Applications of formal verification for secure Cloud environments at CEA LIST

Nikolai Kosmatov

joint work with A.Blanchard, F.Bobot, M.Lemerre,. . .

SEC2, Lille, June 30th, 2015

Outline

Outline

Frama-C, a brief history

- 90's: CAVEAT, Hoare logic-based tool for C code at CEA
- 2000's: CAVEAT used by Airbus during certification process of the A380 (DO-178 level A qualification)
- 2008: First public release of Frama-C (Hydrogen)
- 2012: New Hoare-logic based plugin WP developed at CEA LIST
- Today: Frama-C Sodium (v.11)
  - Multiple projects around the platform
  - A growing community of users...
  - and of plugin developers

Frama-C at a glance

Software Analyzers

- A Framework for Modular Analysis of C code
- Developed at CEA LIST and INRIA Saclay
- Released under LGPL license
- Kernel based on CIL [Necula et al. (Berkeley), CC 2002]
- ACSL annotation language
- Extensible plugin oriented platform
  - Collaboration of analyses over same code
  - Inter plugin communication through ACSL formulas
  - Adding specialized plugins is easy
- `http://frama-c.com/` [Cuoq et al. SEFM 2012, FAC 2015]

ACSL: ANSI/ISO C Specification Language

- Based on the notion of contract, like in Eiffel, JML
- Allows users to specify functional properties of programs
- Allows communication between various plugins
- Independent from a particular analysis
- Manual at `http://frama-c.com/acsl`

Basic Components

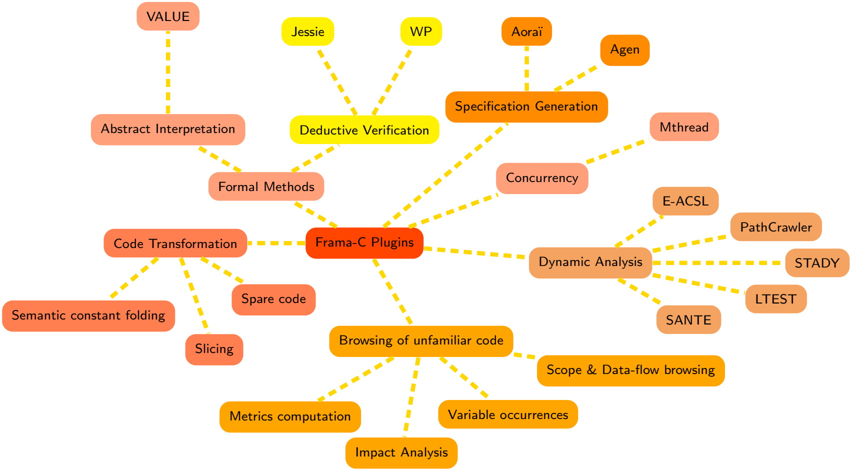
- First-order logic
- Pure C expressions
- C types + $\mathbb{Z}$ (integer) and $\mathbb{R}$ (real)
- Built-in predicates and logic functions

Example: a C program annotated in ACSL

```
/*@ 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;
}
```

Can be proven in Frama-C/WP

Main Frama-C plugins



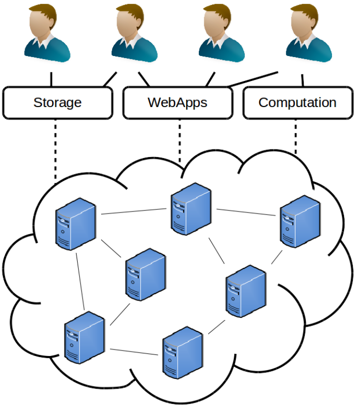
Plugin WP for deductive verification

- Based on Weakest Precondition calculus [Dijkstra, 1976]
- Proves that a given program respects its specification
- Relies on
  - automatic provers (Alt-Ergo, CVC4, Z3, ...)
  - when necessary, interactive proof assistants (Coq)

Outline

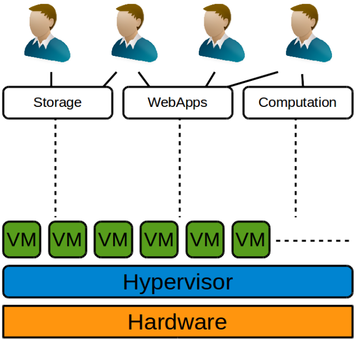
Anaxagoros Microkernel

- Clouds mutualize physical resources between users
  - Safety and security are crucial



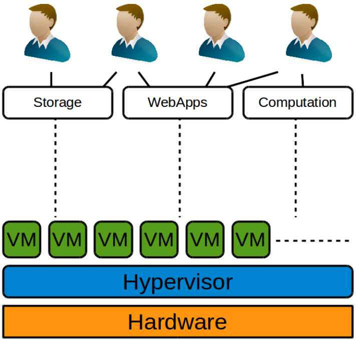
Anaxagoros Microkernel

- Clouds mutualize physical resources between users
  - Safety and security are crucial



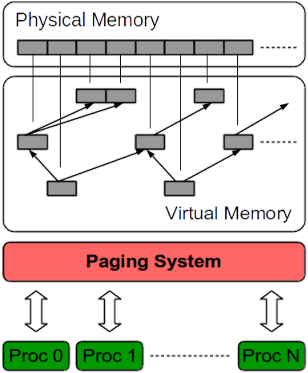
Anaxagoros Microkernel

- Clouds mutualize physical resources between users
  - Safety and security are crucial
- Anaxagoros
  - Secure microkernel hypervisor
  - Developped at CEA LIST by Matthieu Lemerre
  - Designed for resource isolation and protection
- Virtual memory system is a key module to ensure isolation



Virtual Memory Subsystem

- Organizes program address spaces
  - Creates a hierarchy of pages
  - Allows sharing when needed
- Controls accesses and modifications to the pages
  - Only owners can access their pages
  - Types of the pages limit possible actions
- Counts mappings, references, to each page



Memory invariant for sequential version

- Maintain the counters of mappings to pages:
  - The counter $mappings[e]$ must be equal to the real number of mappings to the page e
  - Let Occ^e be the number of mappings, i.e. occurrences of e in all pagetables
- We want ot prove:

$$\boxed{\forall e, validpage(e) \Rightarrow Occ^e = mappings[e] \leq MAX}$$

Memory invariant for concurrent version

Concurrency issues

- Pages might be modified by different processes simultaneously
- That creates a gap between the actual number of mappings and the counter

New invariant :

$$\forall e, validpage(e) \Rightarrow Occ^e \leq mappings[e] \leq MAX$$

and more precisely,

$$\forall e, validpage(e) \Rightarrow \exists k.\ k \geq 0 \wedge Occ^e + k = mappings[e] \leq MAX$$

Here k is the number of threads that have introduced a difference in the counter, difference of at most 1.

Simulation of the concurrency

- To model the execution context, we introduce for each thread :
  - global arrays representing the value of each local variable
  - a global array representing its position in the execution
- We simulate every atomic step with a function that performs this step for one thread
- We create an infinite loop that randomly chooses a thread and makes it perform a step of execution according to its current position

Verification results

- Partial verification of a critical module of Anaxagoros hypervisor
- For low-level functions, we conducted a “classic” verification
  - Specification with ACSL
  - Automatic proof with Frama-C/WP and SMT Solvers (CVC4, Z3)
- For the concurrent function used to change pagetables :
  - First specification and proof for sequential version
  - Weakening of the invariant for concurrency
  - Specification and proof of the simulated version
- Only a few properties could not be proved automatically
  - their proof is done in Coq by extracting them from WP

Lessons Learned, Limitations and Benefits

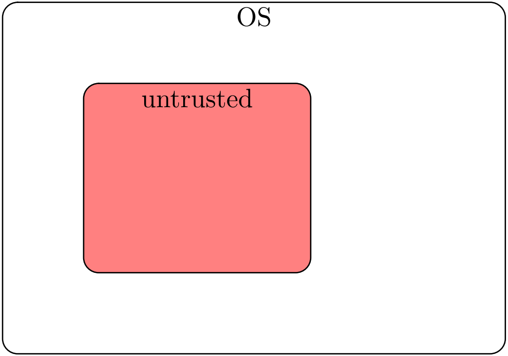
- Ability to treat concurrent programs
  - With a tool that originally does not handle parallelism
  - Proof done mostly automatically
  - Verification of properties in isolation
- Scalability
  - By-hand simulation is tedious and error prone
  - Could perfectly be automized
  - Need for specification mean for concurrent behaviors

Outline

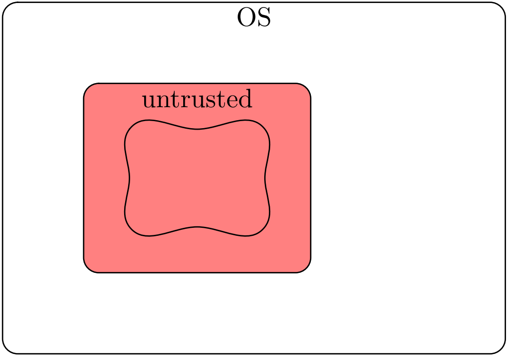
ZeroVM: History

- Developed by Google as a sandboxing technique for Chrome (2009)
- Native Client (NaCl) plugins use Chrome API
- ZeroVM: programs outside Chrome use ZeroVM syscalls (2011)

ZeroVM: Big picture

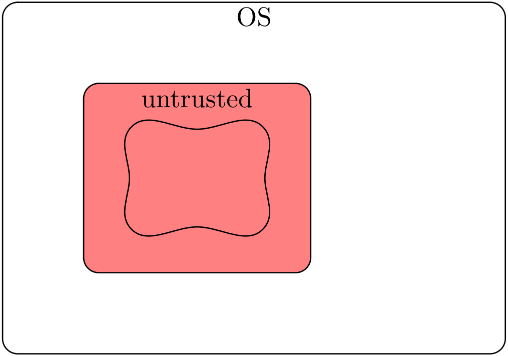


ZeroVM: Big picture



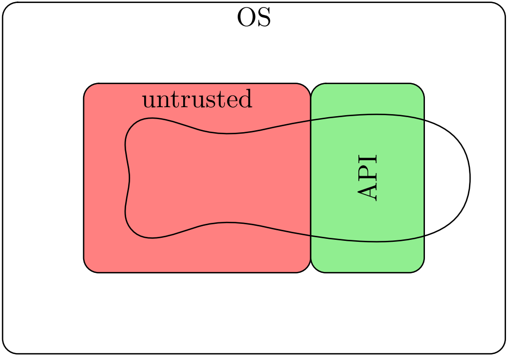
- Prevents privacy issues, privilege escalation, unauthorized device access...

ZeroVM: Big picture



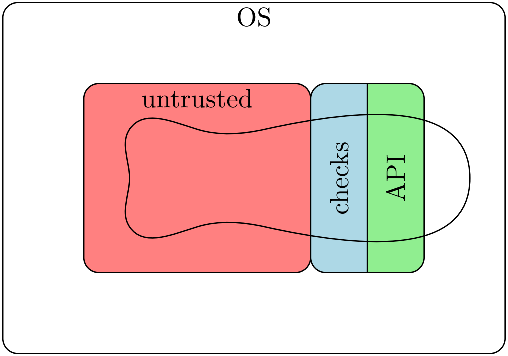
- Prevents privacy issues, privilege escalation, unauthorized device access...
- Performs binary code validation before execution

ZeroVM: Big picture



- Prevents privacy issues, privilege escalation, unauthorized device access...
- Performs binary code validation before execution

ZeroVM: Big picture



- Prevents privacy issues, privilege escalation, unauthorized device access...
- Performs binary code validation before execution
- Checks API calls (used for syscall invocations)

Verification of ZeroVM

Specificaton in ACSL and deductive verification with Frama-C/WP of API checks before syscall invocation:

```
/*@
  requires valid_nap(nap);
  ensures  valid_nap(nap);
  @*/
int32_t TrapHandler(struct NaClApp *nap,
                    uint32_t* args){
...
}
```

API handler for validation of Read operations

```
static int32_t ZVMReadHandle(
    struct NaClApp *nap,
    int ch, char *buffer,
    int32_t size, int64_t offset){
  ...
}
```

Checks performed by ZVMReadHandle:

- ch channel exists
- buffer is writable on size length
- [offset; offset+size] ⊂ [0; channel->size]
- limits are not reached

API handler for validation of memory accesses

```
/*@
requires valid_nap(nap);
requires nap->mem_start <= start;
assigns \nothing;
ensures \result == 0 ==> prot == PROT_READ ==>
        valid_read_segment(start,start+size);
ensures \result == 0 ==> prot == PROT_WRITE ==>
        valid_segment(start,start+size);
ensures \result == 0 || \result == -1;
@*/
static int CheckRAMAccess(struct NaClApp *nap,
  NaClSysPtr start, uint32_t size, int prot)
```

Issues detected by formal verification (1/3)

before correction:

```
int64_t size; uintptr_t start, nap->mem_map[i].end;

size -= (nap->mem_map[i].end - start);
if(size <= 0) return 0;
```

after correction:

Issues detected by formal verification (1/3)

before correction:

```
int64_t size; uintptr_t start, nap->mem_map[i].end;

size -= (nap->mem_map[i].end - start);
if(size <= 0) return 0;
```

after correction:

```
if(size <= (nap->mem_map[i].end - start)) return 0;
size -= nap->mem_map[i].end - start;
```

Issues detected by formal verification (2/3)

before correction:

```
int32_t size, int64_t offset;
int64_t channel->size;

/* prevent reading beyond the end of the channel */
size = MIN(channel->size - offset, size);

/* check arguments sanity */
if(size == 0)
   return 0; /* success. user has read 0 bytes */
if(size < 0) return -EFAULT;
if(offset < 0) return -EINVAL;
```

Issues detected by formal verification (2/3)

after correction:

```
/* check offset sanity */
if(offset < 0 || offset >= channel->size)
    return -EINVAL;

/* prevent reading beyond the end of the channel */
size = MIN(channel->size - offset, size);

/* check arguments sanity */
if(size == 0)
  return 0; /* success. user has read 0 bytes */
if(size < 0) return -EFAULT;
```

Issues detected by formal verification (3/3)

before correction:

```
if(offset >= channel->size + tail) return -EINVAL;
```

after correction:

Issues detected by formal verification (3/3)

before correction:

```
if(offset >= channel->size + tail) return -EINVAL;
```

after correction:

```
if(offset >= channel->size &&
   offset - channel->size >= tail) return -EINVAL;
```

Verification results

- Frama-C/WP automatically proves specified properties
  - 64 proof obligations for functional properties
  - 69 proof obligations to prevent runtime errors
- several issues and potential security flaws detected and reported to the development team
- a new version of ZeroVM fixed the issues

Conclusion

We performed deductive verification in Frama-C for

- a submodule of a Cloud hypervisor
- a sandbox for secure execution of user applications

Results:

- a concurrent version verified via simulation
- a few potential errors and security flaws detected and reported
- Frama-C provides a rich and extensible framework for formal verification of C code

Future work:

- apply Frama-C for formal verification of real-sized Cloud software