Per-Thread Compositional Compilation for Confidentiality-Preserving Concurrent Programs

(Extended Abstract)

Robert Sison
Data61, CSIRO and UNSW
Robert.Sison@data61.csiro.au

Recent work has demonstrated that *per-thread compositional* verification of *value-dependent noninterference* is feasible for concurrent programs: Murray et al. [9] presented a type system that, when applied individually to each of the threads of a concurrent program, can be used to establish that the entire program does not at any time leak *high-classified* data to any presently *low-classified, readable*[1] memory.

However, the gap remains that the verification of this (necessarily timing-sensitive) property for the source code of a program may not guarantee that the property still holds for that program once it has been compiled and is running on actual hardware. Anticipating this, Murray et al. [9] also presented and proved sound a per-thread compositional notion of *refinement* that guarantees that an implementation of some verified system enforces the same noninterference property at the level of the target language's semantics. My present work is now seeking to apply and extend this result by producing the first per-thread compiler that provably guarantees to preserve such a *concurrent noninterference* property.

To this end, I have adapted and implemented in the Isabelle/HOL theorem prover [10] an existing compilation scheme (developed by Tedesco et al. [14]) from a generic imperative `While` language to a generic RISC-style assembly language, and proved that my compiler preserves the compositional value-dependent noninterference property from [9]. This means that you can use it to compile each of the threads in such a program separately and obtain a proof that their compiled assembly implementations compose into a concurrent system that satisfies value-dependent noninterference at the level of the RISC-style target language (which for short we will subsequently refer to as `RISCLang`).

As a proof of concept, I have exercised this compiler on a concurrent `While` model of the Cross Domain Desktop Compositor (CDDC) [2] verified with Murray et al. [9]'s type system, and instantiated all of the related proofs of security for the compiled `RISCLang` artifact of the system. This compiler and all theories about it, including this model and proof of concept compilation of the CDDC, build on the prior work's Isabelle/HOL formalization [8] (itself a direct extension of a formalization of Mantel et al. [6]). The compiler is specified as an executable function in Isabelle/HOL and runs using Isabelle's code generation framework.

Immediate plans for future work include integrating the compiler with support that we have recently added to the type system for richer cross-component invariants (because the existing rely-guarantee support is only for assumptions on access to variables), and applying such an adaptation of the compiler to an already-verified (unpublished) richer model of the CDDC whose verification relies on the establishment of such an invariant.

In the medium-to-long term, I intend to expand on this work in a way that focuses on navigating the tension between performance and security inherent in the question of how to achieve compilation that provably preserves a timing-sensitive security property. Generally speaking, the long-term objective of my work is to inform requirements presented to compiler writers and hardware manufacturers that would help strike a balance between aggressive performance optimizations and an ability to prove the absence of information leaks.

With respect to approaching the functionality expected of a realistic compiler, I intend firstly to tackle register allocation, because the current register allocation scheme in my `While`-to-`RISCLang` proof of concept is naive and will simply fail (instead of spilling to memory) if it runs out of registers. The bigger-picture view of such a project would be to discover and characterize requirements that would make it possible to verify that more efficient register allocation algorithms are making safe choices of which variables spill to memory.

As for the question of grounding this work by compiling to a real target on which code can run, I intend to verify a compilation step from the generic `RISCLang` used here, to a target platform (e.g. Atmel AVR, 8051) with timing properties that are well-defined enough for the timing-sensitivity of our noninterference property to be meaningful. Furthermore, the question remains

[1] I qualify this with "presently" because the classifications are value-dependent and may change dynamically with the state of the system, and "readable" because we leverage rely-guarantee assumptions (directly expanding on the work of Mantel et al. [6]) in order to achieve this per-thread proof compositionality, and those assumptions can change dynamically too.

of how to verify realistic assembly-level software implementations of `RISCLang`'s locking primitives. (Another idea is to verify compilation to a language whose type system guarantees the locking discipline.) To this end, the semantics for Atmel AVR developed by Dewald et al. [5] suggests itself as a prime candidate for adaptation to our framework in the interest of verifying a compilation step that targets AVR assembly. Obtaining a runnable target language then gives us an opportunity to exercise the statistical leakage measurement techniques of Chatzikokolakis et al. [3] (much in the way of Mantel and Starostin [7]) in order to look for any further conceptual gaps between our semantics and the timing properties of live systems. Work is also underway on a translation from `RISCLang` to RISC-V assembly, which would give us another runnable platform that provides a point of comparison for statistical leakage measurement results.

I also plan to explore in more depth the relationship between our work and: the *observation functions* of Costanzo et al. [4], work on mitigating against adversarial caller contexts (as is focused on by full abstraction and the *trace-preserving compilation* of Patrignani and Garg [11]), and work on mitigating against adversarial co-habiting threads (the focus of constant-time cryptography research e.g. Barthe et al. [1]). Although our value-dependent noninterference property is timing-sensitive, our attacker model may differ from some of this other work in that we consider only the threads of a concurrent system that are verified to co-operate with the locking discipline. In some sense, it could appear that we are considering only attackers that conform to restrictions in the form of rely-guarantee assumptions necessary in order for the proof to hold, but another way of looking at this is that our work is seeking to make explicit the necessary assumptions for security to hold in a concurrent context, thus requiring there be a very good reason we should consider an attacker to conform to such assumptions. In certain environments (such as *separation kernels* [12] and *capability machines* [13]) this may be justified by the correct functionality of an underlying operating system or hardware platform whose job it is to enforce separation between the domains, giving us another potential avenue for seeking connections and collaboration with existing work on finding appropriate secure compilation targets.

References

[1] Gilles Barthe, Gustavo Betarte, Juan Campo, Carlos Luna, and David Pichardie. 2014. System-level Non-interference for Constant-time Cryptography. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. ACM, New York, NY, USA, 1267–1279. https://doi.org/10.1145/2660267.2660283

[2] Mark Beaumont, Jim McCarthy, and Toby Murray. 2016. The Cross Domain Desktop Compositor: Using Hardware-based Video Compositing for a Multi-level Secure User Interface. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications (ACSAC '16)*. ACM, New York, NY, USA, 533–545. https://doi.org/10.1145/2991079.2991087

[3] Konstantinos Chatzikokolakis, Tom Chothia, and Apratim Guha. 2010. Statistical Measurement of Information Leakage. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*. Springer-Verlag, Berlin, Heidelberg, 390–404. https://doi.org/10.1007/978-3-642-12002-2_33

[4] David Costanzo, Zhong Shao, and Ronghui Gu. 2016. End-to-end Verification of Information-flow Security for C and Assembly Programs. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '16)*. ACM, New York, NY, USA, 648–664. https://doi.org/10.1145/2908080.2908100

[5] Florian Dewald, Heiko Mantel, and Alexandra Weber. 2017. AVR Processors as a Platform for Language-Based Security. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*. 427–445. https://doi.org/10.1007/978-3-319-66402-6_25

[6] Heiko Mantel, David Sands, and Henning Sudbrock. 2011. Assumptions and Guarantees for Compositional Noninterference. In *Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium (CSF '11)*. IEEE Computer Society, Washington, DC, USA, 218–232. https://doi.org/10.1109/CSF.2011.22

[7] Heiko Mantel and Artem Starostin. 2015. Transforming Out Timing Leaks, More or Less. In *Proceedings, Part I, of the 20th European Symposium on Computer Security – ESORICS 2015 - Volume 9326*. Springer-Verlag New York, Inc., New York, NY, USA, 447–467. https://doi.org/10.1007/978-3-319-24174-6_23

[8] Toby Murray, Robert Sison, Edward Pierzchalski, and Christine Rizkallah. 2016. Compositional Security-Preserving Refinement for Concurrent Imperative Programs. *Archive of Formal Proofs* (June 2016). http://isa-afp.org/entries/Dependent_SIFUM_Refinement.shtml, Formal proof development.

[9] Toby Murray, Robert Sison, Edward Pierzchalski, and Christine Rizkallah. 2016. Compositional Verification and Refinement of Concurrent Value-Dependent Noninterference. In *IEEE Computer Security Foundations Symposium*. Lisbon, Portugal, 417–431.

[10] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. 2002. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg.

[11] Marco Patrignani and Deepak Garg. 2017. Secure Compilation and Hyperproperty Preservation. In *IEEE 30th Computer Security Foundations Symposium, CSF 2017, Santa Barbara, USA, August 21 - 25, 2017 (CSF'17)*.

[12] J. M. Rushby. 1981. Design and Verification of Secure Systems. In *Proceedings of the Eighth ACM Symposium on Operating Systems Principles (SOSP '81)*. ACM, New York, NY, USA, 12–21. https://doi.org/10.1145/800216.806586

[13] Lau Skorstengaard, Dominique Devriese, and Lars Birkedal. 2017. Reasoning about a Capability Machine with Local Capabilities: Provably Safe Stack and Return Pointer Management (without OS Support). http://cs.au.dk/~birke/papers/local-capabilities-conf.pdf Submitted for publication.

[14] F. Del Tedesco, D. Sands, and A. Russo. 2016. Fault-Resilient Non-interference. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. 401–416. https://doi.org/10.1109/CSF.2016.35