A unified tool to fulfill semi formal and formal requirements for CC evaluations

Speakers

Carolina LAVATELLI (Trusted Labs)

Jean-Pierre KRIMM (CESTI-LETI)

7th ICCC

Lanzarote

September 19th-21st 2006

PLAN

- Generalities of the EDEN project
  - Purpose
  - Partners
- Developments
  - Methodology
  - Languages
  - Tools
- Formal requirements of CC v2.x fulfilled
  - Formal security assurance components
  - How are the formal requirements fulfilled
  - What should be the evaluator work
  - Extension to CC v3.1
- Conclusion

Purpose

Methodology and environment for the specification, verification and test that meet CC requirements based on semi-formal and formal descriptions

- Semi-formal and formal modelisation
- Consistency and completeness verifications
- Automatic test generation
- Traceability and documentation
- CC validation of methods and tools

French R&D projects

- **Eden 1: Formal and semi-formal verification of embedded systems components for high CC evaluation levels**
  - 30 months: Nov. 02 – May 05
  - Methodology and faisability
  - Prototypes of languages and tools
  - Validation against CC v2.x

- **Eden 2: Tools for the security evaluation of embedded systems components against CC v3 high evaluation levels**
  - 36 months: Dec. 05 – June 08
  - Evolution and stabilisation of Eden 1 developments
  - Methods and tools validation against CC v3
  - Goal: Industrial evaluation trial

Partners and roles

- Axalto
  - Needs from the industry
  - Experimentation
- CEA-LETI
  - CC validation of methods and tools
- CEA-LIST
  - Test generation from formal specifications
- Trusted Labs
  - Semi-formal and formal specification methodology
  - Experimentation
- Verimag
  - Formal specification methodology
  - Verification of (multiple) formal specifications

Methodology (1/2)

the needs

- Seamless integration to semi-formal descriptions
- Intuitive semantics for security/development teams
- Unified approach for all ADV representation levels

the approach

- Complementary to UML-like descriptions
- Operational descriptions
  Trace semantics
- TSF = what is done
  TSP = what is allowed

Methodology (2/2)

Operational descriptions
Trace semantics

TSF = what is done
TSP = what is allowed

Traces?

Sensitive events that arise during execution

Exemples

ACCESS to assets
USE of sec. funct.
FLOW of information

Modularity?

Each description defines its sensitive events

Exemples

TSF defines actual TOE events

TSP defines abstract events

Relationships ?

Event translation then trace inclusion

Exemples

TSF-1 doesn't do more than TSF-2

TSP allows all what TSF does

TSF and TSP specification

- Given observable events V raised by execution engine
- Specification S(E,P)
  - Observable entities E used to model/describe the system.
  - Collection P of properties of sequences of events in V*(E) .
- Semantics Traces(S(E,P))
  - Domain values for S
  - Traces in V *(E) , obtained by execution of S, that verify P
  - Traces(S(E,P)) $\subseteq$ Traces(S(E, $\varnothing$)) $\subseteq$ V *(E)

Relationship between specifications

- Given specifications $S_1(E_1,P_1)$ and $S2(E_2,P_2)$
- Correspondence from S_1 to S_2
  - Collection $R : V^*(E_1) \rightarrow V^*(E_2)$ of mappings $v_1 \rightarrow v_2$
  - Translation of $v1$ to the alphabet of S_1 yields v_2
- Conformance of S_1 to S_2 based on R
  - Domain values for S_1 and S_2
  - All execution traces of S_1, translated according to each mapping of R,, are included in the execution traces of S_2.
  - $Traces(S_1(E_1, P_1)) \subseteq_R Traces(S_2(E_2, P_2))$

Eden Specification Language (ESL)

TSP – ESLsec

- Most liberal security policy : state machine
- Legal (authorized) traces: additional temporal properties

TSF – ESLdev

- Java-like specification: classes, variables, methods, assertions
- Code instrumentation: READ, WRITE, CALL, EXIT… events

ESLsec

- State machine language
  - Variables: READ, WRITE events
  - Commands: CALL, EXIT, PASS, FAIL, … events
- Temporal properties on events
  - Mandatory sequence of events (e.g. «PASS(authentication) before PASS(transaction)»)
  - Forbidden sequence of events (e.g. « Never WRITE(x,_) »)
  - Chech properties (e.g. « WAIT(WRITE(y,val)); CHECK(val > 0) »)
  -
- TSP specification (Control and flow policies)
  - Operations: commands
  - Attributes: variables
  - Rules: properties, commands
  - Attribute Management: commands, properties

ESLsec: E-purse liberal policy and constraints

```
int pin;
int failure = 0;

void COMMAND_authentify(int code)
{
PASS(pin == code);{failure = 0};
FAIL(pin != code);{failure +=1};
}

void COMMAND_debit()
{
PASS(true);{};
}

void COMMAND_credit()
{
PASS(true);{};
}
```

```
String p= "PASS(authentify(_))";
String f= "FAIL(authentify(_))";
String d= "PASS(debit(_))";

void PROPERTY_P0() {
WAIT("p");
CHECK(ON_ENTRY(failure) < 3);
}


void PROPERTY_P1() {
ORDERED("p;NO(f) BEFORE d");
}
```

E-purse liberal/authorized traces

- [failure < 3 and **pin** = code] READ(**pin**); PASS(**authentify**(code)); PASS(**debit**());

PROPERTY_P1

- [failure < 3 and **pin** != code] READ(**pin**); FAIL(**authentify**(code)); PASS(**debit**());

PROPERTY_P0

- [failure = 3 and **pin** = code] READ(**pin**); PASS(**authentify**(code));

- [**pin** != code] READ(**pin**); FAIL(**authentify**(code)); PASS(**credit**());

PROPERTY_P1

- PASS(**debit**()); PASS(**credit**());

ESLdev

- Code instrumentation, e.g.

```
[type x = constant] → type x = constant;INIT("x",x)
[x := y] → READ("y",y); WRITE("x",x)
[f(x)] → READ("x",x); CALL(f(x)); ...; EXIT(f(x))
```

- Assertions, e.g.

```
REQUIRES( boolean_condition )
ENSURES ( boolean_condition )
```

- FSP specification
  - declarative style (« assignment » and « if-then-else »)
- HLD/LLD specification
  - interacting subsystems/modules (classes)

ESLdev: E-purse example

```
int pin;
int limit=3;
boolean auth = false;
int balance;

public check(int code) {
auth =false;
if (limit > 0) {
    limit = limit-1;
    if (pin == code) {
        limit = 3;
            auth = true;
    }
return;
}
```

```
public transaction(int x, boolean signus) {
if (signus == false) // debit
{
if (auth = true & x =< balance) {
          balance -= x};
     };
    else // credit
    { balance += x};
    };
    return;
    }
```

E-purse semantics

- [counter > 0 and **pin** =code] `CALL (check(code)); WRITE(auth,false); READ(pin); WRITE(auth,true); EXIT(check(code))`
- [counter > 0 and **pin** != code] `CALL (check(code)); WRITE(auth,false); READ(pin); EXIT(check(code));`
- [counter ≤ 0] `CALL(check(code)); WRITE(auth,false); EXIT(check(code));`
- [**balance** < x] `CALL(check(code)); … ; WRITE(auth, true); EXIT(check(code)); CALL(transaction(x,false)); READ(auth); EXIT(transaction(x,false));`
- [x ≤ **balance**] `CALL(check(code)); WRITE(auth,false); EXIT(check(code)); CALL(transaction(x,false)); EXIT(transaction(x,false));`
- [x ≤ **balance**] `CALL(check(code)); … ; WRITE(auth, true); EXIT(check(code)); CALL(transaction(x,false)); READ(auth); READ(balance); WRITE(balance); EXIT(transaction(x,false));`

Mappings R

E-purse TSF	E-purse TSP
pin	pin
balance	--
transaction(_,false)	debit();
transaction(_,true)	credit()
check(code)	authentify(code)
WRITE(auth,true);EXIT(check(code))	PASS(authentify(code))
WRITE(auth,false);EXIT(check(code))	FAIL(authentify(code))
READ(auth)	--
WRITE(auth)	--

$$\text{Traces(E-purse TSF)} \subseteq_R \text{Traces(E-purse TSP)}$$

Test generation

- Symbolic execution of ESL associated automatas
  - Test plan, test scripts, expected results
- Coverage of specifications:
  - All interfaces
  - All instructions
  - All behaviors
  - « Divide and Conquer » methodology

ESL Tools

- UML with constraints to ESL: **TL FIT +** [Trusted Labs]
- ESL to traces : **IF+** [Verimag]
- Trace translation and inclusion: **IF+** [Verimag]
- ESL to tests: **IF+** [Verimag] **Agatha+** [CEA-LIST]
- Goal: Eclipse™ environment

CC v2.x formal assurance components

- The scope
  - Disconnected from EAL, i.e. EAL 7 implies more than formal specifications
  - Only focuses on formal assurance components (semiformal covered)

- Formal assurance components
  - ADV_FSP.4 **Formal functional specification**
  - ADV_HLD.5 **Formal high-level design**
  - ADV_LLD.3 **Formal low-level design**
  - ADV_RCR.3 **Formal correspondence demonstration**
  - ADV_SPM.3 **Formal TOE security policy model**

- Assurance components which depend on ADV
  - ATE_FUN, ATE_COV and ATE_DPT

ADV formal requirements

- Requirements on formal deliveries:
  - be in a formal style (FSP, SPM, HLD, LLD)
  - be internally consistent (FSP, HLD, LLD)
  - where two levels of representation are formal, the proof of correspondence between these representations shall be formal (SPM, RCR)
  - the formal **proof** of correspondence between the TSP model and the functional specification shall show that all of the security functions in the functional specification are **consistent** and **complete** with respect to the TSP model (SPM)
  - for each adjacent pair of provided TSF representations, the analysis shall **prove** that all relevant security functionality of the more abstract TSF representation is **correctly** and **completely** refined in the less abstract TSF representation (RCR)

Formal style

- Definition [CC-3,§309]

  "*A **formal specification** is written in a notation based upon well-established mathematical concepts, and is typically accompanied by supporting explanatory (informal) prose. These mathematical concepts are used to define the syntax and semantics of the notation and the proof rules that support logical reasoning. The syntactic and semantic rules supporting a formal notation should define how to recognise constructs unambiguously and determine their meaning.*"

- These requirements are fulfilled by ESL, based on traces theory

Internal Consistency

- Definition [CC-3,§84]

  "*There are no apparent contradictions between any aspects of an entity. In terms of documentation, this means that there can be no statements within the documentation that can be taken to contradict each other.*"

  – **Consistency of a description (FSP, HLD, LLD)**

    Error $\notin$ States (Traces (APP))

  – **Consistency of a correspondence (RCR)**

    Traces (APP) $\subseteq$ Domaine (RCR)

    where RCR = {T1, …, Tn}
    Domaine (RCR) = traces recognized by {T1, …, Tn}
    = language recognized by (T1 | … | Tn)*

  – **Consistency of a SPM (not required by CC)**

    Traces (SPM) $\neq \varnothing$

Consistency between two models

- Definition [CC-3,§76]

  "*The term **consistent** describes a relationship between two or more entities, indicating that there are no apparent contradictions between these entities.*"

  - **Consistency between refinements of the product description is explained by**

    $LLD \subseteq_{LLD \rightarrow HLD} HLD \subseteq_{HLD \rightarrow FSP} FSP$
  - **Security policy is completely implemented if**

    $SPM \subseteq_{SPM \rightarrow APP} APP$
  - **An application respects a security policy if**

    $APP \subseteq_{APP \rightarrow SPM} SPM$

Tests (ATE)

- Taking advantage of formal representations (traces)
- An automatic tool which processes test scenarios
  - Providing test plans, test procedure descriptions and expected test results (ATE_FUN)
  - Guaranteeing the coverage of all external interfaces (ATE_COV)
  - Depending on which refinement level taken as input, guarantees that the TSF operates in accordance with this level (ATE_DPT)
  - Giving all evidence elements to the evaluator
- The limitation is the input

What should be the evaluator work

- Assumption
  - EDEN methodology and tools are validated
    - ⇒ all the formal requirements on ADV are fulfilled
    - ⇒ confidence on the coverage and depth of tests
- Verify the completeness and accuracy of the description of the TOE for each level of refinement
- Verify the completeness and accuracy of the policy
- Verify the completeness and accuracy of the mappings (RCR and SPM)
- The evaluator shall acquire the knowledge of the TOE

Extension to CC v3.1

Identical to v2.x

- The definition of a formal specification is the same
- Consistency and completeness between FSP and SPM
- Correctness and completeness between TDS and FSP

Different to v2.x

- No internal consistency check for FSP and TDS
- SPM: For all policies that are modelled, the model shall define security for the TOE and provide a formal proof that the TOE cannot reach a state that is not secure.

Conclusion

- These formal tools have been specifically developed for CC evaluations
- Gain assurance on EDEN methodology and tools
- Perform a trial evaluation encompasses formal requirements
  - EAL 4 augmented
  - EAL 7
  - …
- Contribution of the French scheme of evaluation
  - Propositions made in EDEN project have to be validated by at least one CB

Thank you for your attention

Contact

Carolina.Lavatelli@trusted-labs.fr

jean-pierre.krimm@cea.fr