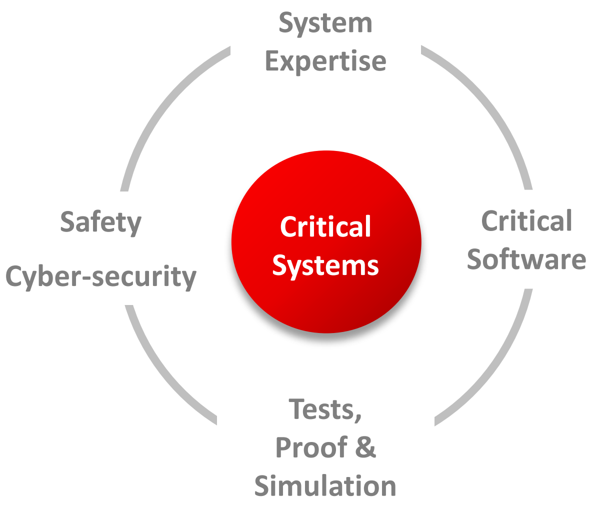
Mixing formal methods to increase robustness against cyber-attacks

Laurent Voisin

12/07/2018

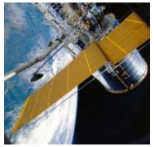
Systerel in a nutshell

Critical systems engineering









Key figures

- About us
- Context
- Approach
- Dynamic Data
- Conclusion

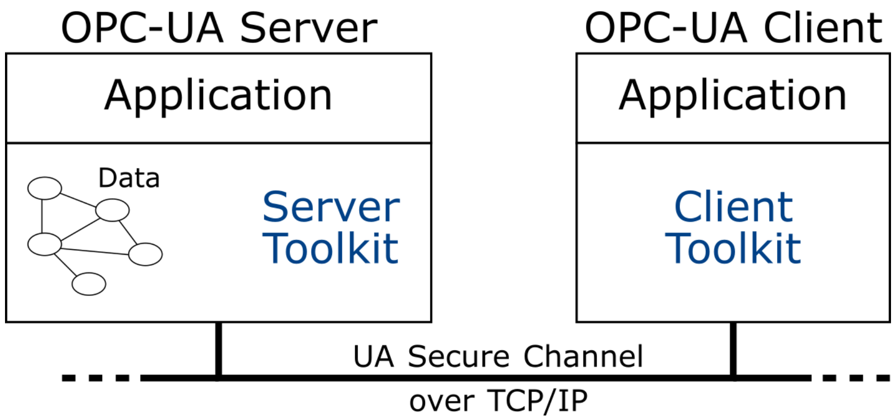
8	100	+10	4
M€ Turnover	ENGINEERS & PhD	YEARS	OFFICES
including 15% dedicated to R&D		Average experience	Aix-en-Provence Paris Toulouse Berlin

OPC-UA

Machine to machine communication

Browse, read/write, subscribe, …

Built-in security



IEC 62541 standard

Cornerstone of Industry 4.0 and Industrial IOT

S2OPC

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

Frama-C / TrustInSoft Analyser

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, c_int_undef	/* Valid pointers */ /* NULL pointer */
PROPERTIES	$t\_int \subseteq t\_int\_i \land$ $c\_int\_undef \in t\_int\_i \land$ $c\_int\_undef \notin t\_int$	

int *p;	$p \in t\_int\_i$
p == null	$p = c\_int\_undef$
p = q;	$p := q$

B model (state)

Model allocated pointers and associated values

VARIABLES	f_int	/* Value of allocated data */
INVARIANT	$f\_int \in t\_int \nrightarrow INT$	
INITIALISATION	$f\_int := \emptyset$	

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

Would be a ghost variable in SPARK.

B model (allocation)


```
p ← int_alloc ≙                    /* p = malloc(sizeof *p); */
  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
```

B model (dereferences)

```
n ← int_get(p) ≙                    /* n = *p; */
   PRE
      p ∈ dom(f_int)
   THEN
      n := f_int(p)
   END

int_set(p, n) ≙                     /* *p = n; */
   PRE
      p ∈ dom(f_int)
   THEN
      f_int(p) := n
   END
```

S² OPC safe & secure

B model (in and out pointers)

Accept a pointer allocated outside of the model

```
int_bless(p) ≙
  PRE
    p ∈ t_int − dom(f_int)
  THEN
    ANY ni WHERE ni ∈ INT THEN f_int(p) := ni END
  END
```

Release a pointer for use outside

```
int_forget(p) ≙
  PRE
    p ∈ dom(f_int)
  THEN
    f_int := {p} ⩤ f_int
  END
```


systerel Safe real-time solutions

Extension to structures

Use several partial functions, one for each field

The domains of these functions must be equal

Example:

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

f_pos_x ∈ t_pos ⇸ INT
f_pos_y ∈ t_pos ⇸ INT
dom(f_pos_x) = dom(f_pos_y)

A field can itself be a pointer to another structure

A field can be an array of dynamic length

Application to S2OPC

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).