International Cybersecurity Forum

SECURITY AND PRIVACY BY DESIGN

Europe kicks off !

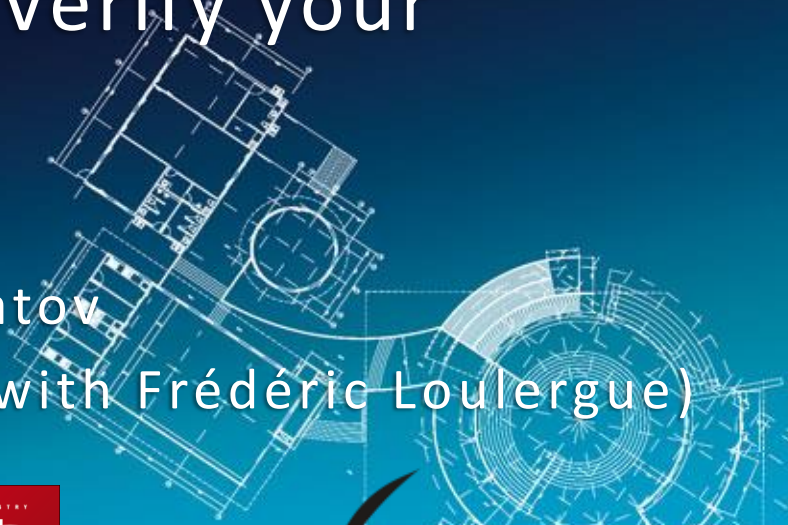
Lille Grand Palais

22nd and 23rd January 2019

Cybersecurity for IoT: Verify your Software Today!

Allan Blanchard, Nikolai Kosmatov

(based on a tutorial prepared with Frédéric Loulergue)



Inria inventors for the digital world

CEA tech List

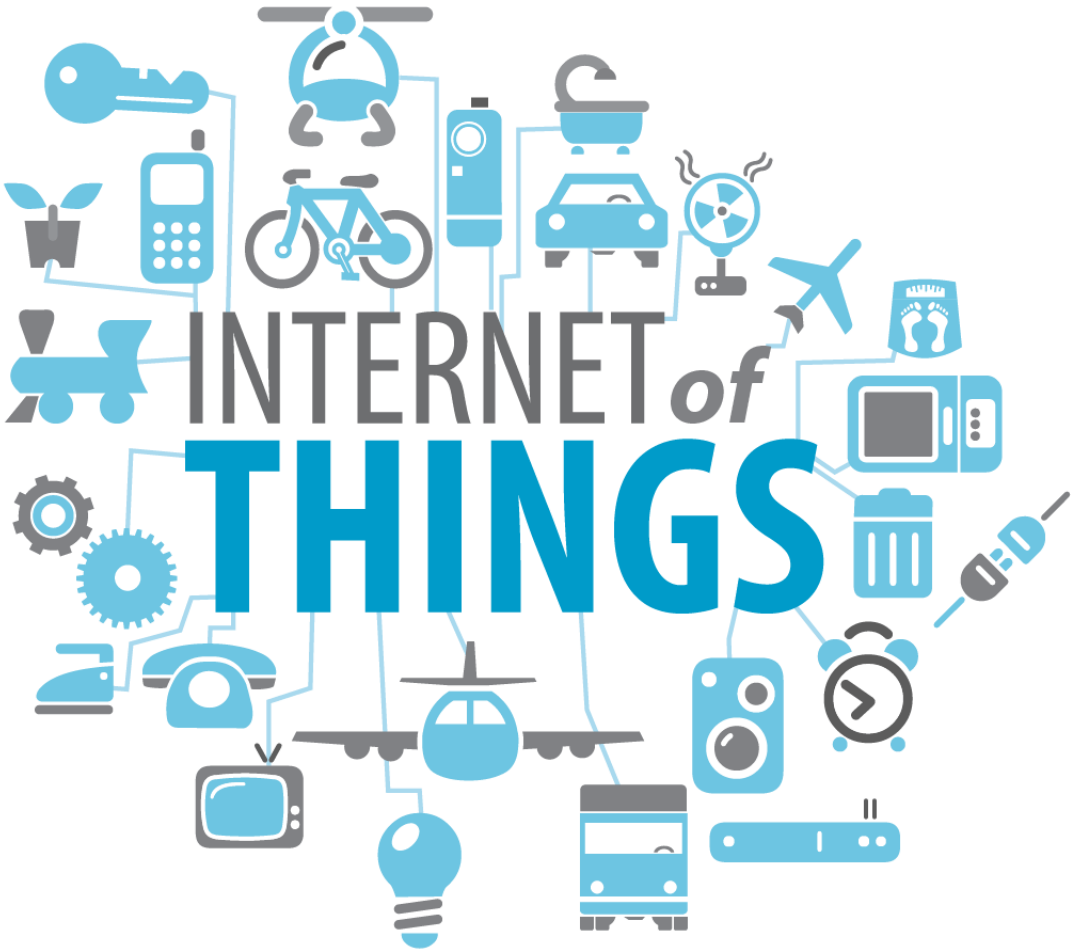
essedia

forum-fic.com

Outline

Outline

Internet of Things



- connect all devices and services
- 46 billions devices by 2021
- transport huge amounts of data

(c) Internet Security Buzz

And Security?

And Security?

By Waqas on October 22, 2016 Email @hackread CYBER ATTACKS MALWARE SECURITY

And Security?

By Waqas on October 22, 2016 Email @hackread CYBER ATTACKS MALWARE SECURITY

ANDY GREENBERG SECURITY 07.21.15 06:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

And Security?

By Waqas on October 22, 2016 Email @hackread CYBER ATTACKS MALWARE SECURITY

by Tom Spring August 26, 2016 , 2:55 pm

ANDY GREENBERG SECURITY 07.21.15 06:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

And Security?

By Waqas on October 22, 2016 Email @hackread CYBER ATTACKS MALWARE SECURITY

ANDY GREENBERG SECURITY 07.21.15 06:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

by Tom Spring August 26, 2016 , 2:55 pm

Hacking a computer-aided sniper rifle

Elizabeth Weise | USATODAY
Published 5:56 p.m. UTC Aug 7, 2015

Outline

Frama-C Open-Source Distribution

Framework for Analysis of C source code

http://frama-c.com

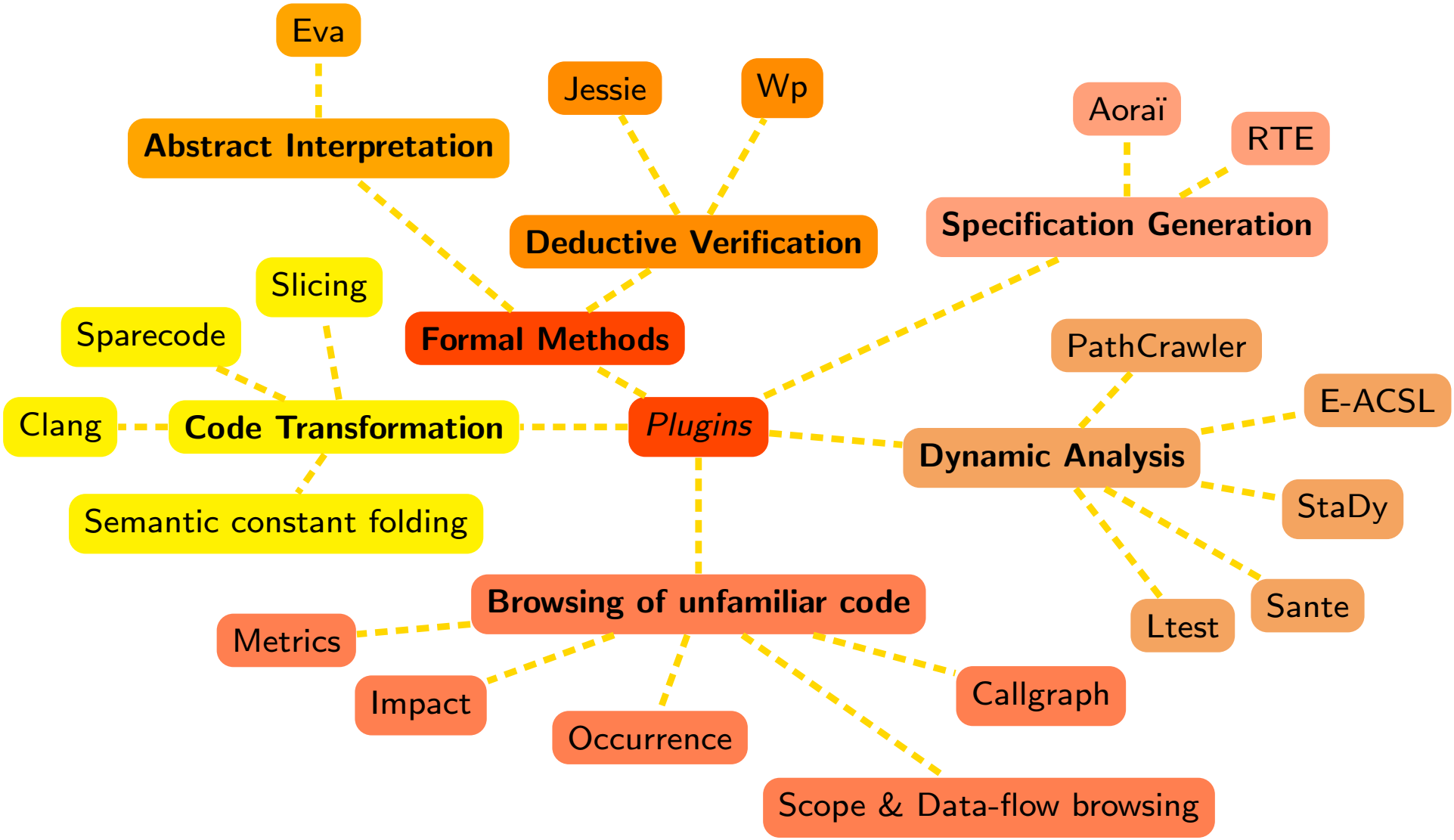
- offers a specification language called ACSL
- targets both academic and industrial usage

Frama-C, a Collection of Tools

Several tools inside a single platform

- plugin architecture like in Eclipse
  - over 20 plugins in the open-source distribution
  - also close-source plugins, either at CEA (about 20) or outside
- a common kernel
  - provides a uniform setting
  - provides general services

Plugin Gallery



Use the Right Tool for the Right Task

We may want to assure different degrees of confidence:

- absence of runtime errors or functional correctness
- partial/complete analysis (testing vs. verification)

Different tools require from us more or less work:

- Just provide the source code
- Configure tool parameters
- Provide code annotations

The higher the confidence is, the more information we have to provide

Outline

A lightweight OS for IoT

Contiki is a lightweight operating system for IoT

It provides a lot of features:

- (rudimentary) memory and process management
- networking stack and cryptographic functions
- ...

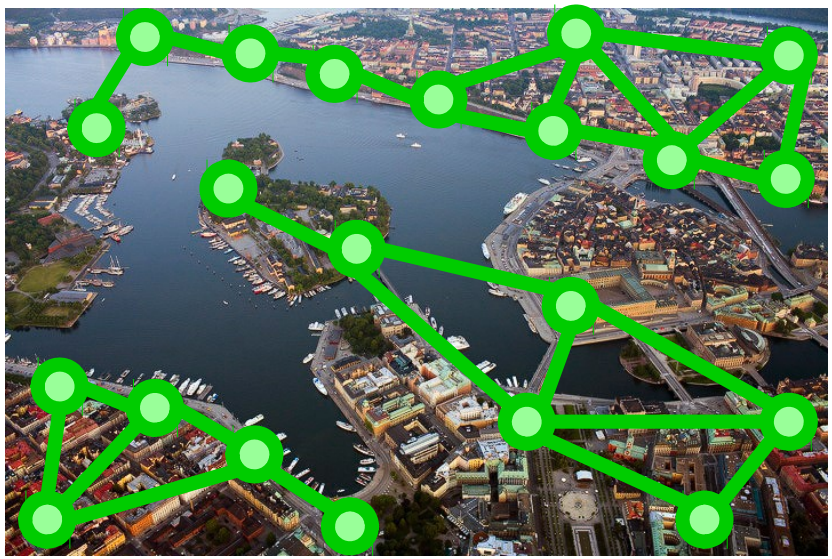
Typical hardware platform:

- 8, 16, or 32-bit MCU (little or big-endian),
- low-power radio, some sensors and actuators, ...

Note for security: there is *no* memory protection unit.

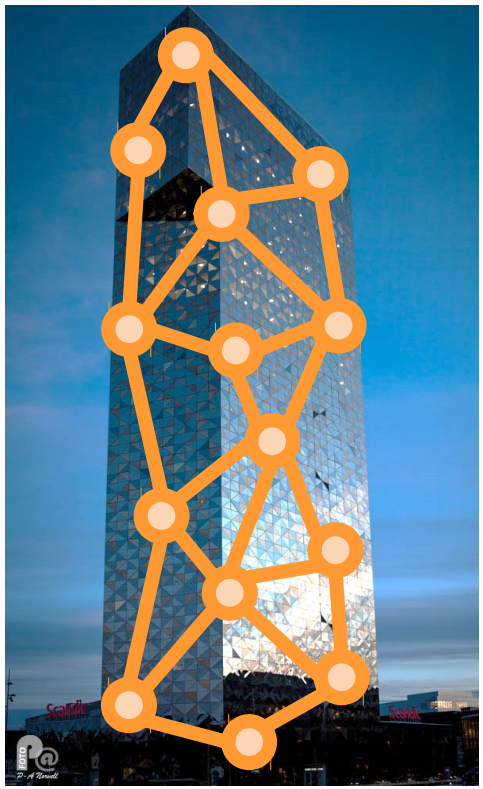
Contiki: Typical Applications

- IoT scenarios: smart cities, building automation, ...
- Multiple hops to cover large areas
- Low-power for battery-powered scenarios
- Nodes are interoperable and addressable (IP)



Traffic lights
Parking spots
Public transport
Street lights
Smart metering
...

Light bulbs
Thermostat
Power sockets
CO2 sensors
Door locks
Smoke detectors
...



Outline

Runtime errors

Runtime errors in C are undefined behaviors:

- out-of-bound accesses,
- integer overflows,
- division by 0,
- invalid pointers
- ...

They can raise important security issues

- For example, HeartBleed vulnerability (found in 2014 in OpenSSL)

Value Analysis Overview

Compute possible values of variables at each program point

- an automatic analysis based on abstract interpretation
- computes a correct over-approximation
- reports alarms for potential runtime errors
- reports alarms for potentially invalid annotations
- can prove the absence of runtime errors
- graphical interface: displays the domains of each variable

Outline

Example 1

Run Eva: `frama-c-gui div1.c -val -main=f`

```
int f ( int a ) {
  int x, y;
  int sum, result;
  if(a == 0){
    x = 0; y = 0;
  }else{
    x = 5; y = 5;
  }
  sum = x + y; // sum can be 0
  result = 10/sum; // risk of division by 0
  return result;
}
```

Example 1

Run Eva: `frama-c-gui div1.c -val -main=f`

```
int f ( int a ) {
  int x, y;
  int sum, result;
  if(a == 0){
    x = 0; y = 0;
  }else{
    x = 5; y = 5;
  }
  sum = x + y; // sum can be 0
  result = 10/sum; // risk of division by 0
  return result;
}
```

Risk of division by 0 is detected, it is real.

Example 2

Run Eva: `frama-c-gui div2.c -val -main=f`

```
int f ( int a ) {
  int x, y;
  int sum, result;
  if(a == 0){
    x = 0; y = 5;
  }else{
    x = 5; y = 0;
  }
  sum = x + y; // sum cannot be 0
  result = 10/sum; // no div. by 0
  return result;
}
```

Example 2

Run Eva: `frama-c-gui div2.c -val -main=f`

```
int f ( int a ) {
  int x, y;
  int sum, result;
  if(a == 0){
    x = 0; y = 5;
  }else{
    x = 5; y = 0;
  }
  sum = x + y; // sum cannot be 0
  result = 10/sum; // no div. by 0
  return result;
}
```

Risk of division by 0 is detected, but it is a false alarm.

Eva Parameterization

- Eva is automatic, but can be imprecise due to over-approximation
- a fine-tuned parameterization for a trade-off precision / efficiency

Example 2, cont'd

Run Eva: `frama-c-gui div2.c -val -main=f -slevel 2`

```
int f ( int a ) {
  int x, y;
  int sum, result;
  if(a == 0){
    x = 0; y = 5;
  }else{
    x = 5; y = 0;
  }
  sum = x + y; // sum cannot be 0
  result = 10/sum; // no div. by 0
  return result;
}
```

Example 2, cont'd

Run Eva: `frama-c-gui div2.c -val -main=f -slevel 2`

```
int f ( int a ) {
  int x, y;
  int sum, result;
  if(a == 0){
    x = 0; y = 5;
  }else{
    x = 5; y = 0;
  }
  sum = x + y; // sum cannot be 0
  result = 10/sum; // no div. by 0
  return result;
}
```

Absence of division by 0 is proved, no false alarm.

Outline

Overview of the aes-ccm Modules

- Critical! – Used for communication security
  - end-to-end confidentiality and integrity
- Advanced Encryption Standard (AES): a symmetric encryption algo.
  - AES replaced in 2002 Data Encryption Standard (DES)
- Modular API – independent from the OS
- Two modules:
  - AES-128
  - AES-CCM* block cypher mode
  - A few hundreds of LoC
- High complexity crypto code
  - Intensive integer arithmetics
  - Intricate indexing
  - based on multiplication over finite field $\mathrm{GF}(2^8)$

Example 3

We analyze two versions of a part of the aes module

- `frama-c-gui aes1.c -val`
- `frama-c-gui aes2.c -val`

Outline

Functional properties

With Eva, we can prove that no bad things can happen

Can we go further and prove that good things will eventually happen?

Functional properties

With Eva, we can prove that no bad things can happen

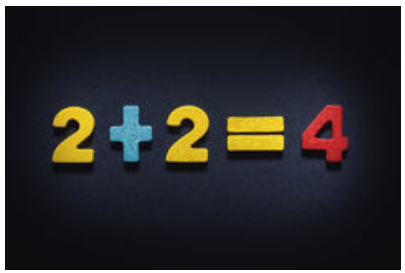
Can we go further and prove that good things will eventually happen?

Yes!

- we have to define what we mean by "good things"
- we still have to show that no bad things happen

Legend: Bad things = runtime errors, good things = expected behavior

Objectives of Deductive Verification



Rigorous, mathematical proof of semantic properties of a program

- functional properties
- absence of runtime errors
- termination

Requires some extra work from us to define the expected properties...

WP plugin

- Modular deductive verification (function by function)
- Input: a program and its specification written in ACSL
- If the proof succeeds, the program respects the given specification
  - Does it mean that the program is correct?

WP plugin

- Modular deductive verification (function by function)
- Input: a program and its specification written in ACSL
- If the proof succeeds, the program respects the given specification
  - Does it mean that the program is correct?
  - NO! If the specification is wrong, the program can be wrong!

Function contracts

- Goal: specification of imperative functions
- Approach: give assertions (i.e. properties) about the functions
  - Precondition is supposed to be true on entry (ensured by the caller)
  - Postcondition must be true on exit (ensured by the function)
- Nothing is guaranteed when the precondition is not satisfied

Example 1

Run WP: `frama-c-gui -wp -wp-rte all_zeros.c`

```
/*@ requires n>=0 && \valid(t+(0..n-1));
    assigns \nothing;
    ensures \result != 0 <==>
      (\forall integer j; 0 <= j < n ==> t[j] == 0);
*/
int all_zeros(int t[], int n) {
  int k;
  /*@ loop invariant 0 <= k <= n;
      loop invariant \forall integer j; 0<=j<k ==> t[j]==0;
      loop assigns k;
      loop variant n-k;
  */
  for(k = 0; k < n; k++)
    if (t[k] != 0)
      return 0;
  return 1;
}
```

Outline

Overview of the `memb` Module

- No dynamic allocation in Contiki
  - to avoid fragmentation of memory in long-lasting systems
- Memory is pre-allocated (in arrays of blocks) and attributed on demand
- The management of such blocks is realized by the `memb` module

The `memb` module API allows the user to

- initialize a `memb` store (i.e. pre-allocate an array of blocks),
- allocate or free a block,
- check if a pointer refers to a block inside the store
- count the number of allocated blocks

We specified and verified the memb module with Frama-C/WP

The *textual* contract of memb_alloc

1. If the store is full, then leave it intact and return NULL (lines 12–15)
2. If the store has a free block, then return a free block b such that:
   - b is properly aligned in the block array (line 8)
   - b was marked as free, and is now marked as allocated (line 7)
   - b is valid, i.e. points to a valid memory space of a block size that can be safely read or written to (line 10)
   - the states of the other blocks have not changed (line 9)
   - the number of free blocks is decremented (line 11)

The contract of memb_alloc

```
/*@
  requires valid_memb(m);
  ensures valid_memb(m);
  assigns m->count[0 .. (m->num − 1)];
  behavior free_found:
    assumes ∃ℤi; 0 ≤ i <m->num ∧m->count[i] ==0;
    ensures ∃ℤi; 0 ≤ i <m->num ∧\old(m->count[i]) ==0 ∧m->count[i] ==1 ∧
      \result ==(char*) m->mem + (i * m->size) ∧
      ∀ℤj; (0 ≤ j <i ∨i <j <m->num) ⇒m->count[j] ==\old(m->count[j]);
    ensures \valid((char*) \result + (0 .. (m->size − 1)));
    ensures _memb_numfree(m) ==\old(_memb_numfree(m)) − 1;
    ensures _memb_allocated(m, \result);
  behavior full:
    assumes _memb_full(m);
    ensures ∀ℤi; 0 ≤ i <m->num ⇒m->count[i] ==\old(m->count[i]);
    ensures _memb_numfree(m) ==\old(_memb_numfree(m));
    ensures \result ==NULL;
  complete behaviors;
  disjoint behaviors;
*/
void *memb_alloc(struct memb *m);
```

Other modules of Contiki analyzed with WP

Absence of security vulnerabilities coming from runtime errors :

- for several low-level modules of the core part of Contiki

Functional verification of the list module:

- a buggy function found and fixed
- different verification techniques studied and compared

Outline

Completeness

A complete static analysis (for all inputs) can be hard and costly

A partial, dynamic analysis (for selected inputs) is usually easier

Objectives of E-ACSL

- E-ACSL is a runtime assertion checking tool
  - detect runtime errors
  - detect annotation failures
  - treat a concrete program run (i.e. with concrete inputs)

E-ACSL plugin at a Glance

http://frama-c.com/eacsl.html

- Main idea: convert annotations into C code

```
int div(int x, int y) {
  /*@ assert y-1 != 0; */
  return x / (y−1);
}
```

E-ACSL →

```
int div(int x, int y) {
  /*@ assert y-1 != 0; */
  e_acsl_assert(y-1 != 0);
  return x / (y−1);
}
```

E-ACSL plugin at a Glance

http://frama-c.com/eacsl.html

- Main idea: convert annotations into C code

```
int div(int x, int y) {
  /*@ assert y-1 != 0; */
  return x / (y−1);
}
```

E-ACSL →

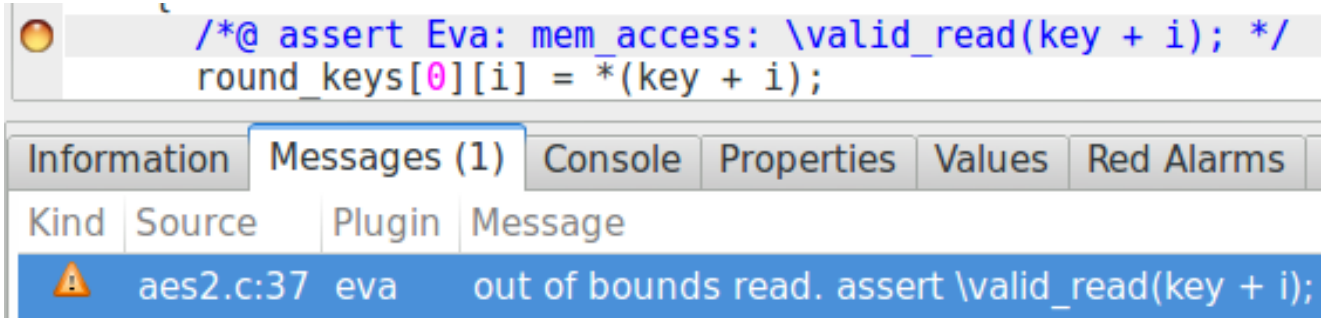
```
int div(int x, int y) {
  /*@ assert y-1 != 0; */
  e_acsl_assert(y-1 != 0);
  return x / (y−1);
}
```

- The real translation is more complex than it may look

Outline

E-ACSL applied to the AES module

Remember our previous analysis on aes2.c ...



We can check this at runtime:

```
$ e-acsl-gcc.sh aes2.c --rte=all -c -Omonitored-aes2
$ ./monitored-aes2
$ ./monitored-aes2.e-acsl
Assertion failed at line 37 in function aes_128_set_key.
The failing predicate is:
rte: mem_access: \valid_read(key + i).
Abandon (core dumped)
```

Possible Usages in Combination with Other Tools

- check properties unproved by static analyzers (e.g. Eva, WP)
- check the absence of runtime errors
- check memory consumption and violations (use-after-free)
- help testing tools to check properties which are not easy to observe

Outline

Conclusion

Frama-C allows us to:

- verify the absence of runtime errors with Eva
- formally specify functional properties with ACSL
- prove a program respects its specification with WP
- verify annotations at runtime or detect runtime errors with E-ACSL

All of these and much more inside Frama-C

Conclusion

IoT software is critical

- Connected devices are used in many critical domains today
- Their usage is rapidly expanding

Formal verification tools can be helpful

- Verification tools have become more efficient in practice: faster hardware, more memory...
- Formal methods are successfully used in several critical domains (avionics, energy, rail,...)
- Applying formal methods improves software quality in 92% of projects

  Source: Formal Methods Practice and Experiments, ACM Comp.Surveys

Verify your IoT software today!

Further reading

User manuals:

- user manuals for Frama-C and its different analyzers, on the website:
  `http://frama-c.com`

About the use of WP:

- Introduction to C program proof using Frama-C and its WP plugin
  Allan Blanchard
  `https://allan-blanchard.fr/publis/frama-c-wp-tutorial-en.pdf`
- ACSL by Example
  Jochen Burghardt, Jens Gerlach
  `https://github.com/fraunhoferfokus/acsl-by-example`

Further reading

Tutorial papers:

- A. Blanchard, N. Kosmatov, and F. Loulergue. A Lesson on Verification of IoT Software with Frama-C (HPCS 2018)
- on deductive verification:
  N. Kosmatov, V. Prevosto, and J. Signoles. A lesson on proof of programs with Frama-C (TAP 2013)
- on runtime verification:
  - N. Kosmatov and J. Signoles. A lesson on runtime assertion checking with Frama-C (RV 2013)
  - N. Kosmatov and J. Signoles. Runtime assertion checking and its combinations with static and dynamic analyses (TAP 2014)
- on test generation:
  N. Kosmatov, N. Williams, B. Botella, M. Roger, and O. Chebaro. A lesson on structural testing with PathCrawler-online.com (TAP 2012)
- on analysis combinations:
  N. Kosmatov and J. Signoles. Frama-C, A collaborative framework for C code verification: Tutorial synopsis (RV 2016)

Further reading

On the verification of Contiki:

- on the MEMB module:
  F. Mangano, S. Duquennoy, and N. Kosmatov. A memory allocation module of Contiki formally verified with Frama-C. A case study (CRiSIS 2016)
- on the AES-CCM* module:
  A. Peyrard, S. Duquennoy, N. Kosmatov, and S. Raza. Towards formal verification of Contiki: Analysis of the AES–CCM* modules with Frama-C (RED-IoT 2017)
- on the LIST module:
  - A. Blanchard, N. Kosmatov, and F. Loulergue. Ghosts for lists: A critical module of contiki verified in Frama-C (NFM 2018)
  - F. Loulergue, A. Blanchard, and N. Kosmatov. Ghosts for lists: from axiomatic to executable specifications (TAP 2018)
  - A. Blanchard, N. Kosmatov, and F. Loulergue. Logic against Ghosts: Comparison of Two Proof Approaches for a List Module (SAC 2019)