Secure Compilation Using Micro-Policies
(Extended Abstract)

Yannis Juglaret
Université Paris Diderot (Paris 7)
and Inria Paris-Rocquencourt

Cătălin Hriţcu
Inria Paris-Rocquencourt

***Abstract*—Micro-policies are instruction-level security monitoring mechanisms based on fine-grained metadata tags. In this talk, we will show how targeting a micro-policy machine can help in building an efficient secure compiler for a simple object-oriented language. We will present the challenges of devising a fully abstract compiler for this language, and discuss the additional challenges that arise when moving to more complex languages.**

Secure—or *fully abstract* [1]—compilers preserve all abstractions of the source language when translating a program. This means that a low-level attacker has no more power than a high-level one. In particular, such a compiler allows programmers and automated tools to reason about the security of a program using the abstractions available at the high level. Enforcing full abstraction all the way to the machine code level is, however, very hard. While some fully abstract compilers have been built [5], they often have very large overhead, large TCB, or only provide probabilistic guarantees against specific attacks [2]. We believe that this is the necessary price one has to pay for using current hardware, with its lack of security. Providing more protection at the hardware level looks like a mandatory first step towards efficient secure compilation. Recent work [5] has illustrated how *protected module architectures*, a coarse-grained hardware isolation mechanism, can help in devising a secure compilation scheme for a simple object-oriented language. This compilation scheme distinguishes one trusted component from its untrusted low-level context (attacker), and ensures that the context cannot break the guarantees of the protected component.

Many vulnerabilities in today's computer systems can be avoided if low-level code is constrained to obey sensible safety and security properties. Ideally, such properties might be enforced statically, but for obtaining pervasive guarantees all the way to the level of running machine code it is often more practical to detect and prevent violations dynamically using a *reference monitor*. Our work is part of a long-term collaborative project aimed at showing how a rich set of *micro-policies*—instruction-level security monitoring mechanisms based on fine-grained metadata tags—can be described as instances of a common dynamic monitoring framework, formalized and reasoned about with unified verification tools, and efficiently implemented using programmable metadata-propagation hardware [3], [4]. Micro-policies can be described as a combination of software-defined rules and monitor services. In a micro-policy machine, every word of data is augmented with a word-sized metadata tag, and a hardware monitor is in charge of propagating these tags every time a machine instruction is executed. The rules define how the monitor will perform tag propagation instruction-wise, while the services allow for direct interaction between the running code and the monitor.

Our current goal, which will be the topic of this talk, is to show how targeting a micro-policy machine can help in building efficient secure compilers. For this purpose, we will present a simple object-oriented calculus with classes, public methods, private fields, and static object declarations. Compared to [5], we consider a finer-grained setting with *an arbitrary number of mutually distrustful components*. In this setting, we do not assume any compile-time knowledge regarding *which* components may be corrupted at runtime, and a secure compiler thus has to provide protection for each of the high-level components separately to make sure that the non-corrupted components' abstractions will be preserved.

We will show the obstacles to full abstraction, and how we can design a naive compartmentalization micro-policy that ensures it. Then, we will consider a more efficient micro-policy that allows low-overhead secure method calls between different compartments; this micro-policy uses linear return capabilities to ensure a proper call and return discipline. In both cases, we will also outline the challenges of obtaining a formal proof of full abstraction.

We will conclude by presenting future work. Our final goal is to provide full abstraction for a functional programming language. We plan to reach this goal gradually, by first extending the source language with dynamic allocation, and then moving to the untyped λ-calculus, in which dynamic allocation is implicit, as a source language. To this end, we intend to use a set of micro-policies that can be composed to enforce the higher-level abstractions with low overhead: control-flow integrity, stack protection, and memory safety. We will outline some of the challenges that arise, and how we plan to overcome them.

References

[1] M. Abadi. Protection in programming-language translations, 1998.

[2] M. Abadi and G. D. Plotkin. On protection by layout randomization. *ACM Trans. Inf. Syst. Secur.*, 15(2):8, 2012.

[3] A. Azevedo de Amorim, M. Dénès, N. Giannarakis, C. Hriţcu, B. C. Pierce, A. Spector-Zabusky, and A. Tolmach. Micro-policies: Formally verified, tag-based security monitors. In *36th IEEE Symposium on Security and Privacy (Oakland S&P)*. 2015.

[4] U. Dhawan, C. Hriţcu, R. Rubin, N. Vasilakis, S. Chiricescu, J. M. Smith, T. F. Knight, Jr., B. C. Pierce, and A. DeHon. Architectural support for software-defined metadata processing. *ASPLOS*, 2015.

[5] M. Patrignani, P. Agten, R. Strackx, B. Jacobs, D. Clarke, and F. Piessens. Secure compilation to protected module architectures. *ACM Transactions on Programming Languages and Systems*, 2015.