Formally Secure Compilation of Compartmentalized C Programs

Cătălin Hrițcu, MPI-SP, Bochum

Joint work with
Carmine Abate, Cezar-Constantin Andrici, Arthur Azevedo de Amorim, Roberto Blanco, Ștefan Ciobâcă, Adrien Durier, Akram El-Korashy, Boris Eng, Ana Nora Evans, Guglielmo Fachini, Deepak Garg, Aïna Linn Georges, Théo Laurent, Guido Martínez, Marco Patrignani, Benjamin Pierce, Exequiel Rivas, Marco Stronati, Éric Tanter, Jérémy Thibault, Andrew Tolmach, Théo Winterhalter, ...

In part supported by ERC Starting Grant SECOMP
Secure Compilation of Secure Source Programs

• Suppose we have a secure source program ...
  – For instance formally verified in F* [POPL'16,'17,'18,'20, ICFP'17,'19, ...]
  – e.g. EverCrypt verified crypto library, shipping in Firefox, Linux Kernel, ...
  – e.g. simple verified web server, linking with unverified libraries [arXiv'23]

• What happens when we compile such a verified program and link it with adversarial low-level code?
  – low-level code that can be buggy, vulnerable, compromised, malicious
  – currently: all guarantees are lost, lower-level attacks become possible
  – secure compilation: protect the source abstractions all the way down
Insecure languages like C enable devastating vulnerabilities.

Mitigate vulnerabilities by compartmentalizing the program.

We don't know which compartments will be compromised.
  - protect vulnerable C compartments from each other.

We don't know when a compartment will be compromised.
  - every compartment should receive protection until compromised.
Secure Compilation is for us an instance of ...

- Goal
- Formally Verified Security
- Proof
- Enforcement
∀ security property π

∀ F*code

∀ low-level code

∀ ∀ F*code

∀ verified program

∀ compiled program

∀ low-level code

∀ protected

∀ no extra power

F*code satisfies π

∀ protected

∀ no extra power

↓

∀ satisfies π

Where π can e.g. be "the web server's private key is not leaked"

We explored many classes of properties one can preserve this way:
Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

More interesting definition for vulnerable C compartments [CSF'16, CCS'18]
2. Security Enforcement

Large subset of C with compartments

CompCert verified C compiler extended with compartments

RISC-V ASM with compartments

magically secure semantics

Software-Fault Isolation

- vanilla ASM
  
  Done for simplified languages, yet to be ported to RISC-V

- Micro-Policies: ASM with programmable tags
  
  [POPL'14, S&P'15, ASPLOS'15, POST'18, CCS'18, CSF'23]

Hardware-accelerated enforcement

- CHERI RISC-V capability machine
  
  (inspiration for ARM Morello)

  [PriSC'23, ongoing]
3. Security Proofs

- Proving mathematically that our compilation chains achieve secure compilation
  - such proofs generally **very difficult and tedious**
    - wrong conjectures for full abstraction have survived for decades
    - 250 pages of proof on paper for toy compiler
  - we propose **more scalable proof techniques** [CCS'18, CSF'22]
  - **machine-checked proofs** in the Coq and F* proof assistants
  - **systematic testing** to find wrong conjectures early
    [POPL'17, ICFP'13, ITP'15, JFP'16]
Testing and Proving Secure Compilation in Coq

Machine-checked proofs

Large subset of C with compartments

CompCert with compartments

RISC-V ASM with compartments

Scalable proof technique for secure compilation
• applied to simpler languages [CCS'18, CSF'22]
• now extending to CompCert with compartments
• reuses compiler correctness proof (extended!)
• aiming to finish secure compilation proof by fall
  – this will be a milestone in terms of realism!
  – all prior work on full-abstraction-like proofs is toy!

Software-Fault Isolation

vanilla ASM

Micro-Policies: ASM with programmable tags

Done for simpler languages, yet to be ported to RISC-V

Future verification challenge

cheri RISC-V capability machine

Systematic testing
Future Plans on Formally Secure Compilation

Stronger Security Goals

Preserve data confidentiality against micro-architectural side-channel attacks, for arbitrary compartmentalized programs in F*, C, or Wasm (not only constant time crypto code)

Realistic Enforcement

ARM Morello capability machine

Better Proof Techniques

Capability passing

Verify capability backend