UFR IM²AG
UNIVERSITÉ Grenoble Alpes

Grenoble INP Ensimag

Software security, secure programming

A brief introduction to Frama-C

Master M2 Cybersecurity

Academic Year 2021 - 2022

The Frama-C plateform

An open-source collaborative plateform for the analysis of C programs
`http://frama-c.com/index.html`

- developed by the CEA List and INRIA Saclay
- offers an integrated set of code analysis plug-ins:
  - runtime-error detection (RTE)
  - value analysis (VSA)
  - dependency analysis and slicing
  - control-flow-grah and call-graph computations
  - property proof using weakest preconditions computations (WP)
  - etc.

→ we are going to use essentially RTE, VSA, and (possibly) WP ...

Some reminders about Value-Analysis[1]

Goal: staticaly compute an (over-approximated !) set of values, for each variable, at each program location

Principle

Abstract Interpretation

- “compute” the program behavior using an **abstract semantics** (using abstract domains of values and abstract operations) as an iterative fix-point computation
- loop termination enforced/accelerated using widening & narrowing operators (over-approximate the loop behavior)

Outcomes

- help to detect potential runtime errors (arithmetic overflow, invalid memory access, etc.)
- may produce **false positives** (i.e., non existing bugs) when the over-approximation is too coarse . . .

[1] (see previous lectures for more details !)

Using Frama-C

Through its graphical user interface:

`frama-c-gui example.c`

or, to produce **runtime error assertions:**

`frama-c-gui -rte example.c`

or, to run **value analysis (VSA):**

`frama-c-gui -eva example.c`

These plugins can also be accessed through the `Analyses` menu:

(`Rtegen`, `Value analysis` and `WP`)

Using Frama-C

Through its graphical user interface:

`frama-c-gui example.c`

or, to produce **runtime error assertions:**

`frama-c-gui -rte example.c`

or, to run **value analysis (VSA):**

`frama-c-gui -eva example.c`

These plugins can also be accessed through the `Analyses` menu:

(`Rtegen`, `Value analysis` and `WP`)

A possible workflow (for each example):

1. Generate the runtime assertions (Rtegen)
   → verify that you understand them . . .

Using Frama-C

Through its graphical user interface:

`frama-c-gui example.c`

or, to produce **runtime error assertions:**

`frama-c-gui -rte example.c`

or, to run **value analysis (VSA):**

`frama-c-gui -eva example.c`

These plugins can also be accessed through the `Analyses` menu:

(`Rtegen`, `Value analysis` and `WP`)

A possible workflow (for each example):

1. Generate the runtime assertions (Rtegen)
   $\rightarrow$ verify that you understand them ...
2. Run the value analysis
   $\rightarrow$ verify that you understand the results
   Why some (obvious ?) assertions may not be validated ?

Using Frama-C

Through its graphical user interface:

`frama-c-gui example.c`

or, to produce **runtime error assertions:**

`frama-c-gui -rte example.c`

or, to run **value analysis (VSA):**

`frama-c-gui -eva example.c`

These plugins can also be accessed through the `Analyses` menu:

(`Rtegen`, `Value analysis` and `WP`)

A possible workflow (for each example):

1. Generate the runtime assertions (Rtegen)
   $\rightarrow$ verify that you understand them ...
2. Run the value analysis
   $\rightarrow$ verify that you understand the results
   Why some (obvious ?) assertions may not be validated ?
3. If you thing the code is incorrect/unsecure, try to strengthen it and goto 1

Using Frama-C

Through its graphical user interface:

```
frama-c-gui example.c
```

or, to produce **runtime error assertions:**

```
frama-c-gui -rte example.c
```

or, to run **value analysis (VSA):**

```
frama-c-gui -eva example.c
```

These plugins can also be accessed through the `Analyses` menu:

(`Rtegen`, `Value analysis` and `WP`)

A possible workflow (for each example):

1. Generate the runtime assertions (Rtegen)
   → verify that you understand them . . .
2. Run the value analysis
   → verify that you understand the results
   Why some (obvious ?) assertions may not be validated ?
3. If you thing the code is incorrect/unsecure, try to strengthen it and goto 1
4. Otherwise, if you think the code is correct:
   - try to add some extra assertions (and loop invariants ?)
   - optionally, try to use `WP` to prove them ?
   - re-run the VSA with these new assertions . . .

The value analysis plug-in

(Evolved) Value Analysis

- Based on Abstract Interprattion to compute abstract variable domains
- Fully automated, but can be user-guided through ACSL annotations
- mainly used to discharge runtime-error asssertions (RTE), but internaly used by other plugins . . .

Some practical informations

- abstract domains = value sets and intervals (non relational domains)
- controlling approximations (*time vs memory*)
  - syntactic loop unrolling (`-ulevel`)
  - semantic unrolling (`-slevel`)
    $\rightarrow$ useful when widenning operators are too coarse
  - adding ACSL loop invariants, or extra assertions . . .

WP - Expressing assertions with ACSL

Ansi-C Specification Language

- first order logic
- use C types (int, float, pointers, arrays, etc.) + Z + R
- built-in predicates for memory access: `valid, separated`
  $\rightarrow$ allows to express memory-level requirements (beyond the C semantics)
- used as special comments:

  `/*@ .......... */`

$\Rightarrow$ have a look to the short tutorial:

`http://frama-c.com/acsl_tutorial_index.html`

Example of assertion

- valid memory access:
  `\valid(a)` means that address `a` refers to
  a memory location **correctly allocated** (w.r.t. the C type of `a`)

```
     \valid(p)
     \valid(t+i)
 \valid(t+)(0..n-1)
```

- pre- and post- conditions

```
\requires x<= n && \valid(t+x)
\ensures (t+x) = x
```

- loop invariants, assertions

```
loop invariant z==x+y
assert x>=0
```

- etc.

Lab Session

Objective:

Evaluate the strengths and weaknesses of static analysis tools (like Frama-C) for source-level vulnerability detection . . .

1. Play with the examples/exercices provided in the course web page . . .
2. You can also check if the vulnerabilities in the C files of Lab session 1 are detected by Frama-C ?