

Proof of Security Properties: Application to JavaCard Virtual Machine

Adel DJOUDI, Nikolai KOSMATOV

Joint work with Martin HANA

Frama-C Days 2024
Paris, June 13, 2024



- Specification and verification of security properties with MetAcsI
- Experience of Verification of JavaCard Virtual Machine for Common Criteria Certification
- Ongoing and Future Work

Tool challenge: deductive verification of global security properties

Expressivity challenge

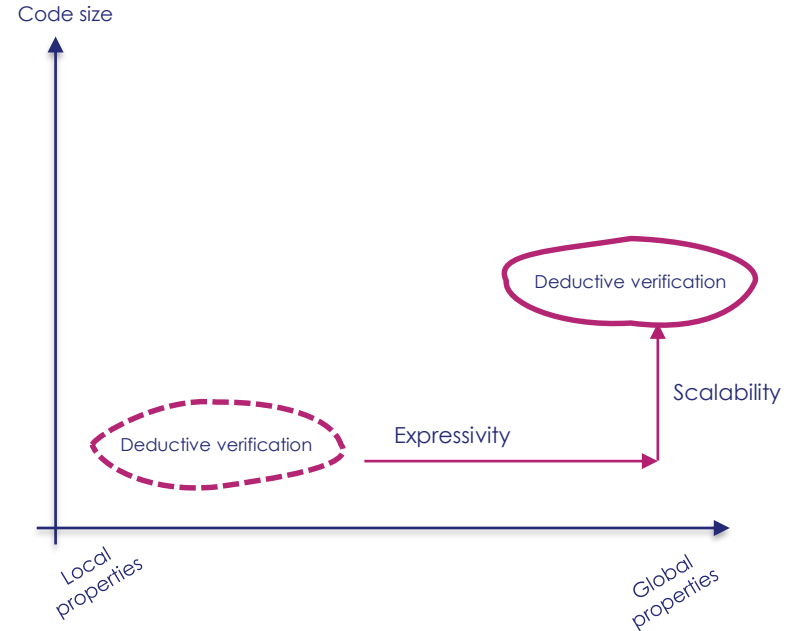
- Initial uncertainty about ability to specify high-level global security properties (confidentiality & integrity) with **ACSL**

Local properties

- **ACSL** annotations
- **Frama-C/RTE** ensures that code is free of undefined behaviors

Global properties

- **Frama-C/WP** ensures weak global invariants for integrity (maintained at function calls and returns, at loop invariants...)
- **FRAMA-C/MetAcsl** ensures strong global invariants (maintained at every sequence point in the program) both for integrity and confidentiality



(FM 2021) Djoudi et al. Formal Verification of a JavaCard Virtual Machine with Frama-C

High-level (security) properties are hard to specify and verify in Frama-C

Examples of High-Level Properties

- A non-privileged user never reads a privileged (private) data page
- A privileged user never writes to a non-privileged (public) page
- The privilege level of a page cannot be changed unless...
- The privilege level of a user cannot be changed unless...
- A free page cannot be read or written, and must contain zeros
- Object data can be written only by the object owner
- Object data can be read only by the object owner

Such properties can be expressed as

- Constraints on reading / writing operations, calls to some functions,
- Strong or weak invariants

Solution: Metaproperties, or HILARE (High-Level ACSL Requirements)

We introduce meta-properties, which are a combination of:

- **A set of targets functions**, on which the property must hold.

```
foo          {foo, bar}          \ALL          \diff(\ALL, {foo, bar})
```

- **A context**, which characterizes the situation in which the property must hold.

```
\strong_invariant          \writing          \reading
```

- **An ACSL predicate**, expressed over the set of global variables.

```
A < B          *p == 0          \separated(\written, p)
```

```
meta \prop,  
  \name(A < B everywhere in foo and bar),  
  \targets({foo, bar}),  
  \context(\strong_invariant),  
  A < B;
```

Writing context: for integrity

- The given predicate must hold whenever the memory is modified.
- The predicate uses a predefined variable `\written` that refers to the written memory location.
- Typically, we specify that some variable `Var` is not written by `\separated(&Var, \written)`

Reading context: for confidentiality

- The given predicate must hold whenever the memory is read.
- The predicate uses a predefined variable `\read` that refers to the read memory location.
- Typically, we specify that some variable `Var` is not read by `\separated(&Var, \read)`

Examples of Metaproperties

```
meta \prop, \name(Do not write to lower pages outside free),  
  \targets(\diff(\ALL , {page_free})),  
  \context( \writing ),
```

```
\forall integer i; 0 <= i < MAX_PAGE_NB ==>  
\let p = pages + i;  
p->status == PAGE_ALLOCATED &&  
user_level > p->confidentiality_level ==>  
\separated(\written, p->data + (0.. PAGE_SIZE - 1));
```

```
meta \prop, \name(Free pages are never read),  
  \targets(\ALL),  
  \context( \reading ),
```

```
\forall integer i; 0 <= i < MAX_PAGE_NB &&  
pages[i].status == PAGE_FREE ==>  
\separated(\read, pages[i].data + (0 .. PAGE_SIZE - 1));
```

Example: Integrity Metaproperty Verified with MetAcsL – Writing context

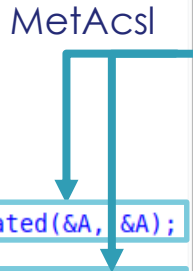
Resulting code after generating assertions with MetAcsL and proof with Frama-C/WP:

Initial C code:

```
/*@ meta "A_unchanged_unless";
*/
/*@ requires
  ensures
    (C >= 0
     C < 0
     assigns A,
*/
void foo(void)
{
  if (C >= 0) {
    /*@ check A_unchanged_unless: _1: meta: C < 0 -> \separated(&A, &A);
    A = C;
    /*@ check A_unchanged_unless: _2: meta: C < 0 -> \separated(&B, &A);
    B = C;
  }
  return;
}
```

If all instances are proved, the metaproperty is true

Contrary to an assert, a check is not kept in the proof context and does not overload the proof



```
test5.c
1 int A, B, C;
2 /*@
3 meta \prop, \name(A_unchanged_unless),
4 \targets(\ALL), \context(\writing),
5 C < 0 ==> \separated(\written, &A);
6 */
7 /*@
8 requires A==B;
9 assigns A,B;
10 ensures C>=0 && A==C && B==C ||
11 C<0 && A==\old(A) && B==\old(B); */
12 void foo(){
13 if ( C >= 0 ){
14 A = C;
15 B = C;
16 }
```

MetAcsL instantiates a metaproperty in all relevant locations

Common Criteria Certification for Integrated Circuits and Smart cards



EAL7
EAL6
EAL5
EAL4
EAL3
EAL2
EAL1

Formal verification
required for **EAL6/EAL7**



Meet High-security requirements of customers.

Chips in ID documents are tiny computers with embedded Operating System and applications.

We apply **deductive verification** with **Frama-C/WP** on a **C** implementation of a **Java Card Virtual Machine**.

Common Criteria : security policy and security mechanism

Firewall Security Aspect

*“The Firewall shall ensure controlled sharing of class instances, and **isolation of their data and code between packages** (that is, controlled execution contexts) as well as between packages and the JCRE context...”*

[Java Card System – Open Configuration Protection Profile – V3.1]

> Security problem

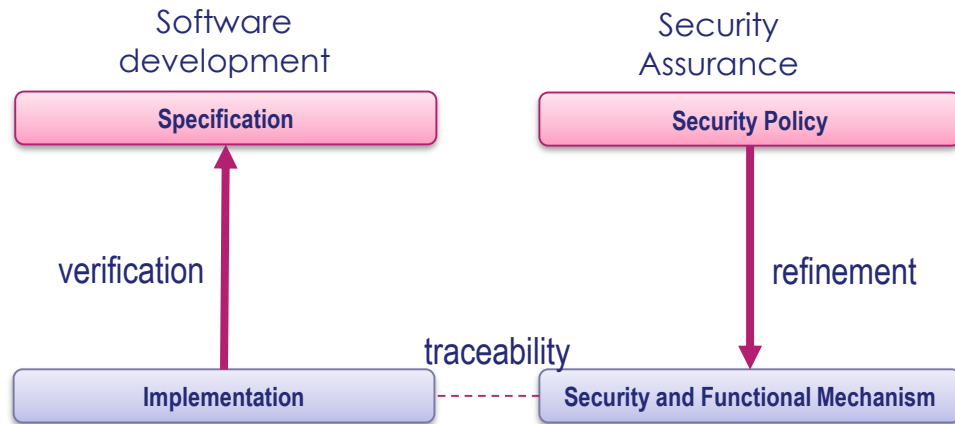
- Threats to Confidentiality and Integrity

> Security Objective

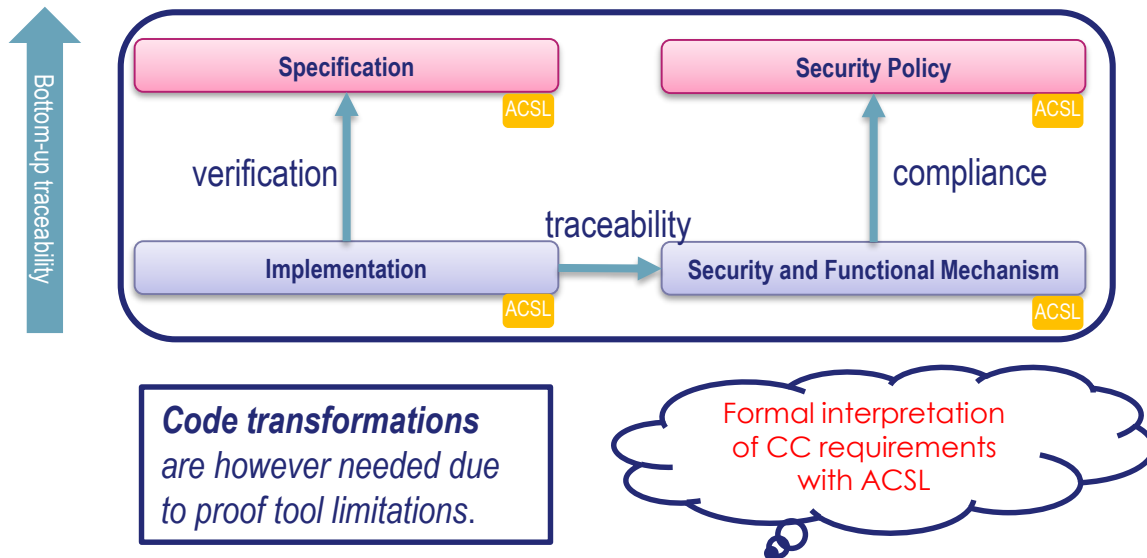
- Ensure isolation of data according to their owners

> Security Requirements

- Catalogue of security mechanisms related to the Firewall



Novel Bottom-up approach accepted by ANSSI : intrinsic refinement



Bottom-up approach **intrinsically** encompasses the **refinement** from the security and functional specifications through the design to the implementation.

(ERTS 2022) A Bottom-Up Formal Verification Approach for Common Criteria Certification: Application to JavaCard Virtual Machine (jointly with ANSSI and CEA-Leti, **best paper award**)

Bastore : function contract example

```
... /*@ admit requires bcv: valid_ref_or_null; // Admitted hypothesis without proof
103   requires vhm: valid_heap_model;
104   ensures  vhm: valid_heap_model;
105   ensures  oh: object_headers_intact{Pre, Post}; */
106 void aastore(u4 ObjRef, u4 DestOff, u1 Ref) { // u1/u4: unsigned char/int
107   if( ! firewall(ObjRef, DestOff) ) // Check access and
108     return; // exit if forbidden
109   if( GET_FLAG(ObjHeader+ObjRef) & 0x08 ) // If transient bit set,
110     TransData[GET_OFF(ObjHeader+ObjRef) + DestOff] = Ref; // write to transient body
111   else // Otherwise
112     PersiData[GET_OFF(ObjHeader+ObjRef) + DestOff] = Ref; // write to persistent body
113   updateJPC();
114 }
```



toy example

- **aastore**: write value **Ref** into a given array at a given offset
- **valid_heap_model** is maintained both as **pre-condition** and **post-condition**
- **Line 105** ensures security property **integrity_header**
- **Firewall** is called to check the access

Properties propagated up to the main dispatch loop and maintained as global loop invariants.



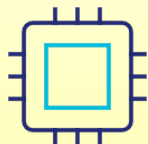
Further details in : Djoudi, A., Hána, M., Kosmatov, N., [Formal Verification of a JavaCard Virtual Machine with Frama-C](#). FM 2021.

Achievements



Formal Model Expressivity

- Functional properties
- Ghost code
- **Global security properties**



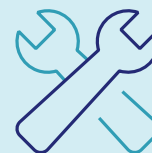
Formal Model Representativity

- Application on source code
- Limited code transformations
- **Full traceability**



Verification Efficiency

- C: 8K loc
- ACSL(user): 40k loc
- Proof Oblig.: 80k
- **Proof Scripts: 1k**
- **Proof time: 5h**



Verification tool automation

- Metaprop. : 50
- MetAcsl: 700k loc
- RTE: 5K loc
- **Strategies: 500**

Feedbacks from certification evaluations at Thales DIS (with CEA-Leti and ANSSI)

Good points

- **Straightforward correspondence from the security mechanisms to the formal model**
- The approach perfectly fits into the continuation of other tasks of the CC evaluation process
- Immediate understanding of formal entities (e.g. JCVm memory model)
- No refinement, thus no relation between multiple models to be evaluated
- The implementation challenges the formal model by construction

Points of attention

- **Code complexity directly transferred to the model**
- Sensitivity to tool scalability issues
- Organization of a high number of manual annotations



Ongoing and future work

More expressivity

- Handle more C features (eg. union types, setjmp/longjmp, function pointers)
- Extend supported ACSL features (e.g. statement contracts, \from clauses)
- Combine Frama-C/WP memory models to adapt to locally used C features

More automation

- Automatic generation of proof scripts is required for industrial usage
- CC documentation generation (traceability of security requirements)
- Need for an IDE dedicated to C/ACSL coding and proof debugging

More efficiency

- Enhance proof parallelization
- Enhance proof time profiling (especially for Qed simplifications)
- Allow partial proofs as needed while updating code and specification

Formal verification of **security properties** is mature and integrated into the software engineering process.

Tool enhancements are still needed to **facilitate daily usage** by specification and verification engineers.

Ongoing and Future Work, cont'd

Reasoning about metaproperties and other annotations can be helpful

- Sometime metaproperties can be deduces from other ones and ACSL annotations
- Preliminary ideas of deduction proposed in Virgile Robles' PhD thesis
- Externalizing verification of metaproperties at the callsite for two functions reduced proof time by 1 hour!!

Scaling to large programs

- Complex programs often have parts with many properties and with low-level operations
- Some of the maintained properties are irrelevant for some properties
- More abstract levels of reasoning can be helpful
- Combining deductive verification with abstract interpretation based tools [Bernier et al, FASE 2024]

Automatic generation of global properties from a high-level specification mechanism

- Express global properties in a dedicated domain-specific language
- Generate metaproperties from it
- Create a bridge between high-level and code-level artifacts

References

- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall. “MetAcsI: Specification and Verification of High-Level Properties.” **TACAS 2019**. Springer.
- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall. “Tame your annotations with MetAcsI: Specifying, Testing and Proving High-Level Properties”. **TAP 2019**. Springer.
- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall. “Methodology for Specification and Verification of High-Level Properties with MetAcsI”. **FormaliSE 2021**. IEEE.
- Adel Djoudi, Martin Hana and Nikolai Kosmatov. “Formal verification of a JavaCard virtual machine with Frama-C”. **FM 2021**. Springer.
- Adel Djoudi, Martin Hána, Nikolai Kosmatov, Milan Kříženecký, Franck Ohayon, Patricia Mouy, Arnaud Fontaine and David Féliot. “A Bottom-Up Formal Verification Approach for Common Criteria Certification: Application to JavaCard Virtual Machine”. **ERTS 2022, Best paper award**.
- Loïc Correnson, Allan Blanchard, Adel Djoudi and Nikolai Kosmatov. “Automate where Automation Fails: Proof Strategies for Frama-C/WP.” **TACAS 2024**. Springer. To appear.