Mixing formal methods to increase robustness against cyber-attacks

Laurent Voisin
Systerel in a nutshell

Critical systems engineering

System Expertise

Critical Systems

Safety Cyber-security

Critical Software

Tests, Proof & Simulation
**OPC-UA**

Machine to machine communication

Browse, read/write, subscribe, ...

Built-in security

![OPC-UA Diagram](image)

IEC 62541 standard

Cornerstone of Industry 4.0 and Industrial IOT
French collaborative R&D project INGOPCS

- Backed by ANSSI ("French NSA")
- Partial funding by the French Government (FUI19)

Cleanroom development of the OPC-UA protocol in C99

Main S2OPC targets

- Safety (SIL2 – IEC 61508)
- Security (EAL4 – Common Criteria)
- Embedded systems
- Open source
How to reach high quality software?

Apply formal methods!

But difficult in the S2OPC context:

- Open world
- Concurrent
- Cryptography
- Dynamic data allocation
Taming concurrency

Architectural pattern

Sequential automata executing concurrently

Post asynchronous messages between automata

No shared memory, but ownership transfer by message passing

Examples:

- Low-level socket operations
- Channel events
- Application interface

Simple to reason about

Programming is a bit more difficult (asynchronous, callback based)
About Cryptography

Difficult to get it right

Do not reinvent the wheel

Reuse existing crypto library (e.g., Mbed TLS)

Isolate it through a thin API adaptor

Allows plugging hardware crypto when available
Mixing Formal Methods

The S2OPC code is heterogeneous

• Use Frama-C / TrustInSoft Analyser for low level
• Use the B method for high level

Take advantage of the strengths of each formal method

Do not attempt to cover 100 %

• Diminishing returns
Applied to low-level code

- OS interface
- crypto API
- message en/decoding

Provides extended static analysis

- Absence of undefined behavior
- Check dynamic CERT coding rules (e.g., buffer overflow)

A posteriori verification
What is the B Method?

Developed in the 90s
Correct by construction software
High level specifications in set theory (similar to SQL)
Then stepwise refinement to actual code (B0)
Finally automated one-to-one translation to C99 code
Proof of correctness and consistency of the model

Usually applied to SIL4 embedded software (e.g., CBTC)
Use of the B Method

Applied to high-level code

- Channel automaton
- Session automaton
- Query processing on the address space

Simple high-level description, complex implementation

- Refinement to the rescue

Global invariants

A priori verification
Development process

Formal methods are not enough

Apply an agile process
  • With long runs (about two months)

Apply best practices of software engineering
  • Automated code formatting
  • Code reviews
  • Source version control (incl. signed commits and pull requests)
  • Continuous integration
  • Static analyses (each compiler gives a different feedback)
  • Unit, integration and acceptance testing (where applicable)
  • Fuzz testing
Need to model dynamic data

Traditionally B is applied in a safety-critical context

Dynamic data allocation is not permitted

But for a network protocol:

- The size of messages is unknown
- Fixed boundaries would be difficult to estimate
- Fixed boundaries are a waste of tight memory

The networking world is open by nature
Simple C pointers

Simple pointers

- int *p;
- p = malloc(sizeof *p);
- p == NULL
- *p = 42;
- x = *p;
- p = q;
- free(p);

Not considered (aliasing)

- p = &x;

Similar to Pascal pointers
## B model (types)

### SETS
- **t_int_i** /* Any value */

### CONSTANTS
- **t_int**, /* Valid pointers */
- **c_int_undef**, /* NULL pointer */

### PROPERTIES
- \( t_{\text{int}} \subseteq t_{\text{int}_i} \land 
  c_{\text{int}_\text{undefined}} \in t_{\text{int}_i} \land 
  c_{\text{int}_\text{undefined}} \notin t_{\text{int}} \)

```c
int *p;
p \in t_{\text{int}_i}
p == null
p = c_{\text{int}_\text{undefined}}
p = q;
p := q
```
Model allocated pointers and associated values

VARIABLES f_int /* Value of allocated data */

INVARIANT f_int ∈ t_int → INT

INITIALISATION f_int := ∅

Note: f_int is abstract (does not exist outside the model)

Would be a ghost variable in SPARK.
Dynamic Data

Conclusion

About us

Context

Approach

Dynamic Data

p ← int_alloc △

CHOICE

 p := c_int_undef

OR

 ANY np, ni

WHERE np ∈ t_int – dom(f_int) ∧ ni ∈ INT

THEN p := np || f_int(np) := ni

END

END

int_free(p) △

/* free(p) */

PRE

p ∈ dom(f_int)

THEN

f_int := {p} ⊲ f_int

END
The present document is the property of Systerel and cannot be reproduced or disclosed without Systerel prior written consent.

B model (dereferences)

\begin{align*}
\text{n} & \leftarrow \text{int\_get(p)} \triangleq \\
\text{PRE} & \\
\quad \text{p} \in \text{dom(f\_int)} & \\
\text{THEN} & \\
\quad \text{n} & := \text{f\_int(p)} \\
\text{END} & \\
\end{align*}

/* n = *p; */

\begin{align*}
\text{int\_set(p, n)} & \triangleq \\
\text{PRE} & \\
\quad \text{p} \in \text{dom(f\_int)} & \\
\text{THEN} & \\
\quad \text{f\_int(p)} & := \text{n} \\
\text{END} & \\
\end{align*}

/* *p = n; */
B model (in and out pointers)

Accept a pointer allocated outside of the model

\[
\text{int\_bless}(p) \triangleq \\
\text{PRE} \\
\quad p \in t\_int - \text{dom}(f\_int) \\
\text{THEN} \\
\quad \text{ANY } ni \text{ WHERE } ni \in \text{INT THEN } f\_int(p) := ni \text{ END} \\
\text{END}
\]

Release a pointer for use outside

\[
\text{int\_forget}(p) \triangleq \\
\text{PRE} \\
\quad p \in \text{dom}(f\_int) \\
\text{THEN} \\
\quad f\_int := \{p\} \triangleleft f\_int \\
\text{END}
\]
Extension to structures

Use several partial functions, one for each field

The domains of these functions must be equal

Example:

```c
struct pos { int x; int y; };

f_pos_x ∈ t_pos ↦ INT
f_pos_y ∈ t_pos ↦ INT
\text{dom}(f_{pos_x}) = \text{dom}(f_{pos_y})
```

A field can itself be a pointer to another structure

A field can be an array of dynamic length
High-level services are modelled in B

Can require that an input pointer is allocated
  • precondition of an operation

Can guarantee that an output pointer is allocated
  • postcondition of an operation body

Can detect and report unavailable memory
  • check and propagate the alloc return value

Can transfer ownership of memory
  • bless and release operations
Conclusion

Adapt your software architecture

Use the right tool for the job

Keep your ROI positive

Efficient and excellent quality code

S2OPC integrated in commercial software
  • network bridge in railway supervision

General availability of B model shows modeling patterns used in industry

Full development available at

https://gitlab.com/systerel/S2OPC
Additional points

But limited to non-recursive structures

- Recursive structures (e.g., linked lists, trees) would need more global invariant (e.g., lists are not circular).