PER-based certification of secure information flow

Andrzej Filinski
University of Copenhagen
andrzej@di.ku.dk

Ken Friis Larsen
University of Copenhagen
kflarsen@di.ku.dk

Thomas Jensen
Inria and University of Copenhagen
thomas.jensen@inria.fr

Operating systems kernels use extensions to increase functionality. An extension is code provided by an external source and executes with increased system privileges, giving it access to confidential data. This work is concerned with the security properties of kernel extensions and in particular about how to ensure that they do not leak confidential information available to them. That is, how information flow control (IFC) can be enforced. A number of techniques have been proposed for executing kernel extensions safely, including runtime monitoring [1], Software Fault Isolation [2] and Proof Carrying Code (PCC) [3]. In PCC, the foreign code comes with a proof of its safety that can be checked before execution. This has the advantage that the code can avoid the runtime overhead inherent in monitoring or SFI. It is by now well understood how to produce proofs of memory safety. PCC for IFC, on the other hand, is much less developed. PCC for IFC has previously been based on type systems [4]–[6].

Necula's original work addressed extensions written in the Berkeley Packet Filter and focused on their memory safety properties. Recent work [7] proposes a verifier for Linux extensions in eBPF that provides some IFC guarantees, but these are mainly concerned with protecting the kernel itself, and not about where data from user space will flow.

Formal reasoning about information flow Secure IFC is traditionally presented in terms of statically dividing program variables (or, more generally, components of data structures, such as record fields) into *high and low security*, with the stipulation that, upon termination, the final values of low-security variables must not depend on the initial values of high-security ones. (Additionally, one often requires that termination itself must not depend on high-security variables.) Logics for information flow have been based on dynamic logic [8], relational logic [9], Hoare logic [10], and PER models [11].

Limitations of partitioning-based logics However, for many purposes, such a cartesian partitioning is far too coarse to express natural security policies that one would want to impose on a data inspector. For example, we may want to allow a query about whether a sensitive data field is zero or non-zero, but in the latter case, without revealing its actual value. Or we may allow a check of whether a purported checksum (or even a cryptographic hash) of a message is correct, but again without leaking the actual message content. We will present a PER-based technique for using a Hoare logic to provide proofs of IFC properties, extending Necula's original PCC approach. By using Hoare logic, we can prove information flow properties that are out of reach of type-based static analyses.

PER-based models While limited queries on sensitive data can be accommodated in an ad hoc way by extending the inspector's API with a collection of privileged observer functions (zero-test, verify-checksum, etc.) that return low-security results on high-security arguments, a more versatile framework would allow the data owner/custodian to specify allowable observations by means of a logical formula, and the inspector must then *prove* that the program—no matter how implemented—complies with the policy, just like for traditional safety properties. Formally, the security policy can be expressed as part of a *partial equivalence relation (PER)* on states, with equivalence of elements corresponding to indistinguishabilty by an unprivileged observer (whereas safety-violating elements are not even related to themselves). A program respecting the policy is then a *PER morphism*, which maps related inputs to related outputs.

For example, the PER specifying that a variable x is low-security but y is high-security can be expressed as the logical assertion $x_1 = x_2$, i.e., two input tuples are related if they agree on the value of x, but not necessarily on y. A policy-compliant inspector may then compute as its observable output any function f of x and y that does not actually depend on y, because $x_1 = x_2$ then ensures $f(x_1, y_1) = f(x_2, y_2)$. Dependence is here seen as a *semantic*, not *syntactic* property, so the function $f(x, y) = x \times (y - y + 1)$ would be acceptable.

If we want to allow the output to also depend on whether y is zero (but not on its actual value when it's not), we can add the conjunct $(y_1 = 0) \Leftrightarrow (y_2 = 0)$ to the PER specification. This will also make, e.g., $f(x, y) =$ if $y + 1 = 1$ then 3 else $5 + x$ a PER morphism.

Certifying compliance For terminating programs, proving compliance with a safety-and-security policy effectively amounts to proving functional correctness in a *relational* Hoare Logic [12], establishing a property involving two runs of the program, not just one. The key observation that we want to present here is that, given reasonable restrictions on how the PERs can be expressed, and by a suitable use of *ghost variables* in the proof, we can reduce relational correctness to (1) ordinary (unary) predicate-based correctness of the program, and (2) a little additional reasoning about integer arithmetic. For instance, the security of the program above comes from provability of the Hoare triple

$$\{(y + 1 = 1 \wedge g = 3) \vee (y + 1 \neq 1 \wedge g = 5 + x)\}$$
if y + 1 = 1 **then** x := 3 **else** x := 5 + x
$$\{g = x\},$$

together with semantic validity of the formula

$$(x_1 = x_2 \wedge (y_1 = 0 \Leftrightarrow y_2 = 0)) \Rightarrow \\ \exists g.((y_1 + 1 = 1 \wedge g = 3) \vee (y_1 + 1 \neq 1 \wedge g = 5 + x_1)) \wedge \\ ((y_2 + 1 = 1 \wedge g = 3) \vee (y_2 + 1 \neq 1 \wedge g = 5 + x_2)) .$$

Thus, we can largely piggy-back on an existing PCC infrastructure for plain Hoare logic (including certified reasoning about arithmetic, as already required for the consequence rule) to certify compliance with a very general notion of secure IFC.

REFERENCES

[1] G. Le Guernic and T. Jensen, "Monitoring Information Flow," in *Workshop on Foundations of Computer Security - FCS'05*, ser. Proceedings of the 2005 Workshop on Foundations of Computer Security (FCS'05), A. Sabelfeld, Ed. Chicago/USA: DePaul University, Jul. 2005, pp. 19–30. [Online]. Available: https://hal.inria.fr/inria-00001218

[2] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient software-based Fault Isolation," in *SOSP*. ACM, 1993, pp. 203–216.

[3] G. C. Necula and P. Lee, "Safe kernel extensions without run-time checking." in *Proc. of 2nd Symp. on Operating System Design and Implementation (OSDI '96)*. USENIX, 1996, pp. 229–243.

[4] G. Barthe, D. Pichardie, and T. Rezk, "A certified lightweight non-interference Java bytecode verifier," *Mathematical Structures in Computer Science (MSCS)*, vol. 23, no. 5, pp. 1032–1081, 2013.

[5] T. Amtoft, J. Dodds, Z. Zhang, A. Appel, L. Beringer, J. Hatcliff, X. Ou, and A. Cousino, "A certificate infrastructure for machine-checked proofs of conditional information flow," in *Principles of Security and Trust (POST'12)*. Springer LNCS vol. 7215, 2012, pp. 369–389.

[6] M. Töws and H. Wehrheim, "Information flow certificates," in *Theoretical Aspects of Computing – ICTAC 2018*. Springer LNCS vol. 11187, 2018, p. 435–454.

[7] E. Gershuni, N. Amit, A. Gurfinkel, N. Narodytska, J. A. Navas, N. Rinetzky, L. Ryzhyk, and M. Sagiv, "Simple and precise static analysis of untrusted linux kernel extensions," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019. New York, NY, USA: ACM, 2019, pp. 1069–1084. [Online]. Available: http://doi.acm.org/10.1145/3314221.3314590

[8] A. Darvas, R. Hähnle, and D. Sands, "A theorem proving approach to analysis of secure information flow," in *Proc. 2nd International Conference on Security in Pervasive Computing*. Springer LNCS vol. 3450, 2005, pp. 193–209.

[9] C. Müller, M. Kovács, and H. Seidl, "An analysis of universal information flow based on self-composition," in *IEEE 28th Computer Security Foundations Symposium, CSF 2015*. IEEE Computer Society, 2015, pp. 380–393.

[10] L. Beringer and M. Hofmann, "Secure information flow and program logics," in *20th IEEE Computer Security Foundations Symposium, CSF 2007*. IEEE Computer Society, 2007, pp. 233–248.

[11] A. Sabelfeld and D. Sands, "A per model of secure information flow in sequential programs," *Higher-Order and Symbolic Computation*, vol. 14, no. 1, pp. 59–91, March 2001.

[12] N. Benton, "Simple relational correctness proofs forstatic analyses and program transformations," in *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Venice, Italy, Jan. 2004, pp. 14–25.