Current group:

Talk
- Cătălin Hrițcu (Faculty)

Posters
- Cezar Andrici (PhD)
- Jérémy Thibault (PhD)
- Rob Blanco (PostDoc)

Discuss
- Maxi Wuttke (PhD)
- Dongjae Lee (Intern)

Alumni:

- Carmine Abate (PhD)
- Théo Winterhalter (PostDoc)
- Adrien Durier (PostDoc)
- Aïna Linn Georges (Intern)
+ many more before MPI-SP
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 (Cezar's poster)

• 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.
∀ security property π

∀ F*code

verified program

∀ low-level code

compiled program

compiler

∀

satisfies π

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]
Large subset of C with compartments

CompCert verified C compiler extended with compartments

RISC-V ASM with compartments

Software-Fault Isolation

vanilla ASM

Micro-Policies: ASM with programmable tags

[POPL'14, S&P'15, ASPLOS'15, POST'18, CCS'18, CSF'23]

CHERI RISC-V capability machine

(inspiration for ARM Morello)

magically secure semantics

Hardware-accelerated enforcement

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

[ 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 survived for decades
    • 250 pages of proof on paper for toy compiler
  – we propose more scalable proof techniques
  – 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 July
  — milestone in terms of realism! all prior work toy!
- for details see poster by Jérémy and Rob

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

Capability passing

Better Proof Techniques

Verify capability backend