  
Date: September 26, 2016
Safety in AADL: Research Background

Backwards-iterating, component-based analysis

Merges top-down / bottom-up styles

1. Derive a component’s local notion of harm

3A. Consider the impacts of internal problems (ie, faults)

2. Consider the impacts of bad input from other components (ie, errors)

3B. Move backwards one element in the control structure
How do you incrementally assure a system?

Start early – link requirements to:

- Each other
- Architectural components

Document:

- Goals, stakeholders, etc.
- Verification plans

Generate:

- Coverage reports
- Hazard analyses
ALISA Example

- **Hierarchical assurance plan**
- **JUnit-Style evidence evaluation**
- **Human-readable description**
- **Goal link**
EMV2: Contracts for Error Behavior

```java
package PCA_ShutoffLogic
public
with PCA_Shutoff_Types, PCA_Shutoff_Properties, MAP_Properties;

process ICEpcaShutoffProcess
features
  SpO2 : in event data port PCA_Shutoff_Types::SpO2;
  CommandPumpNormal : out event data port PCA_Shutoff_Types::PumpNormalCommand;

properties
  MAP_Properties::Component_Type => logic;
annex EMV2 {**
    use types PCA_Shutoff_Errors;
    error_propagations
      SpO2 : in propagation {SpO2ValueHigh};
      CommandPumpNormal : out propagation {InadvertentPumpNormally};
    flows
      HighSpO2LeadsToOD : error path SpO2{SpO2ValueHigh} -> CommandPumpNormal{InadvertentPumpNormally};
    end propagations;
    **};
end ICEpcaShutoffProcess;

-- Process implementation redacted
end PCA_ShutoffLogic;
```
Interaction between report generation and error propagation

1. Here’s an empty cell... could anything go wrong?

2. Create Error Model annotations

3. Where else could this error go?

4. What effect would this error have?

Cause -> Effect

Effect -> Cause

Interaction between report generation and error propagation
Agenda

0. AADL Primer
1. Safety in AADL

2. Security in AADL
   1. Background
   2. AADL & MILS

3. Safety + Security
Security in AADL: Research Background

1970s: Multi-level security
- Bell-LaPadula (Confidentiality)
- Biba (Integrity)

2000s: Multiple Independent Levels of Security (MILS) – Four arguments:
- Local Policy Assurance
- Integrating Policy Assurance
- Individual Resource Separation Assurance
- Integration Resource-Sharing Assurance
AADL in large-scale formal methods: SMACCM & D-MILS

D-MILS
- Extension of MILS to networked systems
- Customized subset of AADL

SMACCM
- “Unhackable” UAVs
- AGREE / Resolute
Security policy vulnerabilities: Analyze Information Flows
Examples: Verify secrets stay secret, and Sensors can’t send commands

Security enforcement vulnerabilities: Analyze Deployment Mechanisms
Example: Hi and low-security channels shouldn’t coexist on unpartitioned hardware

AADL Support for MILS

Research Connection:
Apply Multiple Independent Levels of Security (MILS) framework (confidentiality) to system security (integrity)
Partitioning code sample

```java
package Sandbox
public
    with SecurityProps;

system An
end An;

system implementation An.Example
    subcomponents
        ComponentA: system A;
        ComponentB: system B;
    connections
        conn: port ComponentA.out_port -> ComponentB.in_port;
    properties
        SecurityProps::PolicyElements => SecurityProps::BibaPolicy;
end An.Example;

system A
    features
        out_port: out event data port;
    properties
        SecurityProps::level => high;
end A;

system B
    features
        in_port: in event data port;
    properties
        SecurityProps::level => low;
end B;
end Sandbox;
```

Policy specification

Security levels and / or partition

Conflict if we switch to Bell-LaPadula!
Security Analysis Techniques and Tools

0. Consistency in security policy specification & enforcement
1. Model-Based Attack Impact Analysis (AIA) tool
2. Model-Based Attack Tree Analysis (ATA) tool
3. Generation of security configuration files
   - Model-based auto-configuration of certified kernel (seL4/CAmkES) security policy
Using Security Assurance Techniques and Tools

1. Specify security policy as verifiable requirements
2. Formalize existing verification activities
3. Automate execution of verification plans
Agenda

0. AADL Primer
1. Safety in AADL
2. Security in AADL
3. Safety + Security
   1. Effects focus
   2. Code generation
   3. Slicing & Data-Flow
Modeling Security Requirements in the Context of Safety

**Approach:** Use effects-focused analysis and tooling

- When are various techniques appropriate?
  - Biba model (integrity)
  - Bell–LaPadula (confidentiality)
- What “building blocks” should be used?
  - examples: encryption, partitioning, checksums
- How should requirements be verified?

**Measurement:** Proposed user study (in FY 20) to measure qualities of design and analysis guidance

- Objective qualities
  - Number of issues found / avoided
  - Time required
- Subjective qualities
  - Quality of issues found / avoided
  - Complexity
Using Theory to Guide Tool Development

**Approach:** Use fault-injection tooling
- Fault-injection pairs naturally with an effects focus
- Collaborators are building a large simulation and verification environment to enable this testing

**Measurement:**
- Current AADL can describe component behavior in the presence of errors
- This project will let us verify those descriptions
Code Auto-generated from AADL

```python
thread Patient_Bolus_Checker
features
    Minimum_Time_Between_Bolus: in data port ICE_Types::Minute;
    Patient_Button_Request: in event port;
    Patient_Request_Not_Too_Soon: out event port;
    Patient_Request_Too_Soon: out event port;
end Patient_Bolus_Checker;

def sendPatient_Request_Not_Too_Soon(value : Slang_Types.Empty) : Unit = {
    Art.putValue(Patient_Request_Not_Too_Soon_Id, Slang_Types.Empty_Payload(value))
}

def sendPatient_Request_Too_Soon(value : Slang_Types.Empty) : Unit = {
    Art.putValue(Patient_Request_Too_Soon_Id, Slang_Types.Empty_Payload(value))
}

def getMinimum_Time_Between_Bolus() : ICE_Types.Minute = {
    val ICE_Types.Minute_Payload(value) = Art.getValue(Minimum_Time_Between_Bolus_Id)
    return value;
}
```
Looking forward: Data-Flow Analysis

What do all these analyses have in common?

- The use the “data flow” view of a system

Colleagues at K-State (Hariharan Thiagarajan, John Hatcliff, Robby) are bringing data-flow and slicing to AADL models / generated simulation code.

We’re working on integrating this into our tool’s standard distribution
Architecture-Level Security Concerns in a Safety Critical System

Sam Procter
Questions: Modeling Strategy

We don’t model users – how do we model access control?
  • Data types

We don’t model state – how do we model protocols?
  • Virtual buses

Larger question: How should security-related concepts be modeled?
  • Should adding new concepts be a last resort?
    - This can give a nice, compact language
  • … Or should they be added to avoid “hacks?”
    - This can make the language more readable

Related: When should security-related concepts be modeled?