Innovating Tool for Software Security
the Frama-C Framework

Julien Signoles
Software Reliability & Security Lab

Forum on Formal Methods and Cybersecurity
Toulouse – 2017/01/31
1. Frama-C
2. Value Analysis
3. Runtime Monitoring
Frama-C at a Glance

- extensible framework for C code verification [Kirchner et al @FAC’15]
- strong safety & security guarantees for critical applications
- open source, with close-source extensions
- both academic and industrial usage

http://frama-c.com
ACSL = ANSI/ISO C Specification Language

*lingua franca* of Frama-C’s analyzers:

- **verify** annotations
- **generate** annotations to be verified by other means

```c
/*@ assert \valid(p); */
/*@ assert y+1 != 0; */
x = *p / (y+1);
```
Example: Function Contract in ACSL

```c
/*@ predicate sorted{L}(int* a, int length) =
\forall integer i,j; 0 <= i <= j < length ==> a[i] <=a [j]; */

/*@ requires \valid(a+(0..length-1));
requires sorted(a,length);
requires length >= 0;

assigns \nothing;

behavior exists:
    assumes \exists integer i; 0 <= i < length && a[i] == key;
    ensures a[\result] == key;

behavior not_exists:
    assumes \forall integer i; 0 <= i < length ==> a[i] != key;
    ensures \result == -1;

complete behaviors;
disjoint behaviors; */

int search(int* a, int length, int key);
```
Frama-C, a Collection of Tools

Several tools inside a single platform

- **plug-in architecture à la Eclipse** [S. @F-IDE’15]
  - tools provided as plug-ins
    - 21 plug-ins in the open source distribution
    - a few outside open source plug-ins
    - close source plug-ins, either at CEA (about 20) or outside
  - plug-ins connected to a kernel
    - provides an uniform setting
    - provides general services
    - synthesizes useful information
    - analyzer combinations [Correnson & S. @FMICS’12]
Plug-in Gallery
1. Frama-C

2. Value Analysis

3. Runtime Monitoring
Value Analysis Overview

Domain of variations of variables of the program

- **abstract interpretation** [Cousot & Cousot @POPL’77]
- **automatic** analysis, but fine tuning required
- **correct** over-approximation (e.g. $x \in [1..99], \frac{1}{2}$)
- **alarms** for potential undefined behaviors
- may ensure absence of undefined behaviors
- **alarms** for potential **invalid ACSL annotations**
- **graphical interface** : display the domain of each variable at each program point
Plug-in Eva
Main Features

- Frama-C plug-in for Value Analysis
- generic analysis on the abstract domain
- allow combination of abstract domains and some inter-reductions of their states
- should be easy to add new domain
  - domains was added (e.g. Apron) and are still being added
the design relies on the separation between:

▶ values
  ▶ abstraction of the possible C values of an expression
  ▶ abstract transformers for arithmetic operators on expressions
  ▶ communication interface for abstract domains

▶ domains
  ▶ abstraction of the set of reachable states at a program point
  ▶ abstract transformers of states through statements
  ▶ can be queried for the values of some C expressions

[Bühler, Yakobowski and Blazy @VMCAI’17]
can we guarantee absence of defaults in large system-level code?

- scada systems of 100+ kloc of C code
- highest certification requirements (IEC60880 class 1)
- pinpoint the undefined behaviors and help investigate their cause
- structural properties on memory separation and cyclic behaviors
- 80% code coverage, 200 alarms
- [Ourghanlian in Nuclear Engineering and Technology, 2015]
1. Frama-C

2. Value Analysis

3. Runtime Monitoring
E-ACSL plug-in [Delahaye, Kosmatov & S. @SAC’13]

- program transformation which generates an inline monitor
- verify annotations at runtime
- input : program $P$ annotated with (E-)ACSL
- output : program $P'$ which
  - is equivalent to $P$ for valid inputs; otherwise
  - fails on the first violated annotation

```c
char succ ( char x) {
 /*@ assert x+1 <= 127; */
  return x +1;
}
```
E-ACSL plug-in [Delahaye, Kosmatov & S. @SAC’13]

- program transformation which generates an inline monitor
- verify annotations at runtime

Input: program \( P \) annotated with (E-)ACSL

Output: program \( P' \) which
- is equivalent to \( P \) for valid inputs; otherwise
- fails on the first violated annotation

```c
char succ(char x) {
    /*@ assert x+1 <= 127; */
    return x+1;
}
```

```c
char succ(char x) {
    /*@ assert x+1 <= 127; */
e_acsl_assert(x+1 <= 127);
    return x+1;
}
```
Runtime Verification with E-ACSL

Applications

Monitoring

- undefined behaviors with plug-in RTE
  - memory accesses
  - arithmetic overflows
- complex functional properties
- memory consumption
- correct API usage with plug-in Aoraï
- information flow issue with plug-in SecureFlow
information flow analysis: verify absence of leakage of sensitive data (termination insensitive non interference)

[Assaf et al @SEC’13, Barany @VPT’16]
Frama-C is also a tool of choice for your security analyses

- **TIS-analyzer** and **TIS-interpreter** [TrustInSoft]
- **taint analysis** [Ceara, Mounier and Potet @ICSTW’10]
- **source code model for physical attacks on smart cards** [Berthomé et al @PLAS’10]
- **fault injection** in implementations of cryptographic protocols [Christofi’s PhD thesis, 2013]
- **secure generative programming** [Anin and Rompf @POPL’17]
- **security counter-measures** based on static detection and runtime assertion checking of CVEs [Pariente & S., submitted]
Conclusion

- **Frama-C**, collaborative framework for C code verification
  - open and extensible framework
  - both academic and industrial usage

- **Eva** (Frama-C plug-in)
  - verify the absence of undefined behaviors
  - may help prevent security flaws

- **E-ACSL** (Frama-C plug-in)
  - runtime verification
  - bug finding tool
  - verify more properties than testing or other dynamic tools

- **SecureFlow** (Frama-C plug-in)
  - verify non-interference
  - in combination with Eva and/or E-ACSL
Selected Publications

http://frama-c.com

- F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski.

- D. Bühler, B. Yakobowski, and S. Blazy.
  *Structuring Abstract Interpreters through State and Value Abstractions* VMCAI’17.

- N. Kosmatov, and J. Signoles.
  *A lesson on runtime assertion checking with Frama-C (tutorial)*. RV’2013.

- M. Assaf, J. Signoles, É. Totel, and F. Tronel.
  *Program transformation for non-interference verification on programs with pointers*. SEC’2013.