Efficient Formally Secure Compilers to a Tagged Architecture

Cătălin Hrițcu
INRIA Paris
The problem: devastating low-level attacks

• inherently insecure low-level languages (C, C++)
  – memory unsafe: any buffer overflow can be catastrophic allowing remote attackers to gain complete control

• unsafe interoperability with lower-level code
  – even code written in safer high-level languages (Java, C#, OCaml) has to interoperate with insecure low-level libraries (C, C++, ASM)
  – unsafe interoperability: all high-level safety guarantees lost

• Today’s languages & compilers plagued by low
  – fixing this purely in software would be way
software-defined, hardware-accelerated, tag-based monitoring

store

software monitor’s decision is hardware cached

(e.g. out of bounds write)
Micro-policies are cool!

- : unbounded per-word metadata, checked & propagated on each instruction
- : tags and monitor defined by software
- : hardware caching, <10% overhead
  - heap safety, control-flow integrity, taint tracking
- : complex policies for secure compilation
- and enough to verify security in Coq
- : FPGA implementation on top of RISC-V

[Oakland '13 & '15, POPL '14, ASPLOS '15]
Use micro-policies to build
or

- C with protected components and memory safety

- ASM, C, and F* \([F^* = \text{ML} + \text{verification}]\)
Formally verify: full abstraction

Benefit:
- Sound security reasoning in the source language
- Forget about the compiler chain (linker, loader, runtime system)
- Forget that libraries are written in a lower-level language

Holy grail of secure compilation, enforcing abstractions all the way down

(e.g. CompCert)
SECOMP: achieving full abstraction at scale

- miTLS* (ML + verification)
- CompSec
- SecF
- SecML
- C component
- protecting component boundaries
- legacy C component
- memory safe C component
- ASM component
- protecting higher-level abstractions

+ memory safety
+ components

(RISC-V + micro-policies)
Protecting component boundaries

- Add mutually distrustful components to C – interacting only via

- CompSec compiler chain (based on CompCert) – propagate interface information to produced binary

- Micro-policy simultaneously enforcing
  - component separation
  - type-safe procedure call and return discipline

- Interesting attacker model – extending full abs. to mutual distrust + unsafe source

Recent preliminary work, joint with Yannis Juglaret et al.
Compartmentalization micro-policy


at most one return capability per call stack level
Secure compartmentalizing compilation

∀ compromise scenarios.

∀ low-level attack from compromised $C_2$, $C_4$, $C_5$

∃ high-level attack from some fully defined $A_2$, $A_4$, $A_5$

, et al, CSF’16]
•

– types, value immutability, opaqueness of closures, parametricity (dynamic sealing), GC vs malloc/free, ...

– some can be turned into checked dynamically

– fully abstract compilation of F* to ML
  (because F* allows and tracks effects, as opposed to Coq)

– functional purity, termination, relational reasoning
Micro-policies:

- remaining fundamental challenges

  • Micro-policies for C and ML
    - needed for vertical compiler composition
    - will put micro-policies in the hands of programmers

  • Secure micro-policy composition
    - micro-policies are interferent reference monitors
      - one micro-policy's behavior can break another's guarantees
        • e.g. composing anything with IFC can leak
SECOMP in a nutshell

- We need more secure languages, compilers, hardware

- Key enabler: micro-policies (software-hardware protection)

- Grand challenge: the first efficient formally secure compilers for realistic programming languages (C, ML, F*)

- Answering challenging fundamental questions – attacker models, composition, micro-policies for C and ML

- + testing and proving formally that this is the case

- Most of this is at this point but ...
  - trying to build a community and looking for collaborators & students & PostDocs to try to make some of this real