Take a SEAT: Security-Enhancing Architectural Transformations

Isaac Amundson [1] Darren Cofer [1] Junaid Babar [1] Eric Mercer [2] Karl Hoech [1] David Hardin [1] Johannes Åman Pohjola [3] Konrad Slind [1]

Sept. 14, 2020

[1] Collins Aerospace

[2] BYU

[3] Data61

Overview

1. Architectural Transformations
2. Deep Dive: Message Analysis
3. Assembling a Security Case

Architectural Transformations

System Architecture

- General setting: **Architectural Design Languages**
- An ADL supports complete, highly abstract, views of a system, including hardware, software, (and possibly humans)
- An architecture model should provide a high-level setting in which the *whole picture* of a system can be surveyed
- Thus: a place where existing implementations, new design features, high-level requirements, implementations, and verifications can be combined.
- Not just boxes and arrows!

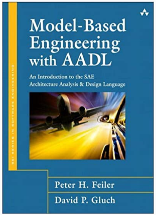
CASE

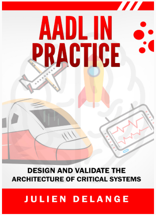
- In the DARPA **CASE** project we are developing the idea of *Security-Enhancing* transformations on such architectural descriptions.
- The goal is to develop a methodology and case studies where
  - the structure of an existing (legacy) system is captured in an architectural model;
  - system security is automatically analyzed and any security problems are addressed by applying architectural transformations
- A key aspect is use of formal specification languages and automatic synthesis of security mechanisms

AADL

We have been using Architecture Analysis and Design Language (AADL) as our architecture modelling language.

- Expressive: allows specification of
  - memory and buses
  - software (types + behavior)
  - hierarchical organization of components
  - communication
  - scheduling
- Tool support in Eclipse (via OSATE)
- Popular: growing user base, tutorials, books, etc.





AADL in CASE

AADL is extensible via *annexes*. At Collins we have developed two annexes used on many projects in the Trusted Systems Group:

AGREE SMT-based reasoning over Assume-Guarantee contracts on components

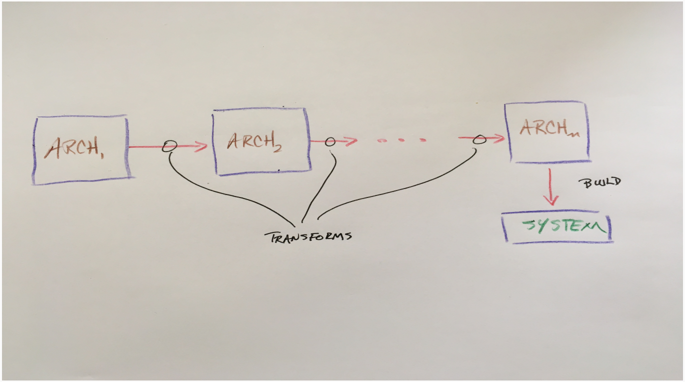
Resolute Assurance cases as formal entities, using proof search to explore cases.

In CASE, AGREE is used to formulate behavioral security rqts.

Resolute is used for structural properties, and also for linking results from disparate proof systems.

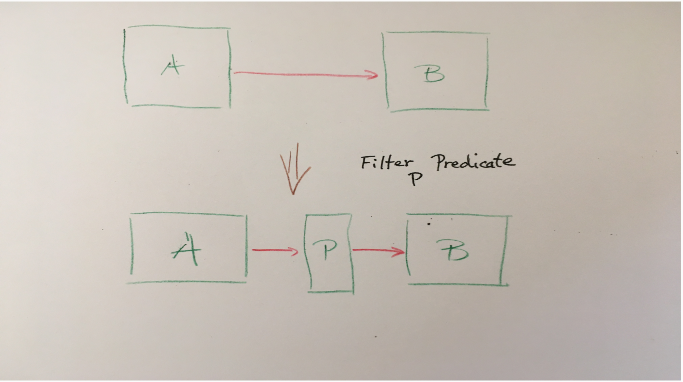
Architecture Transformations

We have been developing a collection of architecture-to-architecture maps that can be applied to *provably increase* the security of a system.



Transformation: Message Filtering

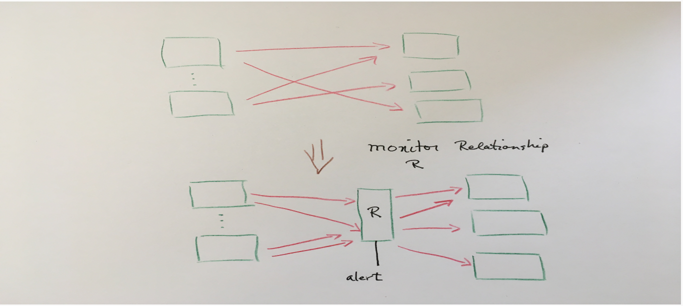
A *filter* is conceptually very simple: it checks validity of its input data.



If the data is valid, then it is passed on. Otherwise it is dropped.

Transformation: Message Monitoring

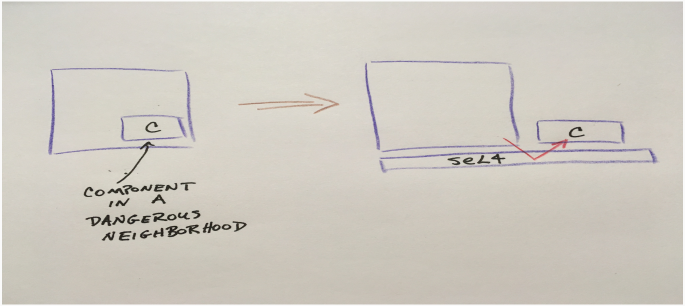
A *monitor* checks to see that a relationship $\mathcal{R}$ holds over a collection of message streams through time. If the specification is violated, an *alert* is sent out.



We currently use past-time temporal logic to specify monitors

Transformation: Isolation of ‘at risk’ components

An unprotected computational element can be isolated by transparently lifting it out of its context and mediating access via seL4.



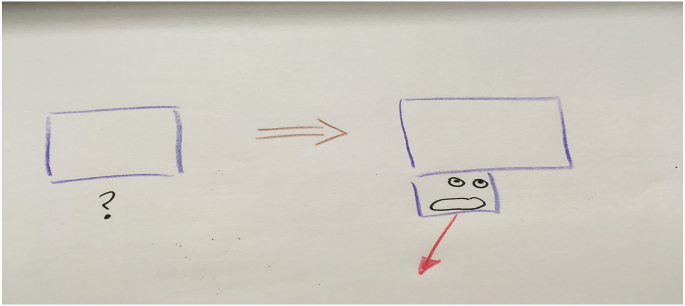
Correctness of this transformation depends on formal guarantees provided by seL4.

seL4

- seL4 microkernel guarantees partitioning of components and communication, backed by computer-checked proofs
- seL4 guarantees no infiltration, exfiltration, eavesdropping, interference, and provides fault containment for untrusted code

Transformation: Attestation

Attestation inserts measurement mechanisms into a system. These examine various aspects of system behavior, and send summaries back to an observer system.



Example System: uxAS

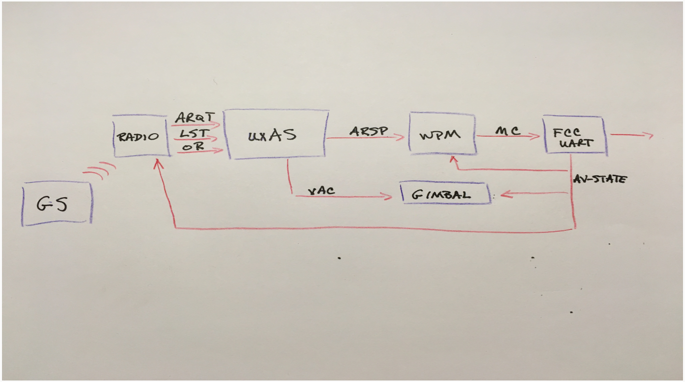
The example we have been using is **uxAS**, a framework for creating autonomous aerial systems from AFRL.

`https://github.com/afrl-rq/OpenUxAS`

- Open Source
- Previous experience during the AFRL Summer of Innovation
- Good setting in which to exercise our ideas

uxAS system architecture

The initial system model we start from :



Notes on the model

- UAV is preloaded with a collection of *Operating Regions* (Keep-in and Keep-out zones)
- Commands from GS:
  - **OR** Set operating region
  - **LST** LineSearchTask: *Follow the given sequence of points*
  - **ARQT** : AutomationRequest: *Create a flight plan to achieve a high-level description, e.g., "surveil the given OR in a grid pattern"*
- Internal messages
  - **ARSP** Response to Automation Request
  - **VAC** VehicleActionCommand
  - **MC** MissionCommand
  - **AV-State** AirVehicleState

Security Considerations

- **Problem**: An Operating Region message should designate a known region

Security Considerations

- **Problem**: An Operating Region message should designate a known region
- **Solution**: Filter OR message to be valid regions

Security Considerations

- **Problem**: An Operating Region message should designate a known region
- **Solution**: Filter OR message to be valid regions
- **Problem**: Each point in a Line Search Task should be a valid GPS coordinate

Security Considerations

- **Problem**: An Operating Region message should designate a known region
- **Solution**: Filter OR message to be valid regions
- **Problem**: Each point in a Line Search Task should be a valid GPS coordinate
- **Solution** Filter LST message on GPS coords

Security Considerations

- **Problem**: An Operating Region message should designate a known region
- **Solution**: Filter OR message to be valid regions
- **Problem**: Each point in a Line Search Task should be a valid GPS coordinate
- **Solution** Filter LST message on GPS coords
- **Problem**: Each point in an Automation Response should be in a keep-in zone and not in a keep-out zone

Security Considerations

- **Problem**: An Operating Region message should designate a known region
- **Solution**: Filter OR message to be valid regions
- **Problem**: Each point in a Line Search Task should be a valid GPS coordinate
- **Solution** Filter LST message on GPS coords
- **Problem**: Each point in an Automation Response should be in a keep-in zone and not in a keep-out zone
- **Solution** Monitor ARSP messages wrt Keep-in and Keep-out zones

Security Considerations

- **Problem**: An Operating Region message should designate a known region
- **Solution**: Filter OR message to be valid regions
- **Problem**: Each point in a Line Search Task should be a valid GPS coordinate
- **Solution** Filter LST message on GPS coords
- **Problem**: Each point in an Automation Response should be in a keep-in zone and not in a keep-out zone
- **Solution** Monitor ARSP messages wrt Keep-in and Keep-out zones
- **Problem**: Every point in an Automation Response should be a valid GPS coord

Security Considerations

- **Problem**: An Operating Region message should designate a known region
- **Solution**: Filter OR message to be valid regions
- **Problem**: Each point in a Line Search Task should be a valid GPS coordinate
- **Solution** Filter LST message on GPS coords
- **Problem**: Each point in an Automation Response should be in a keep-in zone and not in a keep-out zone
- **Solution** Monitor ARSP messages wrt Keep-in and Keep-out zones
- **Problem**: Every point in an Automation Response should be a valid GPS coord
- **Solution** Filter ARSP message

Security Considerations continued

- **Problem**: Every Automation Request should have a corresponding Response

Security Considerations continued

- **Problem**: Every Automation Request should have a corresponding Response
- **Solution**: Monitor ARQT and ARSP for responses matching requests

Security Considerations continued

- **Problem**: Every Automation Request should have a corresponding Response
- **Solution**: Monitor ARQT and ARSP for responses matching requests
- **Problem**: uxAS software is open source and Waypoint Manager could be compromised

Security Considerations continued

- **Problem**: Every Automation Request should have a corresponding Response
- **Solution**: Monitor ARQT and ARSP for responses matching requests
- **Problem**: uxAS software is open source and Waypoint Manager could be compromised
- **Solution**: Isolate Waypoint Manager on a VM

Security Considerations continued

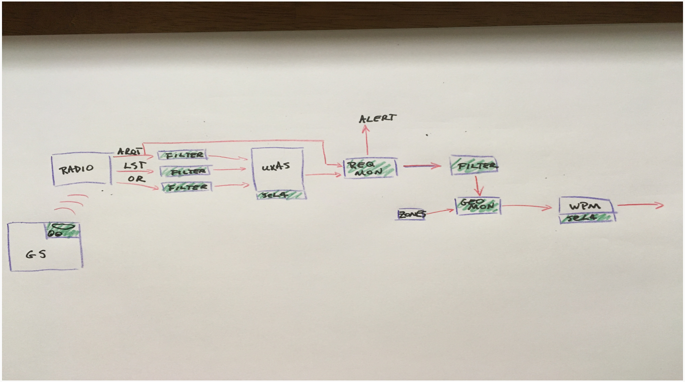
- **Problem**: Every Automation Request should have a corresponding Response
- **Solution**: Monitor ARQT and ARSP for responses matching requests
- **Problem**: uxAS software is open source and Waypoint Manager could be compromised
- **Solution**: Isolate Waypoint Manager on a VM
- **Problem**: Ground station could be compromised; UAV needs to defend itself against that

Security Considerations continued

- **Problem**: Every Automation Request should have a corresponding Response
- **Solution**: Monitor ARQT and ARSP for responses matching requests
- **Problem**: uxAS software is open source and Waypoint Manager could be compromised
- **Solution**: Isolate Waypoint Manager on a VM
- **Problem**: Ground station could be compromised; UAV needs to defend itself against that
- **Solution**: Attestation Manager on GS, observing activity on the GS and talking to UAV

Transformed uxAS

The transformed model :



Implementations and Build

As the transformations are applied, implementations and configuration information are generated.

- For “simple” transformations code can be generated at transformation time.
- For the VM and Attestation transforms, configuration information can be generated, but the details of the implementation are more involved.

Finally the **BUILD** can now be invoked.

Build

The build is very challenging. The **HAMR** toolsuite implements multi-stage translation architecture to address the following goals:

- Semantic consistency from model to execution ensures model-level analysis applies to deployed code
- Build for multiple target platforms: seL4, Linux
- Same computational model across different platforms
- Same semantics for threading and communication

Summary

System designer uses OSATE, using menus to specify and configure transforms.

The interface makes sanity checks and sets up for the eventual system build

For a nice video on the interface see

`http://loonwerks.com/projects/case.html`

Deep Dive: Message Filters

Solutions lead to new problems

Once implementations are created for the new system components, we have to guard against increasing the attack surface of the system.

Formal specification and proof to the rescue!

Ports and Messages

Question: what does a filter operate over?

- An AADL component communicates with other components via *connections*
- An endpoint (port) on a connection has a *type* (booleans, integers, floats, arrays, unions, ...)
- Properties in AGREE (our specification language) are therefore written over these types.
- However, in the implementation, messages coming into a port are byte arrays (essentially untyped)

Example : Wellformed GPS coordinates in AGREE

```
AltitudeType = AGL | MSL

Location3D = {
 Latitude  : real64,
 Longitude : real64,
 Altitude  : real32,
 AltitudeType : AltitudeType}

Good_Location (loc) =
   -90.0 <= loc.Latitude <= 90.0 and
  -180.0 <= loc.Longitude <= 180.0 and
     0.0 <= loc.Altitude <= 15000.0
```

Example : Wellformed GPS strings

$$
\begin{aligned}
&\textbf{Good\_Location\_String}(s) \iff \\
&\quad \exists s_1 s_2 s_3 s_4. \\
&\qquad s = s1 \bullet s2 \bullet s3 \bullet s4 \wedge \\
&\qquad -90.0 \leq \textbf{doubleVal}(s1) \leq 90.0 \wedge \\
&\qquad -180.0 \leq \textbf{doubleVal}(s2) \leq 180.0 \wedge \\
&\qquad 0.0 \leq \textbf{floatVal}(s3) \leq 15000.0 \wedge \\
&\qquad 0 \leq \textbf{natVal}(s4) \leq 1
\end{aligned}
$$

where

$$
\begin{aligned}
&\textbf{doubleVal} && : \textbf{string} \rightarrow \textbf{double} \\
&\textbf{floatVal} && : \textbf{string} \rightarrow \textbf{float} \\
&\textbf{natVal} && : \textbf{string} \rightarrow \textbf{nat}
\end{aligned}
$$

SPLAT

We need to span the gap between high level data and flat strings.

Our approach : **SPLAT** (Semantic Properties for Language and Automata Theory)

Try to apply ideas from Formal Language Theory to showing properties of operations on high-level data.

Contiguity Types

The goal is to automatically generate an implementation of the well-formedness predicate `Good_Location` from its specification.

There is a problem : the encoding from datastructures to strings is not specified.

For uxAs, this is a fairly complex encoding.

We specify the message format using *contiguity types*. With this representation we can

- automatically generate message filters and parsers
- automatically prove that the filter has the desired property (`Good_Location_String` in our example)

uxAS LineSearchTask messages

uxAS messages are quite elaborate

Include features such as unions and variable-length arrays

```
{TaskID :  i64,
 Label :   vString,
 EligibleEntities : BoundedArray i64 32,
 ...
 Parameters : BoundedArray keyValuePair_Option 8,
 ...
 DesiredWavelengthBands : BoundedArray WavelengthBand 8,
 ...
 PointList : BoundedArray location3D_Option 1024,
 ViewAngleList : BoundedArray wedge_Option 16,
 ...}
```

Contiguity Types: Syntax

The syntax of contiguity types is very similar to a standard collection of base types closed under formation of records and arrays.

$$
\begin{array}{rcl}
\mathit{base} & = & \textbf{bool} \mid \textbf{char} \mid \textbf{u8} \mid \textbf{u16} \mid \textbf{u32} \mid \textbf{u64} \mid \textbf{i16} \mid \textbf{i32} \mid \textbf{i64} \mid \textbf{f32} \mid \textbf{f64} \\
\tau & = & \mathit{base} \\
 & \mid & \textbf{Recd}\ (f_1 : \tau_1) \ldots (f_n : \tau_n) \\
 & \mid & \textbf{Array}\ \tau\ \mathit{exp} \\
 & \mid & \textbf{Union}\ (\mathit{bexp}_1 : \tau_1) \ldots (\mathit{bexp}_n : \tau_n)
\end{array}
$$

Contiguity Types: Semantics

The semantics of contiguity types is in terms of formal languages (sets of strings):

$$\mathcal{L}_\theta(\tau) = \texttt{case}\ \tau \begin{cases} \mathit{base} \Rightarrow \{s \mid \textbf{len}(s) = \textbf{width}(\mathit{base})\} \\ \textbf{Recd}\ (f_1 : \tau_1) \ldots (f_n : \tau_n) \Rightarrow \mathcal{L}_\theta(\tau_1) \cdot \ldots \cdot \mathcal{L}_\theta(\tau_n) \\ \textbf{Array}\ \tau_1\ \mathit{exp} \Rightarrow \mathcal{L}_\theta(\tau_1)^{\textbf{evalExp}\ \theta\ \mathit{exp}} \\ \textbf{Union}\ (\mathit{bexp}_1 : \tau_1) \ldots (\mathit{bexp}_n : \tau_n) \Rightarrow \\ \qquad \begin{cases} \mathcal{L}_\theta(\tau_i) & \text{if } \textbf{evalBexp}\ \theta\ \mathit{bexp}_i = \textbf{true} \\ & \text{and no other } \mathit{bexp}_j \text{ is } \textbf{true} \\ \emptyset & \text{otherwise} \end{cases} \end{cases}$$

Correctness of parameterized matcher

We can define a function **match** which takes a contig type and a string and returns an assignment of slices of the string to elements of the type.

Theorem (Correctness)

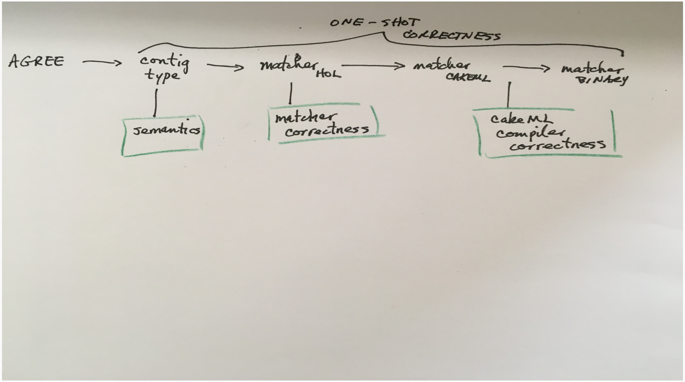
$$\vdash \textbf{match}\ \textit{contig string} = \textbf{Some}(\theta) \Rightarrow \textit{string} \in \mathcal{L}_\theta(\textit{contig})$$

match has the flavor of a *parser generator*: it takes a specification of the language to be parsed and returns an implementation

We use **match** to implement all filters in the transformed uxAS.

Joining theorems together

We can join the correctness of the message matcher with the correctness of the CakeML toolchain to obtain a *single-shot* correctness theorem in HOL4:



Question: value of end-to-end verification

Q: What is the value of combining verified programs with a verified compiler to get a property of the compiled program?

A: It removes places to look for bugs. Instead, the assumptions of the final joined-up theorem reveal the limitations on applicability of the result.

Thread level correctness

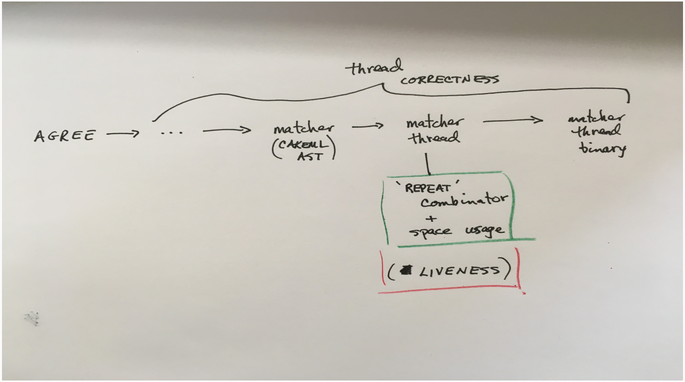
But we need to do better: the filter is run inside a loop:

```
while true
do {
  getInput();
  if match contig inputBuf {
    putOutput (inputBuf);
   }
  else skip;
 }
```

Thread level correctness

Thanks to some great work from Johannes Åman Pohjola at Data61 we can prove the correctness at the thread level:

- proof rules for infinitary computations with I/O
- space bounds proofs



Assembling a Security Case

What is the Claim?

We have to argue that the newly generated implementations have improved the security of the system.

Current work, so this is somewhat brain-stormy.

System-level correctness

Recall the **NEAT** acronym on necessary properties for a reference monitor:

Non-bypassable All paths to target go through RM

Evaluable testable, verifiable

Always invoked The RM algorithm is invoked on each and every input

Tamper proof The RM is not over-writable

Desirable properties for our implementation! We already ‘have’ some of these :

Evaluable (Formal proofs)

Tamper proof (seL4 gives isolation)

System-level correctness (continued)

Non-bypassable System-wide property which depends on the executable rigorously obeying the boundaries in the model. This property depends on the fact that HAMR / seL4 enforces architectural boundaries all the way down.

Always invoked Does the filter sometimes ignore its input and output a stored well-formed element? It shouldn't but how could one tell? (Answer: the thread property implies this.)

Surveyable system-level correctness with Resolute

- For each filter and monitor, the above properties need to be shown and stored.
- The vast iceberg of the correctness of the security properties (in Isabelle/HOL) that seL4 implements need to be brought into the picture.
- For attestation: the toolchain verification story is Coq-based.
- Disparate evidence supporting our security claim.
- Our solution: Represent the security argument in Resolute.
- Goal is surveyability of the full correctness story for the enhancements generated by the architectural transformations

Current and Future Work

- Starting final stage of CASE; CH47 helicopter is our transition platform
- Currently writing documentation and training materials for our industrial partners.
- Check it out:

  `http://loonwerks.com/projects/case.html`