From Fine- to Coarse-Grained Dynamic Information Flow Control and Back

Other titles in Foundations and Trends® in Programming Languages

Probabilistic Trace and Testing Semantics: The Importance of Being Coherent
Marco Bernardo
ISBN: 978-1-63828-074-3

Introduction to Neural Network Verification
Aws Albarghouthi
ISBN: 978-1-68083-910-4

Refinement Types: A Tutorial
Ranjit Jhala and Niki Vazou
ISBN: 978-1-68083-884-8

Shape Analysis
Bor-Yuh Evan Chang, Cezara Drăgoi, Roman Manevich, Noam Rinetzky and Xavier Rival
ISBN: 978-1-68083-732-2

Progress of Concurrent Objects
Hongjin Liang and Xinyu Feng
ISBN: 978-1-68083-672-1

QED at Large: A Survey of Engineering of Formally Verified Software
Talia Ringer, Karl Palmskog, Ilya Sergey, Milos Gligoric and Zachary Tatlock
ISBN: 978-1-68083-594-6

From Fine- to Coarse-Grained Dynamic Information Flow Control and Back

A Tutorial on Dynamic Information Flow

Marco Vassena
Utrecht University
m.vassena@uu.nl

Alejandro Russo
Chalmers University of Technology
russo@chalmers.se

Deepak Garg
Max Planck Institute for Software Systems
dg@mpi-sws.org

Vineet Rajani
University of Kent
v.rajani@kent.ac.uk

Deian Stefan
University of California, San Diego
deian@cs.ucsd.edu

the essence of knowledge
Boston — Delft

Foundations and Trends® in Programming Languages

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

M. Vassena *et al.*. *From Fine- to Coarse-Grained Dynamic Information Flow Control and Back*. Foundations and Trends® in Programming Languages, vol. 8, no. 1, pp. 1–117, 2023.

ISBN: 978-1-63828-219-8

Foundations and Trends® in Programming Languages
Volume 8, Issue 1, 2023

Editorial Board

Editorial Scope

Topics

Foundations and Trends® in Programming Languages publishes survey and tutorial articles in the following topics:

- Abstract Interpretation
- Compilation and Interpretation Techniques
- Domain Specific Languages
- Formal Semantics, including Lambda Calculi, Process Calculi, and Process Algebra
- Language Paradigms
- Mechanical Proof Checking
- Memory Management
- Partial Evaluation
- Program Logic
- Programming Language Implementation
- Programming Language Security
- Programming Languages for Concurrency
- Programming Languages for Parallelism
- Program Synthesis
- Program Transformations and Optimizations
- Program Verification
- Runtime Techniques for Programming Languages
- Software Model Checking
- Static and Dynamic Program Analysis
- Type Theory and Type Systems

Information for Librarians

Foundations and Trends® in Programming Languages, 2023, Volume 8, 4 issues. ISSN paper version 2325-1107. ISSN online version 2325-1131. Also available as a combined paper and online subscription.

Contents

From Fine- to Coarse-Grained Dynamic Information Flow Control and Back

Marco Vassena[1], Alejandro Russo[2], Deepak Garg[3], Vineet Rajani[4] and Deian Stefan[5]

[1] *Utrecht University, The Netherlands; m.vassena@uu.nl*
[2] *Chalmers University of Technology, Sweden; russo@chalmers.se*
[3] *Max Planck Institute for Software Systems, Germany; dg@mpi-sws.org*
[4] *University of Kent, UK; v.rajani@kent.ac.uk*
[5] *University of California, San Diego, USA; deian@cs.ucsd.edu*

ABSTRACT

This tutorial provides a complete and homogeneous account of the latest advances in fine- and coarse-grained dynamic information-flow control (IFC) security. Since the 1970s, the programming language and the operating system communities proposed different IFC approaches. IFC operating systems track information flows in a coarse-grained fashion, at the granularity of a process. In contrast, traditional language-based approaches to IFC are fine-grained: they track information flows at the granularity of program variables. For decades, researchers believed coarse-grained IFC to be strictly less permissive than fine-grained IFC—coarse grained IFC systems seem inherently less precise because they track less information——and so granularity appeared to be a fundamental feature of IFC systems.

Marco Vassena, Alejandro Russo, Deepak Garg, Vineet Rajani and Deian Stefan (2023), "From Fine- to Coarse-Grained Dynamic Information Flow Control and Back", Foundations and Trends® in Programming Languages: Vol. 8, No. 1, pp 1–117.
DOI: 10.1561/2500000046.

We show that the granularity of the tracking system does *not* fundamentally restrict how precise or permissive dynamic IFC systems can be. To this end, we mechanize two mostly standard languages, one with a fine-grained dynamic IFC system and the other with a coarse-grained dynamic IFC system, and prove a semantics-preserving translation from each language to the other. In addition, we derive the standard security property of non-interference of each language from that of the other, via our verified translation.

These translations stand to have important implications on the usability of IFC approaches. The coarse- to fine-grained direction can be used to remove the label annotation burden that fine-grained systems impose on developers, while the fine- to coarse-grained translation shows that coarse-grained systems—which are easier to design and implement—can track information as precisely as fine-grained systems and provides an algorithm for automatically retrofitting legacy applications to run on existing coarse-grained systems.

1

Introduction

Dynamic *information-flow control* (IFC) is a principled approach to protecting the confidentiality and integrity of data in software systems. Conceptually, dynamic IFC systems are very simple—they associate *security* levels or *labels* with every bit of data in the system to subsequently track and restrict the flow of labeled data throughout the system, e.g., to enforce a security property such as *non-interference* (Goguen and Meseguer, 1982). In practice, dynamic IFC implementations are considerably more complex—the *granularity* of the tracking system alone has important implications for the usage of IFC technology. Indeed, until somewhat recently (Roy *et al.*, 2009; Stefan *et al.*, 2017), granularity was the main distinguishing factor between dynamic IFC operating systems and programming languages. Most IFC operating systems (e.g., Efstathopoulos *et al.*, 2005; Zeldovich *et al.*, 2006; Krohn *et al.*, 2007) are *coarse-grained*, i.e., they track and enforce information flow at the granularity of a process or thread. Conversely, most programming languages with dynamic IFC (e.g., Austin and Flanagan, 2009; Zdancewic, 2002; Hedin *et al.*, 2014; Hritcu *et al.*, 2013; Yang *et al.*, 2012) track the flow of information in a more *fine-grained* fashion, e.g., at the granularity of program variables and references.

Dynamic coarse-grained IFC systems in the style of LIO (Stefan *et al.*, 2017; Stefan *et al.*, 2011; Stefan *et al.*, 2012; Heule *et al.*, 2015; Buiras *et al.*, 2015; Vassena *et al.*, 2017) have several advantages over dynamic fine-grained IFC systems. Such coarse-grained systems are often easier to design and implement—they inherently track less information. For example, LIO protects against control-flow-based *implicit flows* by tracking information at a coarse-grained level—to branch on secrets, LIO programs must first taint the context where secrets are going to be observed. Finally, coarse-grained systems often require considerably fewer programmer annotations—unlike fine-grained ones. More specifically, developers often only need a single label-annotation to protect everything in the scope of a thread or process responsible to handle sensitive data.

Unfortunately, these advantages of coarse-grained systems give up on the many benefits of fine-grained ones. For instance, one main drawback of coarse-grained systems is that it requires developers to compartmentalize their application in order to avoid both false alarms and the *label creep* problem, i.e., wherein the program gets too "tainted" to do anything useful. To this end, coarse-grained systems often create special abstractions (e.g., event processes (Efstathopoulos *et al.*, 2005), gates (Zeldovich *et al.*, 2006), and security regions (Roy *et al.*, 2009)) that compensate for the conservative approximations of the coarse-grained tracking approach. Furthermore, fine-grained systems do not impose the burden of focusing on avoiding the label creep problem on developers. By tracking information at fine granularity, such systems are seemingly more flexible and do not suffer from false alarms and label creep issues (Austin and Flanagan, 2009) as coarse-grained systems do. Indeed, fine-grained systems such as JSFlow (Hedin *et al.*, 2014) can often be used to secure existing, legacy applications; they only require developers to properly annotate the application.

This tutorial removes the division between fine- and coarse-grained dynamic IFC systems and the belief that they are fundamentally different. In particular, we show that *dynamic* fine-grained and coarse-grained IFC are equally expressive. Our work is inspired by the recent work of Rajani *et al.* (2017) and Rajani and Garg (2018), who prove similar results for *static* fine-grained and coarse-grained IFC systems. Specifically, they establish a semantics- and type-preserving translation from a coarse-grained IFC type system to a fine-grained one and vice-versa. We complete the picture by showing a similar result for dynamic IFC systems that additionally allow *introspection on labels* at run-time. While label introspection is meaningless in a static IFC system, in a dynamic IFC system this feature is key to both writing practical applications and mitigating the label creep problem (Stefan *et al.*, 2017).

Using the Agda proof assistant (Norell, 2009; Bove *et al.*, 2009), we formalize a traditional fine-grained system (in the style of Austin and Flanagan, 2009) extended with label introspection primitives, as well as a coarse-grained system (in the style of Stefan *et al.*, 2017). We then define and formalize modular semantics-preserving translations between them. Our translations are macro-expressible in the sense of Felleisen (1991), i.e., they can be expressed as a pure source program rewriting.

We show that a translation from fine- to coarse-grained is possible when the coarse-grained system is equipped with a primitive that limits the scope of tainting (e.g., when reading sensitive data). In practice, this is not an imposing requirement since most coarse-grained systems rely on such primitives for compartmentalization. For example, Stefan *et al.* (2017) and Stefan *et al.* (2012), provide **toLabeled**$(\cdot)$ blocks and threads for precisely this purpose. Dually, we show that the translation from coarse- to fine-grained is possible when the fine-grained system has a primitive **taint**$(\cdot)$ that relaxes precision to keep the *program counter label* synchronized when translating a program to the coarse-grained language. While this primitive is largely necessary for us to establish the coarse- to fine-grained translation, extending existing fine-grained systems with it is both secure and trivial.

The implications of our results are multi-fold. The fine- to coarse-grained translation formally confirms an old OS-community hypothesis that it is possible to restructure a system into smaller compartments to address the label creep problem—indeed our translation is a (naive) algorithm for doing so. This translation also allows running legacy fine-grained IFC compatible applications atop coarse-grained systems like LIO. Dually, the coarse- to fine-grained translation allows developers building new applications in a fine-grained system to avoid the annotation burden of the fine-grained system by writing some of the code in the coarse-grained system and compiling it automatically to the fine-grained system with our translation. The technical contributions of this monograph are:

- A pair of semantics-preserving translations between traditional dynamic fine-grained and coarse-grained IFC systems equipped with label introspection and flow-insensitive references (Theorems 5 and 7).
- Two different proofs of *termination-insensitive* non-interference (TINI) for each calculus: one is derived directly in the usual way (Theorems 1 and 3), while the other is recovered via our verified translation (Theorems 6 and 8).
- Mechanized Agda proofs of our results (~4,000 LOC).

This monograph is based on our conference paper (Vassena *et al.*, 2019) and extended with:

- A tutorial-style introduction to fine- and coarse-grained dynamic IFC, which (i) illustrates their specific features and (apparent) differences through examples, and (ii) supplements our proof artifacts with general explanations of the proof techniques used.
- *Flow-sensitive* references, a key feature for boosting the permissiveness of dynamic IFC systems (Austin and Flanagan, 2009). We extend both fine- and coarse-grained language with flow-sensitive references (Sections 2.3 and 3.3), adapt their security proofs (Theorems 2 and 4), and the verified translations to each other.
- A discussion and analysis of our extended proof artifact (~6,900 LOC)[1]. Our analysis finds that the security proofs for fine-grained languages are between 43% and 74% longer than for coarse-grained languages. These empirical results suggests that it is indeed easier to reason about coarse-grained IFC languages than fine-grained languages.

[1]The extended artifact is available at https://hub.docker.com/r/marcovassena/granularity-ftpl and supersedes the artifact archived with the conference paper.

This tutorial is organized as follows. Our dynamic fine- and coarse-grained IFC calculi are introduced in Sections 2 and 3, and then extended with flow-sensitive references in Sections 2.3 and 3.3, respectively. We prove the soundness guarantees (i.e., termination-insensitive non-interference) of the original languages (Sections 2.2 and 3.2), and of the extended languages (Sections 2.3.3 and 3.3.3). In Section 4, we discuss our mechanized proof artifacts and compare the security proofs of the two calculi, before and after the extension. In Section 5, we present the fine- to coarse-grained translation and a proof of non-interference for the fine-grained calculus recovered from non-interference of the other calculus through our verified translation. Section 6 presents similar results in the other direction. Related work is described in Section 7 and Section 8 concludes the tutorial.

References

Abadi, M., A. Banerjee, N. Heintze, and J. Riecke. (1999). “A Core Calculus of Dependency”. In: *Proc. ACM Symp. on Principles of Programming Languages*. 147–160.

Abel, A., G. Allais, A. Hameer, B. Pientka, A. Momigliano, S. Schäfer, and K. Stark. (2019). “POPLMark reloaded: Mechanizing proofs by logical relations”. *Journal of Functional Programming*. 29: e19. DOI: 10.1017/S0956796819000170.

Ahmadpanah, M. M., D. Hedin, M. Balliu, L. E. Olsson, and A. Sabelfeld. (2021). “SandTrap: Securing JavaScript-driven Trigger-Action Platforms”. In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association. 2899–2916. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/ahmadpanah.

Algehed, M. and J.-P. Bernardy. (2019). “Simple Noninterference from Parametricity”. *Proc. ACM Program. Lang.* 3(ICFP). DOI: 10.1145/3341693.

Austin, T. H. and C. Flanagan. (2009). “Efficient Purely-Dynamic Information Flow Analysis”. In: *Proc. of the 9th ACM Workshop on Programming Languages and Analysis for Security (PLAS ’09).*

Austin, T. H. and C. Flanagan. (2010). “Permissive Dynamic Information Flow Analysis”. In: *Proc. of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security. PLAS ’10.*

Austin, T. H. and C. Flanagan. (2012). "Multiple Facets for Dynamic Information Flow". In: *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '12*. Philadelphia, PA, USA: Association for Computing Machinery. 165–178. DOI: 10.1145/2103656.2103677.

Austin, T. H., T. Schmitz, and C. Flanagan. (2017). "Multiple Facets for Dynamic Information Flow with Exceptions". *ACM Trans. Program. Lang. Syst.* 39(3). DOI: 10.1145/3024086.

Balliu, M., D. Schoepe, and A. Sabelfeld. (2017). "We Are Family: Relating Information-Flow Trackers". In: *ESORICS*.

Banerjee, A. and D. A. Naumann. (2005). "Stack-based access control and secure information flow". *Journal Functional Programming*. 15(2): 131–177.

Barthe, G., T. Rezk, and A. Basu. (2007). "Security Types Preserving Compilation". *Computer Languages, Systems & Structures*. 33(2): 35–59. DOI: 10.1016/j.cl.2005.05.002.

Bastys, I., M. Balliu, and A. Sabelfeld. (2018). "If This Then What? Controlling Flows in IoT Apps". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS '18*. Toronto, Canada: Association for Computing Machinery. 1102–1119. DOI: 10.1145/3243734.3243841.

Bauer, L., S. Cai, L. Jia, T. Passaro, M. Stroucken, and Y. Tian. (2015). "Run-time Monitoring and Formal Analysis of Information Flows in Chromium". In: *Proc. of the 22nd Annual Network & Distributed System Security Symposium*. Internet Society.

Bell, E. D. and J. L. La Padula. (1976). "Secure computer system: Unified exposition and Multics interpretation". Bedford, MA. URL: http://csrc.nist.gov/publications/history/bell76.pdf.

Bichhawat, A., V. Rajani, D. Garg, and C. Hammer. (2014a). "Generalizing Permissive-Upgrade in Dynamic Information Flow Analysis". In: *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security. PLAS'14*. Uppsala, Sweden: Association for Computing Machinery. 15–24. DOI: 10.1145/2637113.2637116.

Bichhawat, A., V. Rajani, D. Garg, and C. Hammer. (2014b). "Information Flow Control in WebKit's JavaScript Bytecode". In: *International Conference on Principles of Security and Trust (POST)*. 159–178.

Bichhawat, A., V. Rajani, D. Garg, and C. Hammer. (2021). "Permissive runtime information flow control in the presence of exceptions". *Journal of Computer Security*. 29: 361–401. DOI: 10.3233/JCS-211385.

Bielova, N. and T. Rezk. (2016a). "A Taxonomy of Information Flow Monitors". In: *Principles of Security and Trust*. Ed. by F. Piessens and L. Viganò. Berlin, Heidelberg: Springer Berlin Heidelberg. 46–67.

Bielova, N. and T. Rezk. (2016b). "Spot the Difference: Secure Multi-execution and Multiple Facets". In: *Computer Security – ESORICS 2016*. Cham: Springer International Publishing. 501–519.

Birkedal, L., B. Reus, J. Schwinghammer, K. Støvring, J. Thamsborg, and H. Yang. (2011). "Step-Indexed Kripke Models over Recursive Worlds". In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '11*. Austin, Texas, USA: Association for Computing Machinery. 119–132. DOI: 10.1145/1926385.1926401.

Bove, A., P. Dybjer, and U. Norell. (2009). "A Brief Overview of Agda – A Functional Language with Dependent Types". In: *Theorem Proving in Higher Order Logics*. Ed. by S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel. Berlin, Heidelberg: Springer Berlin Heidelberg. 73–78.

Bowman, W. J. and A. Ahmed. (2015). "Noninterference for Free". In: *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming. ICFP 2015*. Vancouver, BC, Canada: Association for Computing Machinery. 101–113. DOI: 10.1145/2784731.2784733.

Broberg, N., B. van Delft, and D. Sands. (2013). "Paragon for Practical Programming with Information-Flow Control". In: *Proc. of the 11th Asian Symposium on Programming Languages and Systems. APLAS '13*. 217–232.

Buiras, P., D. Stefan, and A. Russo. (2014). "On Dynamic Flow-Sensitive Floating-Label Systems". In: *Proc. of the 2014 IEEE 27th Computer Security Foundations Symposium. CSF '14*. Washington, DC, USA: IEEE Computer Society. 65–79. DOI: 10.1109/CSF.2014.13.

Buiras, P., D. Vytiniotis, and A. Russo. (2015). "HLIO: Mixing Static and Dynamic Typing for Information-Flow Control in Haskell". In: *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming. ICFP 2015*. Vancouver, BC, Canada: Association for Computing Machinery. 289–301. DOI: 10.1145/2784731.2784758.

Cheng, W., D. R. Ports, D. Schultz, V. Popic, A. Blankstein, J. Cowling, D. Curtis, L. Shrira, and B. Liskov. (2012). "Abstractions for Usable Information Flow Control in Aeolus". In: *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX Association. 139–151. URL: https://www.usenix.org/conference/atc12/technical-sessions/presentation/cheng.

Devriese, D. and F. Piessens. (2010). "Noninterference through Secure Multi-execution". In: *Proc. of the 2010 IEEE Symposium on Security and Privacy. SP '10*. IEEE Computer Society.

Efstathopoulos, P., M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek, and R. Morris. (2005). "Labels and Event Processes in the Asbestos Operating System". In: *Proc. of the 20th ACM symp. on Operating systems principles. SOSP '05*.

Felleisen, M. (1991). "On the Expressive Power of Programming Languages". *Sci. Comput. Program.* 17(1-3): 35–75. DOI: 10.1016/0167-6423(91)90036-W.

Fernandes, E., J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash. (2016). "FlowFence: Practical Data Protection for Emerging IoT Application Frameworks". In: *USENIX Security Symposium*. 531–548.

Giffin, D. B., A. Levy, D. Stefan, D. Terei, D. Mazières, J. C. Mitchell, and A. Russo. (2012). "Hails: Protecting Data Privacy in Untrusted Web Applications". In: *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI '12*.

Goguen, J. and J. Meseguer. (1982). "Security Policies and Security Models". In: *Proc. of IEEE Symposium on Security and Privacy*. IEEE Computer Society.

Gregersen, S. O., J. Bay, A. Timany, and L. Birkedal. (2021). "Mechanized Logical Relations for Termination-Insensitive Noninterference". *Proc. ACM Program. Lang.* 5(POPL). DOI: 10.1145/3434291.

Hedin, D., A. Birgisson, L. Bello, and A. Sabelfeld. (2014). "JSFlow: Tracking Information Flow in JavaScript and its APIs". In: *Proc. of the ACM Symposium on Applied Computing (SAC '14)*.

Hedin, D. and D. Sands. (2006). "Noninterference in the presence of non-opaque pointers". In: *Proc. of the 19th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press.

Hedin, D. and A. Sabelfeld. (2012). "Information-Flow Security for a Core of JavaScript". In: *Proc. IEEE Computer Sec. Foundations Symposium*. IEEE Computer Society.

Heintze, N. and J. G. Riecke. (1998). "The SLam calculus: programming with secrecy and integrity". In: *Proc. ACM Symp. on Principles of Programming Languages*. 365–377.

Heule, S., D. Stefan, E. Z. Yang, J. C. Mitchell, and A. Russo. (2015). "IFC Inside: Retrofitting Languages with Dynamic Information Flow Control". In: *Proc. of the Conference on Principles of Security and Trust (POST '15)*. Springer.

Hirsch, A. K. and E. Cecchetti. (2021). "Giving Semantics to Program-Counter Labels via Secure Effects". *Proc. ACM Program. Lang.* 5(POPL). DOI: 10.1145/3434316.

Hritcu, C., M. Greenberg, B. Karel, B. C. Pierce, and G. Morrisett. (2013). "All Your IFCException Are Belong to Us". In: *Proc. of the 2013 IEEE Symposium on Security and Privacy. SP '13*. Washington, DC, USA: IEEE Computer Society. 3–17. DOI: 10.1109/SP.2013.10.

Hunt, S. and D. Sands. (2006). "On flow-sensitive security types". In: *Conference record of the 33rd ACM SIGPLAN-SIGACT Symp. on Principles of programming languages. POPL '06*. Charleston, South Carolina, USA: ACM. 79–90.

Jaskelioff, M. and A. Russo. (2011). "Secure Multi-execution in Haskell". In: *Proc. Andrei Ershov International Conference on Perspectives of System Informatics. LNCS*. Springer-Verlag.

Jia, L., J. Aljuraidan, E. Fragkaki, L. Bauer, M. Stroucken, K. Fukushima, S. Kiyomoto, and Y. Miyake. (2013). "Run-Time Enforcement of Information-Flow Properties on Android". In: *Proc. of the 18th European Symposium on Research in Computer Security (ESORICS '13)*. Springer.

Jung, R., R. Krebbers, J.-H. Jourdan, A. Bizjak, L. Birkedal, and D. Dreyer. (2018). "Iris from the ground up: A modular foundation for higher-order concurrent separation logic". *Journal of Functional Programming*. 28: e20. DOI: 10.1017/S0956796818000151.

Kozyri, E., F. B. Schneider, A. Bedford, J. Desharnais, and N. Tawbi. (2019). "Beyond Labels: Permissiveness for Dynamic Information Flow Enforcement". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. 351–35115. DOI: 10.1109/CSF.2019.00031.

Krohn, M., A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. (2007). *Information Flow Control for Standard OS Abstractions*. Stevenson, Washington, USA. DOI: 10.1145/1294261.1294293.

Myers, A. C., L. Zheng, S. Zdancewic, S. Chong, and N. Nystrom. (2006). *Jif 3.0: Java information flow*. URL: http://www.cs.cornell.edu/jif.

Nadkarni, A., B. Andow, W. Enck, and S. Jha. (2016). "Practical DIFC Enforcement on Android." In: *USENIX Security Symposium*. 1119–1136.

Nikiforakis, N., L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. (2012). "You Are What You Include: Large-Scale Evaluation of Remote Javascript Inclusions". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security. CCS '12*. Raleigh, North Carolina, USA: Association for Computing Machinery. 736–747. DOI: 10.1145/2382196.2382274.

Norell, U. (2009). "Dependently Typed Programming in Agda". In: *Advanced Functional Programming: 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*. Ed. by P. Koopman, R. Plasmeijer, and D. Swierstra. Berlin, Heidelberg: Springer Berlin Heidelberg. 230–266. DOI: 10.1007/978-3-642-04652-0_5.

Pedersen, M. V. and S. Chong. (2019). "Programming with Flow-Limited Authorization: Coarser is Better". In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. 63–78. DOI: 10.1109/EuroSP.2019.00015.

Pottier, F. and V. Simonet. (2003). "Information Flow Inference for ML". *ACM Trans. Program. Lang. Syst.* 25(1): 117–158. DOI: 10.1145/596980.596983.

Rajani, V., I. Bastys, W. Rafnsson, and D. Garg. (2017). "Type Systems for Information Flow Control: The Question of Granularity". *ACM SIGLOG News*. 4(1): 6–21. DOI: 10.1145/3051528.3051531.

Rajani, V., A. Bichhawat, D. Garg, and C. Hammer. (2015). "Information Flow Control for Event Handling and the DOM in Web Browsers". In: *2015 IEEE 28th Computer Security Foundations Symposium*. 366–379. DOI: 10.1109/CSF.2015.32.

Rajani, V. and D. Garg. (2018). "Types for Information Flow Control: Labeling Granularity and Semantic Models". In: *Proc. of the IEEE Computer Security Foundations Symp. CSF '18*. IEEE Computer Society.

Roy, I., D. E. Porter, M. D. Bond, K. S. McKinley, and E. Witchel. (2009). "Laminar: Practical Fine-grained Decentralized Information Flow Control". In: *Proc. of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI '09*. Dublin, Ireland: ACM. 63–74. DOI: 10.1145/1542476.1542484.

Russo, A. (2015). "Functional Pearl: Two Can Keep a Secret, if One of Them Uses Haskell". In: *Proc. of the 20th ACM SIGPLAN International Conference on Functional Programming. ICFP 2015*. ACM.

Russo, A., K. Claessen, and J. Hughes. (2009). "A library for light-weight Information-Flow Security in Haskell". *ACM SIGPLAN Notices (HASKELL '08)*. 44(Jan.): 13. DOI: 10.1145/1543134.1411289.

Russo, A. and A. Sabelfeld. (2010). "Dynamic vs. Static Flow-Sensitive Security Analysis". In: *Proc. of the 2010 23rd IEEE Computer Security Foundations Symp. CSF '10*. IEEE Computer Society. 186–199.

Sabelfeld, A. and A. Russo. (2009). "From dynamic to static and back: Riding the roller coaster of information-flow control research". In: *Proc. Andrei Ershov International Conference on Perspectives of System Informatics (PSI '09). LNCS*. Springer-Verlag.

Sabelfeld, A. and A. C. Myers. (2006). "Language-based Information-flow Security". *IEEE J.Sel. A. Commun.* 21(1): 5–19. DOI: 10.1109/JSAC.2002.806121.

Sabelfeld, A. and D. Sands. (2001). "A Per Model of Secure Information Flow in Sequential Programs". *Higher Order Symbol. Comput.* 14(1): 59–91. DOI: 10.1023/A:1011553200337.

Schmitz, T., M. Algehed, C. Flanagan, and A. Russo. (2018). "Faceted Secure Multi Execution". In: *Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS '18*. Toronto, Canada: ACM. 1617–1634. DOI: 10.1145/3243734.3243806.

Stefan, D., A. Russo, P. Buiras, A. Levy, J. C. Mitchell, and D. Mazières. (2012). "Addressing Covert Termination and Timing Channels in Concurrent Information Flow Systems". In: *International Conference on Functional Programming (ICFP)*. ACM SIGPLAN.

Stefan, D., A. Russo, D. Mazières, and J. C. Mitchell. (2017). "Flexible Dynamic Information Flow Control in the Presence of Exceptions". *Journal of Functional Programming*. 27.

Stefan, D., A. Russo, J. C. Mitchell, and D. Mazières. (2011). "Flexible Dynamic Information Flow Control in Haskell". In: *Proc. of the 4th ACM Symposium on Haskell. Haskell '11*. Tokyo, Japan: ACM. 95–106. DOI: 10.1145/2034675.2034688.

Stefan, D., E. Z. Yang, P. Marchenko, A. Russo, D. Herman, B. Karp, and D. Mazières. (2014). "Protecting Users by Confining JavaScript with COWL". In: *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation. OSDI'14*. Broomfield, CO: USENIX Association. 131–146. URL: http://dl.acm.org/citation.cfm?id=2685048.2685060.

Surbatovich, M., J. Aljuraidan, L. Bauer, A. Das, and L. Jia. (2017). "Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes". In: *Proceedings of the 26th International Conference on World Wide Web. WWW '17*. Perth, Australia: International World Wide Web Conferences Steering Committee. 1501–1510. DOI: 10.1145/3038912.3052709.

Tsai, T.-C., A. Russo, and J. Hughes. (2007). "A Library for Secure Multi-threaded Information Flow in Haskell". In: *Proc. of the 20th IEEE Computer Security Foundations Symposium (CSF'07)*. 187–202. DOI: 10.1109/CSF.2007.6.

Vassena, M. and A. Russo. (2016). "On Formalizing Information-Flow Control Libraries". In: *Proc. of the 2016 ACM Workshop on Programming Languages and Analysis for Security. PLAS '16*. Vienna, Austria: ACM. 15–28. DOI: 10.1145/2993600.2993608.

Vassena, M., A. Russo, P. Buiras, and L. Waye. (2017). "MAC A Verified Static Information-Flow Control Library". *Journal of Logical and Algebraic Methods in Programming*. DOI: https://doi.org/10.1016/j.jlamp.2017.12.003.

Vassena, M., A. Russo, D. Garg, V. Rajani, and D. Stefan. (2019). "From Fine- to Coarse-Grained Dynamic Information Flow Control and Back". *Proc. ACM Program. Lang.* 3(POPL). DOI: 10.1145/3290389.

Volpano, D., G. Smith, and C. Irvine. (1996). "A Sound Type System for Secure Flow Analysis". *J. Computer Security*. 4(3): 167–187.

Volpano, D. and G. Smith. (1997). "Eliminating Covert Flows with Minimum Typings". In: *Proc. of the 10th IEEE workshop on Computer Security Foundations. CSFW '97*. IEEE Computer Society.

Xiang, J. and S. Chong. (2021). "Co-Inflow: Coarse-grained Information Flow Control for Java-like Languages". In: *Proceedings of the 2021 IEEE Symposium on Security and Privacy*. Piscataway, NJ, USA: IEEE Press.

Yang, J., K. Yessenov, and A. Solar-Lezama. (2012). "A Language for Automatically Enforcing Privacy Policies". In: *Proc. of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '12*. Philadelphia, PA, USA: ACM. 85–96. DOI: 10.1145/2103656.2103669.

Yip, A., N. Narula, M. Krohn, and R. Morris. (2009). "Privacy-preserving Browser-side Scripting with BFlow". In: *Proc. of the 4th ACM European Conference on Computer Systems. EuroSys '09*. ACM.

Zdancewic, S. A. (2002). "Programming Languages for Information Security". *PhD thesis*. Ithaca, NY, USA.

Zeldovich, N., S. Boyd-Wickizer, E. Kohler, and D. Mazières. (2006). "Making Information Flow Explicit in HiStar". In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7. OSDI '06*. Seattle, WA: USENIX Association. 19–19. URL: http://dl.acm.org/citation.cfm?id=1267308.1267327.

Zeldovich, N., S. Boyd-Wickizer, and D. Mazières. (2008). "Securing Distributed Systems with Information Flow Control". In: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. NSDI'08*. San Francisco, California: USENIX Association. 293–308. URL: http://dl.acm.org/citation.cfm?id=1387589.1387610.