THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Flexible Information-Flow Control

DANIEL SCHOEPE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND GÖTEBORG UNIVERSITY

Göteborg, Sweden 2018

Flexible Information-Flow Control
DANIEL SCHOEPE
ISBN 978-91-7597-832-1

Technical Report 165D
ISSN 0346-718X
Ny serie nr. 4513
Department of Computer Science and Engineering
Research group: Information Security

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY and GÖTEBORG UNIVERSITY
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 1000

Printed at Chalmers
Göteborg, Sweden 2018

ABSTRACT

As more and more sensitive data is handled by software, its trustworthiness becomes an increasingly important concern. This thesis presents work on ensuring that information processed by computing systems is not disclosed to third parties without the user's permission; i.e. to prevent unwanted flows of information. While this problem is widely studied, proposed rigorous *information-flow control* approaches that enforce strong security properties like *noninterference* have yet to see widespread practical use. Conversely, lightweight techniques such as *taint tracking* are more prevalent in practice, but lack formal underpinnings, making it unclear what guarantees they provide.

This thesis aims to shrink the gap between heavyweight information-flow control approaches that have been proven sound and lightweight practical techniques without formal guarantees such as taint tracking. This thesis attempts to reconcile these areas by (a) providing formal foundations to taint tracking approaches, (b) extending information-flow control techniques to more realistic languages and settings, and (c) exploring security policies and mechanisms that fall in between information-flow control and taint tracking and investigating what trade-offs they incur.

List of Publications

This thesis is based on work contained in the following papers, each presented in a separate chapter. Chapters 1, 2, 4, 6, and 7 are published at peer-reviewed conferences while the content of the other papers is currently under submission.

Chapter 1. Explicit Secrecy: A Policy for Taint Tracking. *Daniel Schoepe, Musard Balliu, Benjamin C. Pierce, and Andrei Sabelfeld.* In *Proceedings of the 1st IEEE European Symposium on Security and Privacy (EuroS&P), pages 15–36. IEEE Computer Society, 2016.*

Chapter 2. Let's Face It: Faceted Values for Taint Tracking. *Daniel Schoepe, Musard Balliu, Frank Piessens, and Andrei Sabelfeld.* In *Proceedings of the 21st European Symposium on Research in Computer Security (ESORICS)*, pages 561–580. Springer, 2016.

Chapter 3. VERONICA: Verified Concurrent Information-Flow Security Unleashed. *Daniel Schoepe, Toby Murray, Andrei Sabelfeld. Under submission.*

Chapter 4. JSLINQ: Building Secure Applications across Tiers. *Musard Balliu, Benjamin Liebe, Daniel Schoepe, and Andrei Sabelfeld.* In the *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy (CODASPY)*, pages 307–318. ACM, 2016.

Chapter 5. Information-Flow Control for Database-Backed Applications. *Marco Guarnieri, Daniel Schoepe, Musard Balliu, David Basin, and Andrei Sabelfeld. Under submission.*

Chapter 6. Understanding and Enforcing Opacity. *Daniel Schoepe and Andrei Sabelfeld.* In *Proceedings of 28th IEEE Computer Security Foundations Symposium (CSF)*, pages 539-553. IEEE Computer Security, 2015.

Chapter 7. We Are Family: Relating Information-Flow Trackers. *Musard Balliu, Daniel Schoepe, and Andrei Sabelfeld.* In *Proceedings of the 22nd European Symposium on Research in Computer Security (ESORICS)*, pages 124-145. Springer, 2017.

Chapter 8. An Empirical Study of Information Flows in Real-World JavaScript. *Cristian-Alexandru Staicu, Daniel Schoepe, Musard Balliu, Michael Pradel, and Andrei Sabelfeld. Under submission.*

Acknowledgments

The past four and a half years have been a unique experience: often exciting, sometimes frustrating, but always memorable. Spending so much time on a single topic can be challenging and I owe a debt of gratitude to a many people for helping me to successfully finish my PhD.

Firstly, I want to thank my advisor Andrei for years of great supervision and advice and countless interesting discussions not just about research, but also about food, drinks, politics, and the rest of life. He also provided me with lots of exciting opportunities to work with other researchers and helped me organize research visits and internships. I also owe a debt of gratitude to Musard, Dave, and Wolfgang, for the many fruitful conversations and lots of helpful advice during my PhD.

Next, I want to thank the many great people that make (or made) coming to work each day feel not just like an obligation, but actually fun: Alejandro, Alexander, Aljoscha, Daniel, Elena, Elisabet, Evgeny, Herbert, Hiva, Inari, Iulia, Jeff, Katja, Marco, Mauricio, Max, Michal, Pablo, Per, Raul, Sandro, Simon, Solrún, Steven, Thomas, Víctor, and many others I doubtlessly forgot to mention here.

Being caught up in research and teaching all day makes it all too easy to forget that there's also a life outside of work and I'm grateful to have found so many friends over the years, so I want to thank Noémi, Eike, Nicole, Baldur, Елена, Brian, Irene, Fotini, Marianna, Alla, Ari, Edit, Maria, Rhys, among many others, for putting up with me over all these years. Lastly, I'm grateful to my parents for supporting me throughout this endeavor and many other challenges throughout my life.

At the end of this long road, I'm happy to have taken on this challenge, but also to finally bring it to a conclusion in the form of this thesis.

CONTENTS

CHAPTER
ZERO

INTRODUCTION

Developments in computer science have shaped many aspects of today's society. In fact, it is hard to find an area of modern life that is untouched by information technology. A significant portion of our lives, our institutions, and our economy is controlled by software. On a societal level, code determines how news is disseminated, how the stock market is managed, how our healthcare systems work, and sometimes we even rely on code to safeguard democratic elections. On an individual level, software impacts what we read, how we communicate with friends, express our thoughts, find our way in a city, manage our finances, take pictures, watch movies, find out what to spend our free time on, to name just a few examples.

In the process we make a large amount of sensitive data available to software, including our location, search queries, private messages, pictures and videos, hobbies, and political views. Yet — despite the role computing technology plays — there is no practical way, not even for an expert, to ensure that all the software in use is actually trustworthy. How can one be certain that an application does not send sensitive data to a third party? A malicious application could leak confidential files on the hard drive, passwords, private messages, and credit card numbers.

Disclosing private data handled by applications can have severe consequences: Companies may lose profits if internal data is leaked. If a credit card number is stolen by a scammer, the victim will lose money. A stolen social security number can give rise to identity theft. Revealing a person's political views, private conversations, religious beliefs, or sexual orientation might cause them to lose their job or impact their social life, or even cost them their lives in some countries. Identity and location can be particularly sensitive information; in the case of domestic abuse victims and dissidents living under an oppressive regime, their disclosure can prove quite literally life-threatening.

This problem is exacerbated as more and more adversaries attempt to compromise people's devices and the software that runs on them; attackers range from ordinary criminals motivated by simple greed to government agencies seeking to undermine the ideals of liberty and individual rights that underpin our society. Ever since Edward Snowden's revelations [4], we know that (even) Western governments steal their citizen's information on a large scale [10], ostensibly in an effort to combat terrorism, although evidence that mass surveillance is helpful in that regard has yet to surface [2,7]. While cryptography is essential in defending against attackers who control the network, such protection still relies on trustworthy software on the users' devices to perform the encryption and not leak the data in the process. Since one of the methods used for stealing data is to spread malicious or backdoored software to the users' devices [1, 8], this illustrates the need for a way of reliably establishing whether a piece of code is trustworthy.

The prevalent method to tackle this problem consists of testing and code review to check the security of software; however, just like testing for functionality, this can only show the presence of errors but never prove their absence. Moreover, new code is written at a pace that makes any manual verification process hard to implement in the present and impossible to scale in the long run.

In response, a more rigorous approach to software security is needed; ideally, one would ensure that software is secure *by construction*, with mathematical guarantees establishing that a program leaks no information, where these guarantees are checked mechanically.

However, even though rigorous approaches to software security exist in the academic community, they have yet to see widespread, practical use. This thesis explores approaches to make mathematically rigorous techniques to preventing information leaks more practical.

Section 0.1 provides a cursory look at the topic of *language-based security*, an approach to software security that makes use of ideas from the programming language community to enforce security properties. Section 0.2 gives a quick overview of the area of *information-flow control*, which is concerned with formally expressing what it means for programs to be secure, finding techniques that guarantee the security of a program, and proving that they guarantee the desired notion of security. Section 0.3 gives a brief summary of taint tracking and contrasts the approach with information-flow control. Section 0.4 outlines the challenges addressed in this thesis. Section 0.5 summarizes the contributions of the papers comprising this thesis.

0.1 Language-Based Security

A common approach to security in practice involves testing and code review to find security flaws; however, like testing for functionality, this approach cannot prove the absence of security flaws. Language-based security on the other hand is based on providing provable security guarantees by using the semantics of the underlying language: Instead of trying to find security problems in programs, techniques in this area focus on analyzing programs in order to prove that a program is secure, or on automatically preventing insecurities at runtime. Since untrusted programs are granted access to sensitive data, information-flow policies are fine-grained, making language-based techniques an excellent fit for information-flow control [266].

Broadly speaking, the approaches in language-based security fall into two categories [260]: *static* techniques analyze a program before it is run and aim to establish whether or not the program is secure for all inputs, while *dynamic* approaches prevent information leaks as the program is executing. Additionally, *hybrid* mechanisms combine static analysis with dynamic techniques.

An example for static approaches are *security type systems* [320]. In a normal type system, if a program is well-typed, then the program is proven to be free of certain classes of bugs, such as trying to add integers to strings; this is often summarized as "well-typed programs don't go wrong" [224]. A security type system on the other hand can be used to ensure that all well-typed programs are secure with respect to a security property. Chapter 3 presents a security type system to prevent illicit information flows in web applications.

Dynamic approaches typically modify the semantics of a program in order to prevent leaks of information. For example, if a program sends secret data to the internet, a simple dynamic information-flow enforcement might then terminate the program before the sensitive information is sent out over the network.

0.2 Information-Flow Control

The key feature that makes software security relevant is the amount and type of private information handled by computers today and the fact that they are connected to the internet. However, many programs legitimately need access to both private information and internet connectivity in order to perform their task. As a result, simply not giving a program access to private data is not a viable solution.

For example, a spreadsheet application may be used to manage one's finances, but also requires internet access to look up stock prices and other live data. If the program is malicious or buggy, it might instead leak one's financial situation to an attacker on the internet. Chat applications by their nature have access to the messages the user sends and receives, and need internet access to perform their function. However, if the application is malicious, the messages may be sent to attackers in addition to their intended recipients or the application may not encrypt messages correctly before they are transmitted.

The objective of information-flow control is to prevent information flows from sensitive *sources*, such as local files, to public *sinks*, such as servers on the internet. In this scenario we assume that the attacker controls public sources, such as data received from the internet, and can observe outputs to public sinks. Moreover, the attacker is assumed to know the code of the program being run, but cannot observe sensitive sinks, such as writing to local files, and cannot observe sensitive inputs to the program, such as user input or contents of local files.

To reason about security of software in a way that provides confidence about a program's behavior, however we need to express the absence of unwanted information flows needs formally, in unambiguous mathematical terms. The policy of not allowing information to flow from sensitive sources to public sinks is often formalized as *noninterference*. In this setting, a program is treated as a mathematical object that maps inputs, some of which may be sensitive, to outputs, some of which may be observable by attackers. A program is then considered secure if and only if for two sets of inputs that only differ on their confidential parts, the program always results in the same outputs to the attacker. Otherwise, an attacker can derive information about sensitive inputs by observing the public output of a program. Figure 1 illustrates this policy.

A simple example setting to describe this policy is to model a program as a function $prog : \mathcal{I}_P \times \mathcal{I}_S \to \mathcal{O}_P \times \mathcal{O}_H$ mapping a pair of public inputs from the set $\mathcal{I}_P$ and secret input from the set $\mathcal{I}_S$ to a pair of public outputs in set $\mathcal{O}_P$ and secret outputs $\mathcal{O}_S$[1]. Such a program then satisfies *noninterference* if for the same public input i_P and any two secret inputs i_S and i'_S, we have that $\pi_1(prog(i_P, i_s)) = \pi_1(prog(i_P, i'_S))$ where π_1 refers to the projection a tuple to its first component.

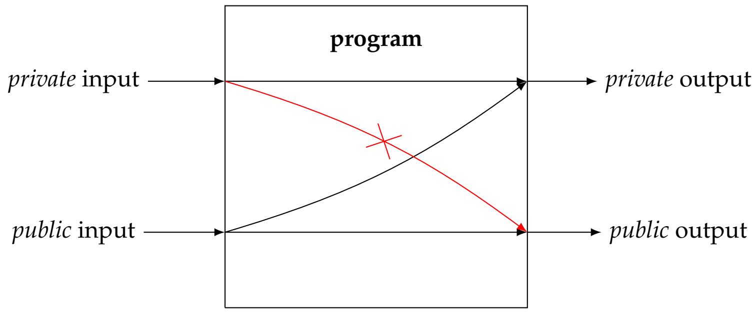

Figure 1: Noninterference

As most interesting program properties, noninterference is undecidable for non-trivial programs [255]. To make matters worse, noninterference is a property about two runs of a program [222] making verification or enforcement more challenging than that of safety or liveness properties commonly considered when verifying functional correctness of programs. As a result, approaches for enforcing noninterference typically compromise on either soundness, i.e. that insecure programs are classified as insecure, or precision, i.e. that secure programs are classified as secure. To still provide meaningful security guarantees, most approaches prioritize soundness over precision at the expense of practicality, since some secure programs can not be successfully verified.

Moreover, since noninterference requires that private information have no influence over any public outputs *at all*, it is sometimes too restrictive. For example, a program handling user logins will necessarily leak information about the stored password, by behaving differently depending on whether or not the user entered the correct password; i.e. it leaks whether the stored sensitive password is equal to the user's input.

0.3 Taint Tracking

Two important categories of information leaks that enforcement mechanisms need to handle are *explicit flows*, which leak data by directly outputting secret information to an attacker, and *implicit flows*, which leak information through the control-flow structure of the program. For example, the program

$$\textbf{outputTo}(attacker, secret)$$

[1]This glosses over some details such as how to model non-terminating programs, but illustrates the overall idea.

directly sends the value of the secret input stored in variable $secret$ to an attacker through an explicit flow, whereas the program

$$\textbf{if } secret = 0 \textbf{ then } \textbf{outputTo}(attacker, 1) \textbf{ else } \textbf{outputTo}(attacker, 2)$$

leaks the whether the secret input is zero through an implicit flow relying on how control-flow in the program is structured.

While both types of flows leak information, implicit flows are more challenging to track accurately throughout a program, since this involves comparing what happens in two separate branches of a conditional. To avoid imprecision introduced by tracking implicit flows, some approaches only track whether sensitive data directly enters a public sink (such as the **outputTo** function in the previous example). Such approaches are referred to as *taint tracking*. While they do not enforce noninterference, such approaches are often more feasible to use in practice. However, compared to noninterference-based approaches, techniques based on taint tracking often lack formal underpinnings that allow to reason about the guarantees that they provide.

0.4 Challenges

Flexible Policies While noninterference provides a solid baseline for reasoning about the security of programs, it can be very restrictive. Many useful programs, such as the login program example in Section 0.2, do not satisfy noninterference. While noninterference policies can be relaxed by adding support for *declassifying* sensitive data [271], declassification policies may be complicated to reason about in practice. Hence, providing a natural and flexible way to specify security policies remains an open problem.

Formal Guarantees Despite renewed interest and recent advances in information-flow research [266], the results are largely unused outside of academia. On the other hand, some approaches such as taint tracking have enjoyed considerable success in practice ranging from bug detection to ensuring confidentiality; moreover the technique has been applied to high-level languages and machine code.

However, the actual security policy enforced by practical approaches often remains unclear; as a consequence, evaluating practical security mechanisms in terms of soundness and precision is not possible in an unambiguous way. Hence, providing formal justification and analysis of practical techniques is an important avenue of research for bridging the gap between academic results and industry practice.

Information-Flow Control for Complex Language Features Another stumbling block for information-flow control in practice is the gap between minimal languages typically considered in research papers and programming language features used in practice. Extending information-flow control techniques to accommodate settings such as shared memory concurrency, modern databases, and web application scenarios presents an opportunity to make security verification more feasible in practice.

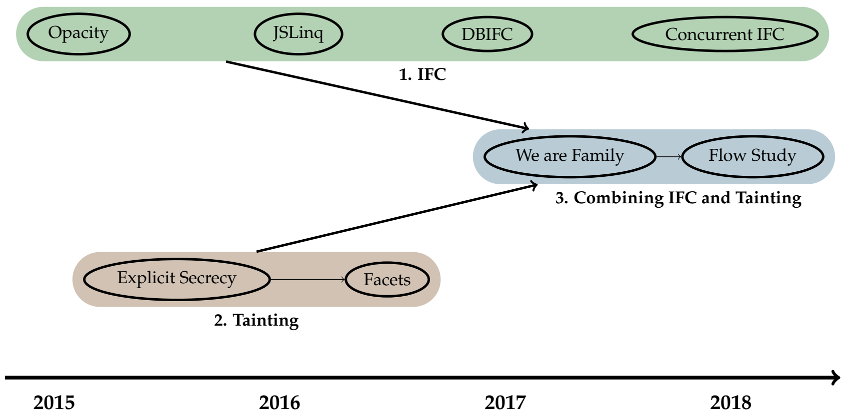

Figure 2: Overview of research tracks in the thesis

0.5 Contributions

This thesis consists of eight papers (Chapters 1-8). Five papers have been published in peer-reviewed conferences while the remaining three are currently under submission. This section outlines each paper's contributions and connects its topic to the challenges outlined in the previous sections. Broadly speaking, the papers fall into three categories:

- Formally analyzing of the security guarantees provided by taint trackers, and building more precise enforcement based on this formal characterization.

- Extending information-flow control to challenging language features and policies encountered in practical scenarios. In particular, we consider shared memory concurrency, declassification policies, code interacting with databases, and web applications.

- Bridging the gap between taint tracking and information-flow by investigating intermediate security properties and what trade-offs they offer for real-world code.

The structure of the research is summarized in Figure 2.

Chapters 1 and 2 aim to make security verification more feasible by exploring a more permissive family of security mechanisms known as *taint tracking* that allows for more practical enforcement mechanisms while nevertheless capturing some realistic use cases:

- Chapter 1 presents a formal characterization of the security property enforced by taint tracking tools. In order to understand the guarantees and ramifications of the taint tracking approach, we need to be able to characterize the attacks that are prevented, or not, by such techniques. In this chapter we present a general security condition capturing which information flows are tracked, and which are ignored, by taint tracking.

- Chapter 2 leverages the security condition from Chapter 5 to construct a general, and in some cases more precise, technique for taint tracking based on the concept of faceted values.

In Chapters 3 through 6, we explore how to extend information-flow control mechanisms to accommodate challenging language features often found in real applications:

- Chapter 3 tackles the problem of information-flow security for programs with shared-memory concurrency requiring expressive declassification policies. We provide a way to accommodate complex thread interaction and controlled information release by decoupling reasoning about functional correctness from information-flow reasoning.

- Chapter 4 investigates how to track information flows in multi-tiered web application that combine databases with both client-side and server-side code.

- Chapter 5 extends information-flow control conditions and enforcement mechanisms to applications interacting with databases making use of advanced features such as triggers, as well as dynamic security policies that change over time.

- Chapter 6 explores more alternative policies for specifying under what circumstances information can be released by investigating *opacity* as another property to specify information-flow policies and relating opacity to noninterference.

Chapters 7 and 8 aim to bridge the gap between noninterference-based approaches to information-flow control and techniques such as taint tracking, that are more easily enforceable:

- Chapter 7 presents a security condition that falls between noninterference and definitions based on taint tracking by extending taint tracking to catch another type of dangerous information flows, called *observable implicit flows* without incurring the same number of false positives as noninterference-based approaches.

- Chapter 8 studies the impact of different types of information flows in the context of real-world JavaScript code to reason about the relative importance of accurately catching different types of information flows.

The rest of this section lists the abstracts of the individual chapters:

0.5.1 Explicit Secrecy: A Policy for Taint Tracking

Daniel Schoepe, Musard Balliu, Benjamin C. Pierce, and Andrei Sabelfeld

Taint tracking is a popular security mechanism for tracking data-flow dependencies, both in high-level languages and at the machine code level. But despite the many taint trackers in practical use, the question of what, exactly, tainting *means*—what security policy it embodies—remains largely unexplored.

We propose *explicit secrecy*, a generic framework capturing the essence of explicit flows, i.e., the data flows tracked by tainting. The framework is semantic, generalizing previous syntactic approaches to formulating soundness criteria of tainting. We demonstrate the usefulness of the framework by instantiating it with both a simple high-level imperative language and an idealized RISC machine. To further understanding of what is achieved by taint tracking tools, both dynamic and static, we obtain soundness results with respect to explicit secrecy for the tainting engine cores of a collection of dynamic and static taint trackers.

Statement of Contribution This paper was co-authored with Musard Balliu, Benjamin C. Pierce, and Andrei Sabelfeld. Daniel contributed to the framework. Daniel is responsible for formalizing and proving the results about the explicit secrecy framework, and for formalizing the enforcement mechanisms as well as the soundness proof of the static and dynamic enforcement mechanisms.

Appeared in: *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P), Saarbrücken, Germany, March 2016*

0.5.2 Let's Face It: Faceted Values for Taint Tracking

Daniel Schoepe, Musard Balliu, Frank Piessens, and Andrei Sabelfeld

Taint tracking has been successfully deployed in a range of security applications to track data dependencies in hardware and machine-, binary-, and high-level code. Precision of taint tracking is key for its success in practice: being a vulnerability analysis, false positives must be low for the analysis to be practical. This paper presents an approach to taint tracking, which does not involve tracking taints throughout computation. Instead, we include shadow memories in the execution context, so that a single run of a program has the effect of computing on both tainted and untainted data. This mechanism is inspired by the technique of secure multi-execution, while in contrast to the latter it does not require running the entire program multiple times. We present a general framework and establish its soundness with respect to explicit secrecy, a policy for preventing insecure data leaks, and its precision showing that runs of secure programs are never modified. We show that the technique can be used for attack detection with no false positives. To evaluate the mechanism in practice, we implement DroidFace, a source-to-source transform for an intermediate Java-like language and benchmark its precision and performance with respect to representative static and dynamic taint trackers for Android. The results indicate that the performance penalty is tolerable while achieving both soundness and no false positives on the tested benchmarks. The key results of this paper have been formalized in Isabelle/HOL.

Statement of Contribution This paper was co-authored with Musard Balliu, Frank Piessens, and Andrei Sabelfeld. Daniel is responsible for formalizing and proving soundness and precision of the results, as well as the prototype implementation. All authors contributed equally to the writing.

Appeared in: *Proceedings of the 21st European Symposium on Research in Computer Security (ESORICS), Heraklion, Greece, 2016*

0.5.3 VERONICA: Verified Concurrent Information-Flow Security Unleashed

Daniel Schoepe, Toby Murray, Andrei Sabelfeld

Methods for proving that concurrent software does not leak its secrets has remained an active topic of research for at least the past four decades. Despite an impressive array of work, the present situation remains highly unsatisfactory. With contemporary compositional proof methods one is forced to choose between *expressiveness* (the ability to reason about a wide variety of security policies), on the one hand, and *precision* (the ability to reason about complex thread interactions and program behaviours), on the other. Achieving both is essential and, we argue, requires a new style of compositional program logic.

We present the first compositional program logic for proving concurrent programs information flow secure that supports high-precision reasoning about a wide range of security policies and program behaviours (e.g. expressive declassification, value-dependent classification, secret-dependent branching). Just as importantly, our approach embodies a new way for engineering such logics that can be re-used elsewhere, called *decoupled functional correctness* (DFC). DFC leads to a substantially simpler logic, even while achieving this unprecedented combination of features. We demonstrate the virtues and versatility of our approach by verifying a range of example programs, beyond the reach of prior methods. All developments are formalized in Isabelle/HOL.

Statement of Contribution This paper was co-authored with Toby Murray, and Andrei Sabelfeld. Daniel is responsible for developing the enforcement framework, Isabelle formalization, soundness results, and case studies. All authors contributed equally to the writing and overall development of the approach.

Under submission.

0.5.4 JSLINQ: Building Secure Applications across Tiers

Musard Balliu, Benjamin Liebe, Daniel Schoepe, and Andrei Sabelfeld

This paper proposes *JSLINQ*, a framework for writing web applications with end-to-end security guarantees. Modern web applications consist of several tiers, often including server-side code, a database, and client-side JavaScript code. In order to achieve end-to-end security, correct communication between tiers must be ensured. JSLINQ leverages meta-programming facilities in F# and the *WebSharper* framework to provide a unified language for securely writing the entire web application. A security type system is then used to guarantee noninterference for well-typed programs.

Aside from formal soundness results, we investigate the practicality of this approach with several case studies, such as location-based services and a Battleship browser game, indicating that JSLINQ can handle practical scenarios; this work is therefore a step toward implementing *information-flow control in practice*.

Statement of Contribution This paper was co-authored with Musard Balliu, Benjamin Liebe, and Andrei Sabelfeld. Daniel contributed to the type inference engine, the language semantics, type system, and the case studies. All authors contributed equally to the writing.

Appeared in: *Proceedings of the ACM Conference on Data and Applications Security and Privacy (CODASPY), New Orleans, LA, March 2016*

0.5.5 Information-Flow Control for Database-Backed Applications

Marco Guarnieri, Daniel Schoepe, Musard Balliu, David Basin, and Andrei Sabelfeld

Securing database-backed applications requires tracking information across the program and the database together, since securing each component in isolation may still result in an overall insecure system. Current research extends language-based techniques with models capturing the database's behavior. Previous work, however, relies on simplistic database models, which ignore security-relevant features that may leak sensitive information.

We propose a novel security monitor for database-backed applications. Our monitor tracks fine-grained dependencies between variables and database tuples by leveraging database theory concepts like disclosure lattices and query determinacy. It also accounts for a realistic database model that supports security-critical constructs like triggers and dynamic policies. The monitor automatically synthesizes program-level code that replicates the behavior of database features like triggers, thereby tracking information flows inside the database. We also introduce symbolic tuples, an efficient approximation of dependency-tracking over disclosure lattices. We implement our monitor for database-backed Scala programs and demonstrate its effectiveness on four case studies.

Statement of Contribution This paper was co-authored with Marco Guarnieri, Musard Balliu, David Basin, and Andrei Sabelfeld. Daniel contributed to the formalization and framework and is responsible for the prototype implementation. All authors contributed equally to writing the paper.

Under submission.

0.5.6 Understanding and Enforcing Opacity

Daniel Schoepe and Andrei Sabelfeld

This paper explores *opacity*, a policy providing more flexible control over what part of a system needs to be kept confidential. Concretely, instead of protecting pieces of data, opacity expresses that properties about the input need to remain secret; for example, a user may not want to disclose whether he is located in a sensitive area, but is okay with disclosing parts of his location otherwise. Hence, this work falls under the challenge of *flexible policies*.

The paper provides a general framework for opacity, parametrized in the power of the attacker, and connects opacity to noninterference. Moreover, we explore two dynamic enforcement techniques and provide a proof-of-concept implementation. All theoretical results are formalized using the Isabelle/HOL proof assistant [244].

Statement of Contribution This paper was co-authored with Andrei Sabelfeld. Daniel is responsible for the proofs of the theoretical results, implementation work, and the Isabelle/HOL formalization. All authors contributed equally to writing the paper.

Appeared in: *Proceedings of the IEEE Computer Security Foundations Symposium (CSF), Verona, Italy, July 2015*

0.5.7 We Are Family: Relating Information-Flow Trackers

Musard Balliu, Daniel Schoepe, and Andrei Sabelfeld

While information-flow security is a well-established area, there is an unsettling gap between heavyweight *information-flow control*, with formal guarantees yet limited practical impact, and lightweight *tainting* techniques, useful for bug finding yet lacking formal assurance. This paper proposes a framework for exploring the middle ground in the range of enforcement from tainting (tracking data flows only) to fully-fledged information-flow control (tracking both data and control flows). We formally illustrate the trade-offs between the soundness and permissiveness that the framework allows to achieve. The framework is deployed in a staged fashion, statically embedding a dynamic monitor, being parametric in security policies, as they do not need to be fixed until the final deployment. This flexibility facilitates a secure app store architecture, where the static stage of verification is performed by the app store and the dynamic stage is deployed on the client. To illustrate the practicality of the framework, we implement our approach for a core of Java and evaluate it on a use case with enforcing privacy policies in the Android setting. We also show how a state-of-the-art dynamic monitor for JavaScript can be easily adapted to implement our approach.

Statement of Contribution This paper was co-authored with Musard Balliu and Andrei Sabelfeld. Daniel contributed to the security and enforcement definitions and is responsible for the prototype implementation. All authors contributed equally to writing the paper.

Appeared in: *Proceedings of the 22nd European Symposium on Research in Computer Security (ESORICS), Oslo, Norway, 2017*

0.5.8 An Empirical Study of Information Flows in Real-World JavaScript

Cristian-Alexandru Staicu, Daniel Schoepe, Musard Balliu, Michael Pradel, and Andrei Sabelfeld

Information flow analysis prevents secret or untrusted data from flowing into public or trusted sinks. Existing mechanisms cover a wide array of options, ranging from lightweight taint analysis to heavyweight information flow control that also considers implicit flows. Dynamic analysis, which is particularly popular for languages such as JavaScript, faces the question whether to invest in analyzing flows caused by not executing a particular branch, so-called hidden implicit flows. This paper addresses the questions how common different kinds of flows are in real-world programs, how important these flows are to enforce security policies, and how costly it is to consider these flows. We address these questions in an empirical study that analyzes 56 real-world JavaScript programs that suffer from various security problems, such as code injection vulnerabilities, denial of service vulnerabilities, memory leaks, and privacy leaks. The study is based on a state-of-the-art dynamic information flow analysis and a formalization of its core. We find that implicit flows are expensive to track in terms of permissiveness, label creep, and runtime overhead. We find a lightweight taint analysis to be sufficient for most of the studied security problems, while for some privacy-related code, observable tracking is sometimes required. In contrast, we do not find any evidence that tracking hidden implicit flows reveals otherwise missed security problems. Our results help security analysts and analysis designers to understand the cost-benefit tradeoffs of information flow analysis and provide empirical evidence that analyzing implicit flows in a cost-effective way is a relevant problem.

Statement of Contribution This paper was co-authored with Cristian-Alexandru Staicu, Musard Balliu, Michael Pradel, and Andrei Sabelfeld. Daniel contributed to the formalization and framework and the soundness results. All authors contributed equally to writing the paper.

Under submission.

BIBLIOGRAPHY

[1] Catalog Reveals NSA Has Back Doors for Numerous Devices - SPIEGEL ONLINE. http://www.spiegel.de/international/world/catalog-reveals-nsa-has-back-doors-for-numerous-devices-a-940994.html. Accessed: 2016-2-19.

[2] Data Mining for Terrorists - Schneier on Security. https://www.schneier.com/blog/archives/2006/03/data_mining_for.html. Accessed: 2016-2-15.

[3] Dynamic instrumentation tool platform. http://www.dynamorio.org/home.html.

[4] Edward Snowden, after months of NSA revelations, says his mission's accomplished - The Washington Post. https://www.washingtonpost.com/world/national-security/edward-snowden-after-months-of-nsa-revelations-says-his-missions-accomplished/2013/12/23/49fc36de-6c1c-11e3-a523-fe73f0ff6b8d_story.html. Accessed: 2016-2-15.

[5] IDS03-J. Do not log unsanitized user input - CERT Oracle Coding Standard for Java - CERT Secure Coding Standards. https://www.securecoding.cert.org/confluence/display/java/IDS03-J.+Do+not+log+unsanitized+user+input. Accessed: 2015-8-5.

[6] Locking ruby in the safe. http://phrogz.net/programmingruby/taint.html.

[7] Mass Surveillance Isn't the Answer to Fighting Terrorism - The New York Times. http://www.nytimes.com/2015/11/18/opinion/mass-surveillance-isnt-the-answer-to-fighting-terrorism.html?_r=0. Accessed: 2016-2-15.

[8] NSA slapped malware on 50,000+ networks, says report - CNET. http://www.cnet.com/news/nsa-slapped-malware-on-50000-networks-says-report/. Accessed: 2016-2-19.

[9] Perl security and taint mode. http://perldoc.perl.org/perlsec.html.

[10] Timeline of NSA Domestic Spying — Electronic Frontier Foundation. https://www.eff.org/nsa-spying/timeline. Accessed: 2016-2-15.

[11] Valgrind. http://valgrind.org/.

[12] Apache Cordova. http://cordova.apache.org/, 2015. Accessed: 2015-09-11.

[13] Attribute-Based Mapping. https://msdn.microsoft.com/en-us/library/bb386971.aspx, 2015. Accessed: 2015-09-11.

[14] Critical Security Controls. http://www.sans.org/critical-security-controls/, 2015. Accessed: 2015-08-25.

[15] F# Compiler Services. `http://fsharp.github.io/FSharp.Compiler.Service/`, 2015. Accessed: 2015-09-11.

[16] FParsec. `http://www.quanttec.com/fparsec/`, 2015. Accessed: 2015-09-11.

[17] 'Mouse over' security flaw causes Twitter trouble. `http://edition.cnn.com/2010/TECH/social.media/09/21/twitter.security.flaw/`, 2015. Accessed: 2015-08-25.

[18] OWASP Top 10 2013. `https://www.owasp.org/index.php/Top_10_2013-Top_10`, 2015. Accessed: 2015-08-25.

[19] Sites hit in massive web attack. `http://www.bbc.com/news/technology-12933053`, 2015. Accessed: 2015-08-25.

[20] WebSharper. `http://websharper.com/`, 2015. Accessed: 2015-08-25.

[21] Linq (language-integrate query), *Microsoft MSDN Library*, May 2016.

[22] Babel JavaScript compiler. `https://babeljs.io`, Accessed: 2018-02-08.

[23] M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. A core calculus of dependency. In *POPL*, 1999.

[24] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[25] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juárez, Arvind Narayanan, and Claudia Díaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 674–689, 2014.

[26] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140. ACM, 2013.

[27] Torben Amtoft, Sruthi Bandhakavi, and Anindya Banerjee. A logic for information flow in object-oriented programs. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 91–102, 2006.

[28] Torben Amtoft and Anindya Banerjee. Information flow analysis in logical form. In *Proc. Symp. on Static Analysis*, pages 100–115, 2004.

[29] Saswat Anand, Mayur Naik, Mary Jean Harrold, and Hongseok Yang. Automated concolic testing of smartphone apps. In *SIGSOFT FSE*, page 59, 2012.

[30] Gregory R Andrews and Richard P Reitman. An axiomatic approach to information flow in parallel programs. Technical report, Cornell University, 1978.

[31] Gregory R Andrews and Richard P Reitman. An axiomatic approach to information flow in programs. *ACM TOPLAS*, 2(1):56–76, 1980.

[32] Anonymized. DAISY: DAtabase and Information-flow SecuritY. `https://sites.google.com/site/databaseifc/`, 2018.

[33] Anonymized. Information-Flow Control for Database-backed Applications – Technical Report. `https://sites.google.com/site/databaseifc/`, 2018.

[34] Owen Arden, Jed Liu, and Andrew C. Myers. Flow-limited authorization. In *CSF*, 2015.

[35] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *PLDI*, 2014.

[36] A. Askarov and S. Chong. Learning is change in knowledge: Knowledge-based security for dynamic policies. In *CSF*, pages 308–322, 2012.

[37] A. Askarov, S. Hunt, A. Sabelfeld, and D. Sands. Termination-insensitive noninterference leaks more than just a bit. In *ESORICS*, 2008.

[38] A. Askarov and A. Myers. A semantic framework for declassification and endorsement. In *ESOP*, 2010.

[39] A. Askarov and A. Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *S&P*, 2007.

[40] Aslan Askarov and Andrei Sabelfeld. Tight enforcement of information-release policies for dynamic languages. In *CSF*, 2009.

[41] T. H. Austin and C. Flanagan. Multiple facets for dynamic information flow. In *POPL*, POPL '12, pages 165–178, 2012.

[42] Thomas H. Austin and Cormac Flanagan. Efficient purely-dynamic information flow analysis. *PLAS*, 2009.

[43] Thomas H. Austin and Cormac Flanagan. Permissive dynamic information flow analysis. In *Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, PLAS '10, pages 3:1–3:12, 2010.

[44] Thomas H. Austin and Cormac Flanagan. Permissive dynamic information flow analysis. In *Workshop on Programming Languages and Analysis for Security (PLAS 2010)*, page 3. ACM, 2010.

[45] Thomas H. Austin, Jean Yang, Cormac Flanagan, and Armando Solar-Lezama. Faceted execution of policy-agnostic programs. In *PLAS*, 2013.

[46] M. Balliu, M. Dam, and R. Guanciale. Automating information flow analysis of low level code. In *CCS*, 2014.

[47] M. Balliu, B. Liebe, D. Schoepe, and A. Sabelfeld. JSLINQ: Building Secure Applications across Tiers. `https://sites.google.com/site/jslinqcodaspy16/`, September 2015. Software and Extended Version.

[48] Musard Balliu. A logic for information flow analysis of distributed programs. In *NordSec*, 2013.

[49] Musard Balliu, Mads Dam, and Gurvan Le Guernic. Epistemic temporal logic for information flow security. In *PLAS*, 2011.

[50] Musard Balliu, Mads Dam, and Gurvan Le Guernic. ENCoVer: Symbolic Exploration for Information Flow Security. In *Proceedings of the IEEE Computer Security Foundations Symposium*, pages 30–44, june 2012.

[51] Musard Balliu, Benjamin Liebe, Daniel Schoepe, and Andrei Sabelfeld. Jslinq: Building secure applications across tiers. In *CODASPY*, 2016.

[52] Musard Balliu, Daniel Schoepe, and Andrei Sabelfeld. We are family: Relating information-flow trackers. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*, pages 124–145, 2017.

[53] Ioannis G. Baltopoulos and Andrew D. Gordon. Secure compilation of a multi-tier web language. In *TLDI*, 2009.

[54] Anindya Banerjee, David A Naumann, and Stan Rosenberg. Expressive declassification policies and modular static enforcement. In *Security & Privacy*, pages 339–353. IEEE Computer Society, 2008.

[55] Tao Bao, Yunhui Zheng, Zhiqiang Lin, Xiangyu Zhang, and Dongyan Xu. Strict control dependence and its effect on dynamic information flow analyses. In *Proceedings of the 19th International Symposium on Software Testing and Analysis*, ISSTA '10, pages 13–24. ACM, 2010.

[56] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *CAV*, 2011.

[57] G. Barthe, G. Betarte, J. D. Campo, C. D. Luna, and D. Pichardie. System-level non-interference for constant-time cryptography. In *CCS*, 2014.

[58] G. Barthe, J. M. Crespo, D. Devriese, F. Piessens, and E. Rivas. Secure multi-execution through static program transformation. In *FMOODS/FORTE*, 2012.

[59] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. *MSCS*, 2011.

[60] Iulia Bastys, Frank Piessens, and Andrei Sabelfeld. Prudent design principles for information flow control. In *PLAS*, October 2018.

[61] Iulia Bastys, Frank Piessens, and Andrei Sabelfeld. Tracking information flow via delayed output. In *NordSec 2018*, 2018.

[62] Lujo Bauer, Shaoying Cai, Limin Jia, Timothy Passaro, Michael Stroucken, and Yuan Tian. Run-time monitoring and formal analysis of information flows in chromium. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*. The Internet Society, 2015.

[63] Thomas Bauereiß, Armando Pesenti Gritti, Andrei Popescu, and Franco Raimondi. CoSMeDis: a distributed social media platform with formally verified confidentiality guarantees. In *Security & Privacy*, pages 729–748, 2017.

[64] Let's face it: Faceted values for taint tracking. Full version and implementation. `http://www.cse.chalmers.se/research/group/security/facets`.

[65] Mark Beaumont, Jim McCarthy, and Toby Murray. The cross domain desktop compositor: using hardware-based video compositing for a multi-level secure user interface. In *Proc. Annual Computer Security Applications Conference*, pages 533–545. ACM, 2016.

[66] D. Elliott Bell and Leonard J. La Padula. Secure computer system: Unified exposition and Multics interpretation. Technical Report MTR-2997, MITRE Corp., March 1976.

[67] Gabriel Bender, Lucja Kot, and Johannes Gehrke. Explainable security for relational databases. In *SIGMOD*, 2014.

[68] Gabriel Bender, Lucja Kot, Johannes Gehrke, and Christoph Koch. Fine-grained disclosure control for app ecosystems. In *SIGMOD*, 2013.

[69] Béatrice Bérard, John Mullins, and Mathieu Sassolas. Quantifying opacity. In *QEST*, 2010.

[70] Lennart Beringer. Relational decomposition. In *Interactive Theorem Proving - Second International Conference, ITP 2011, Berg en Dal, The Netherlands, August 22-25, 2011. Proceedings*, pages 39–54, 2011.

[71] Lennart Beringer. End-to-end multilevel hybrid information flow control. In Ranjit Jhala and Atsushi Igarashi, editors, *APLAS*, 2012.

[72] Lennart Beringer and Martin Hofmann. Secure information flow and program logics. In *Proc. IEEE CSF*, pages 233–248, 2007.

[73] K. J. Biba. Integrity considerations for secure computer systems. Technical report, MITRE Corp., 04 1977.

[74] Abhishek Bichhawat, Vineet Rajani, Deepak Garg, and Christian Hammer. Information flow control in WebKit's JavaScript bytecode. In *Principles of Security and Trust (POST 2014)*, volume 8414 of *LNCS*, pages 159–178. Springer, 2014.

[75] Abhishek Bichhawat, Vineet Rajani, Jinank Jain, Deepak Garg, and Christian Hammer. Webpol: Fine-grained information flow policies for web browsers. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*, pages 242–259, 2017.

[76] N. Bielova and T. Rezk. A taxonomy of information flow monitors. In *POST*, 2016.

[77] Nataliia Bielova. Survey on JavaScript security policies and their enforcement mechanisms in a web browser. *JLAP*, 2013.

[78] Arnar Birgisson, Daniel Hedin, and Andrei Sabelfeld. Boosting the permissiveness of dynamic information-flow tracking by testing. In *European Symposium on Research in Computer Security (ESORICS 2012)*, volume 7459 of *LNCS*, pages 55–72. Springer, 2012.

[79] Arnar Birgisson, Alejandro Russo, and Andrei Sabelfeld. Unifying facets of information integrity. In *ICISS*, 2010.

[80] BNF Converter. `http://bnfc.digitalgrammars.com/`, 2014.

[81] M. Bodin, T. Jensen, and A. Schmitt. Pretty-big-step-semantics-based certified abstract interpretation (preliminary version). In *Semantics, Abstract Interpretation, and Reasoning about Programs: Essays Dedicated to David A. Schmidt on the Occasion of his Sixtieth Birthday*, 2013.

[82] A. Bohannon, B. Pierce, V. Sjöberg, S. Weirich, and S. Zdancewic. Reactive Noninterference. In *ACM CCS*, November 2009.

[83] Alexandre Boisseau. *Abstractions pour la vérification de propriétés de sécurité de protocoles cryptographiques*. PhD thesis, École Normale Supérieure de Cachan, September 2003.

[84] I. Boloşteanu and D. Garg. Asymmetric secure multi-execution with declassification. In *POST*, 2016.

[85] Annalisa Bossi, Carla Piazza, and Sabina Rossi. Compositional information flow security for concurrent programs. *J. Computer Security*, 15(3):373–416, 2007.

[86] Niklas Broberg and David Sands. Flow-sensitive semantics for dynamic information flow policies. In *Programming Languages and Analysis for Security*, pages 101–112, 2009.

[87] Niklas Broberg and David Sands. Paralocks: role-based information flow control and beyond. In *Proc. ACM Symp. on Principles of Programming Languages*, volume 45, pages 431–444, 2010.

[88] Niklas Broberg, Bart van Delft, and David Sands. The anatomy and facets of dynamic policies. In *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 122–136, 2015.

[89] K. Browder and M.A. Davidson. The virtual private database in Oracle9iR2. *Oracle Technical White Paper, Oracle Corporation*, 500, 2002.

[90] D. Brumley, I. Jager, T. Avgerinos, and E. J. Schwartz. BAP: A binary analysis platform. In *CAV*, 2011.

[91] Jeremy Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity Generalised to Transition Systems. In *FAST*, 2005.

[92] Jeremy Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. *Int. J. Inf. Sec.*, 2008.

[93] Jeremy Bryans, Maciej Koutny, and Peter Y. A. Ryan. Modelling opacity using petri nets. *Electr. Notes Theor. Comput. Sci.*, 2005.

[94] Eugene Burmako. Scala macros: let our powers combine!: on how rich syntax and static types work with metaprogramming. In *SCALA@ECOOP*, 2013.

[95] Caffeinemark. `http://www.benchmarkhq.ru/cm30/`.

[96] Caffeinemark for android. `https://play.google.com/store/apps/details?id=com.android.cm3`.

[97] Stefano Calzavara, Ilya Grishchenko, and Matteo Maffei. Horndroid: Practical and sound security static analysis of android applications by smt solving. In *EuroS&P*, 2016.

[98] Roberto Capizzi, Antonio Longo, V. N. Venkatakrishnan, and A. Prasad Sistla. Preventing information leaks through shadow executions. In *ACSAC*, 2008.

[99] Ethan Cecchetti, Andrew C Myers, and Owen Arden. Nonmalleable information flow control. In *ACM CCS*, pages 1875–1891, 2017.

[100] Deepak Chandra and Michael Franz. Fine-grained information flow analysis and enforcement in a java virtual machine. In *23rd Annual Computer Security Applications Conference (ACSAC 2007), December 10-14, 2007, Miami Beach, Florida, USA*, pages 463–475. IEEE, 2007.

[101] W. Chang, B. Streiff, and C. Lin. Efficient and extensible security enforcement using dynamic data flow analysis. In *CCS*, 2008.

[102] A. Chaudhuri, P. Naldurg, and S. K. Rajamani. A type system for data-flow integrity on windows vista. *SIGPLAN Notices*, 2008.

[103] James Cheney. A formal framework for provenance security. In *CSF*, 2011.

[104] James Cheney, Sam Lindley, and Philip Wadler. A practical theory of language-integrated query. In *ICFP*, 2013.

[105] W. Cheng, Q. Zhao, B. Yu, and S. Hiroshige. TaintTrace: Efficient flow tracing with dynamic binary rewriting. In *ISCC*, 2006.

[106] Adam Chlipala. Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications. In *OSDI*, 2010.

[107] S. Chong, K. Vikram, and A. C. Myers. SIF: Enforcing Confidentiality and Integrity in Web Applications. In *USENIX Security*, 2007.

[108] Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng. Secure web applications via automatic partitioning. *Comm. of the ACM*, 2009.

[109] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum. Understanding data lifetime via whole system simulation. In *USENIX Security Symposium*, 2004.

[110] Andrey Chudnov and David A. Naumann. Information flow monitor inlining. In *CSF*, 2010.

[111] Andrey Chudnov and David A. Naumann. Inlined information flow monitoring for JavaScript. In *ACM Conference on Computer and Communications Security (CCS 2015)*, pages 629–643. ACM, 2015.

[112] Ravi Chugh, Jeffrey A. Meister, Ranjit Jhala, and Sorin Lerner. Staged information flow for javascript. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 50–62, 2009.

[113] Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. *Acm sigplan notices*, 46(4), 2011.

[114] D. Clark and S. Hunt. Non-interference for deterministic interactive programs. In *Workshop on Formal Aspects in Security and Trust (FAST'08)*, October 2008.

[115] M. R. Clarkson and F. B. Schneider. Hyperproperties. *JCS*, 2010.

[116] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *POST*, 2014.

[117] J. A. Clause, W. Li, and A. Orso. Dytan: A generic dynamic taint analysis framework. In *ISSTA*, pages 196–206. ACM, 2007.

[118] E. S. Cohen. Information transmission in sequential programs. In *FSC*. Academic Press, 1978.

[119] J. J. Conti and A. Russo. A taint mode for Python via a library. In *NordSec*, 2010.

[120] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter. Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *S&P*, 2009.

[121] Brian J. Corcoran, Nikhil Swamy, and Michael W. Hicks. Cross-tier, label-based security enforcement for web applications. In *SIGMOD*, 2009.

[122] J. R. Crandall and F. T. Chong. Minos: Control data attack prevention orthogonal to memory model. In *MICRO*, 2004.

[123] M. Dam, G. Le Guernic, and A. Lundblad. Treedroid: a tree automaton based approach to enforcing data processing policies. In *CCS*, 2012.

[124] Ádám Darvas, Reiner Hähnle, and David Sands. A theorem proving approach to analysis of secure information flow. In *International Conference on Security in Pervasive Computing*, SPC'05, pages 193–209. Springer, 2005.

[125] Brian A Davey and Hilary A Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

[126] Benjamin Davis and Hao Chen. DBTaint: cross-application information flow tracking via databases. In *WebApps*, 2010.

[127] Willem De Groef, Dominique Devriese, Nick Nikiforakis, and Frank Piessens. Flowfox: A web browser with flexible and precise information flow control. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 748–759, 2012.

[128] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[129] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513, 1977.

[130] Dorothy E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, 1976.

[131] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, Boston, MA, USA, 1982.

[132] Dorothy E Denning and Teresa F Lunt. A multilevel relational data model. In *S&P*, 1987.

[133] Dominique Devriese and Frank Piessens. Noninterference through secure multi-execution. In *S&P*, 2010.

[134] Mohan Dhawan and Vinod Ganapathy. Analyzing information flow in JavaScript-based browser extensions. In *Annual Computer Security Applications Conference (ACSAC 2009)*, pages 382–391. IEEE, 2009.

[135] U. Dhawan, N. Vasilakis, R. Rubin, S. Chiricescu, J. M. Smith, T. F. Knight, B. C. Pierce, and A. DeHon. PUMP – A programmable unit for metadata processing. In *HASP*, 2014.

[136] C. Dima, C. Enea, and R. Gramatovici. Nondeterministic noninterference and deducible information flow. Technical Report 2006-01, University of Paris 12, LACL, 2006.

[137] Rayna Dimitrova, Bernd Finkbeiner, Máté Kovács, Markus N. Rabe, and Helmut Seidl. Model checking information flow in reactive systems. In *VMCAI*, pages 169–185, 2012.

[138] Droidbench: A micro-benchmark suite to assess the stability of taint-analysis tools for android. `https://github.com/secure-software-engineering/DroidBench`.

[139] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicholas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. The matter of heartbleed. In *Internet Measurement Conference (IMC)*, pages 475–488, 2014.

[140] Sebastian Eggert and Ron van der Meyden. Dynamic intransitive noninterference revisited. 29(6):1087–1120, 2017.

[141] W. Enck, P. Gilbert, S. Han, V. Tendulkar, Byung-Gon Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Trans. Comput. Syst.*, 2014.

[142] Michael D. Ernst, René Just, Suzanne Millstein, Werner Dietl, Stuart Pernsteiner, Franziska Roesner, Karl Koscher, Paulo Barros Barros, Ravi Bhoraskar, Seungyeop Han, Paul Vines, and Edward X. Wu. Collaborative verification of information flow for a high-assurance app store. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1092–1104, 2014.

[143] Csilla Farkas and Sushil Jajodia. The inference problem: A survey. *SIGKDD Explorations*, 2002.

[144] J. S. Fenton. Memoryless subsystems. *Comput. J.*, 17(2):143–147, 1974.

[145] Andrew Ferraiuolo, Weizhe Hua, Andrew C Myers, and G Edward Suh. Secure information flow verification with mutable dependent types. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 6, 2017.

[146] Flickr. Flickr geoprivacy settings. `http://www.flickr.com/account/geo/privacy/`, 2011.

[147] Robert W. Floyd. Assigning meanings to programs. *Mathematical Aspects of Computer Science*, 19:19–32, 1967.

[148] Riccardo Focardi and Roberto Gorrieri. Classification of security properties (part i: Information flow). In *FOSAD*, 2000.

[149] Cédric Fournet, Nikhil Swamy, Juan Chen, Pierre-Évariste Dagand, Pierre-Yves Strub, and Benjamin Livshits. Fully abstract compilation to javascript. In *POPL '13*, 2013.

[150] Dario Freni, Carmen Ruiz Vicente, Sergio Mascetti, Claudio Bettini, and Christian S. Jensen. Preserving location and absence privacy in geo-social networks. In *CIKM*, 2010.

[151] R. Giacobazzi and I. Mastroeni. Abstract Non-Interference: Parameterizing Non-Interference by Abstract Interpretation. In *Proc. Principles of Programming Languages*, pages 186–197, New York, January 2004. ACM-Press.

[152] Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, David Mazières, John C. Mitchell, and Alejandro Russo. Hails: Protecting data privacy in untrusted web applications. In *OSDI*, 2012.

[153] J. A. Goguen and J. Meseguer. Security policies and security models. In *S&P*, pages 11–20, Apr 1982.

[154] J. A. Goguen and J. Meseguer. Unwinding and inference control. In *IEEE SP*, 1984.

[155] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard. Information flow analysis of android applications in droidsafe. In *NDSS*, 2015.

[156] M. Graa, N. Cuppens-Boulahia, F. Cuppens, and A. R. Cavalli. Detecting control flow in smarphones: Combining static and dynamic analyses. In *CSS*, 2012.

[157] Adam Granicz. Functional web and mobile development in F#. In *CEFP*, 2013.

[158] Eric Griffis, Jeffrey A Vaughan, and Todd Millstein. A platform for expressive and secure data sharing with untrusted third parties. Technical Report 120017, University of California, Los Angeles, 2011.

[159] Damas P. Gruska. Observation based system security. *Fundam. Inform.*, 2007.

[160] Damas P. Gruska. Informational analysis of security and integrity. *Fundam. Inform.*, 2012.

[161] Marco Guarnieri and David Basin. Optimal security-aware query processing. In *VLDB*, 2014.

[162] Marco Guarnieri, Srdjan Marinovic, and David Basin. Strong and provably secure database access control. In *EuroS&P*, 2016.

[163] G. Le Guernic. *Confidentiality Enforcement Using Dynamic Information Flow Analyses*. PhD thesis, Kansas State University, 2007.

[164] Joshua D. Guttman and Mark E. Nadel. What needs securing. In *CSFW*, 1988.

[165] V. Haldar, D. Chandra, and M. Franz. Dynamic taint propagation for Java. In *ACSAC*, 2005.

[166] Daniel Hedin, Luciano Bello, and Andrei Sabelfeld. Value-sensitive hybrid information flow control for a javascript-like language. In *CSF*, pages 351–365. IEEE, 2015.

[167] Daniel Hedin, Arnar Birgisson, Luciano Bello, and Andrei Sabelfeld. JS-Flow: tracking information flow in JavaScript and its APIs. In *SAC*, pages 1663–1671. ACM, 2014.
[168] Daniel Hedin and Andrei Sabelfeld. Information-flow security for a core of javascript. In *CSF*, pages 3–18. IEEE, 2012.
[169] Daniel Hedin and Andrei Sabelfeld. A perspective on information-flow control. In *Software Safety and Security - Tools for Analysis and Verification*, pages 319–347. 2012.
[170] Nevin Heintze and Jon G. Riecke. The SLam Calculus: Programming with Secrecy and Integrity. In *POPL*, 1998.
[171] Hewlett-Packard. Data execution prevention. http://h10032.www1.hp.com/ctg/Manual/c00387685.pdf, 2018. Accessed: 2018-11-12.
[172] Jaakko Hintikka. *Knowledge and belief*. Cornell University Press, 1962.
[173] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
[174] Charles Antony Richard Hoare. *Communicating sequential processes*. Prentice Hall, 1985.
[175] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications. In *CCS*, 2011.
[176] Catalin Hritcu, John Hughes, Benjamin C. Pierce, Antal Spector-Zabusky, Dimitrios Vytiniotis, Arthur Azevedo de Amorim, and Leonidas Lampropoulos. Testing noninterference, quickly. In *ICFP*, 2013.
[177] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo. Securing web application code by static analysis and runtime protection. In *WWW*, 2004.
[178] Dominic J. D. Hughes and Vitaly Shmatikov. Information hiding, anonymity and privacy: a modular approach. *JCS*, 2004.
[179] Sebastian Hunt and Isabella Mastroeni. The per model of abstract non-interference. In *SAS*, pages 171–185, 2005.
[180] Sebastian Hunt and David Sands. On flow-sensitive security types. In *POPL*, POPL '06, pages 79–90, New York, NY, USA, 2006. ACM.
[181] Sushil Jajodia and Ravi Sandhu. Polyinstantiation integrity in multilevel relations. In *S&P*, 1990.
[182] D. Jang, R. Jhala, S. Lerner, and H. Shacham. An empirical study of privacy-violating information flows in JavaScript web applications. In *CCS*, pages 270–283. ACM, 2010.
[183] Limin Jia, Jassim Aljuraidan, Elli Fragkaki, Lujo Bauer, Michael Stroucken, Kazuhide Fukushima, Shinsaku Kiyomoto, and Yutaka Miyake. Run-time enforcement of information-flow properties on android - (extended abstract). In *ESORICS*, 2013.
[184] Martin Johns. On javascript malware and related threats. *Journal in Computer Virology*, 4(3):161–178, 2008.
[185] Cliff B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. D.Phil. thesis, University of Oxford, June 1981.
[186] R. Joshi and K. R. M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 37(1–3), 2000.

[187] N. Jovanovic, C. Kruegel, and E. Kirda. Static analysis for detecting taint-style vulnerabilities in web applications. *JCS*, 2010.

[188] J. B. Kam and J. D. Ullman. Global data flow analysis and iterative algorithms. *J. ACM*, 1976.

[189] Min Gyung Kang, Stephen McCamant, Pongsin Poosankam, and Dawn Song. DTA++: dynamic taint analysis with targeted control-flow propagation. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*, 2011.

[190] Aleksandr Karbyshev, Kasper Svendsen, Aslan Askarov, and Lars Birkedal. Compositional non-interference for concurrent programs via separation and framing. 2018.

[191] Vineeth Kashyap, Ben Wiedermann, and Ben Hardekopf. Timing- and termination-sensitive secure information flow: Exploring a new approach. In *S&P*, 2011.

[192] Christoph Kerschbaumer, Eric Hennigan, Per Larsen, Stefan Brunthaler, and Michael Franz. Crowdflow: Efficient information flow security. In *ISC*, volume 7807 of *Lecture Notes in Computer Science*, pages 321–337. Springer, 2013.

[193] Narges Khakpour, Oliver Schwarz, and Mads Dam. Machine assisted proof of armv7 instruction level isolation properties. pages 276–291, 2013.

[194] Dave King, Boniface Hicks, Michael Hicks, and Trent Jaeger. Implicit flows: Can't live with 'em, can't live without 'em. In *International Conference on Information Systems Security (ICISS 2008)*, volume 5352 of *LNCS*, pages 56–70. Springer, 2008.

[195] John Krumm. A survey of computational location privacy. *PUC*, 2009.

[196] Lap-Chung Lam and Tzi-cker Chiueh. A general dynamic information flow tracking framework for security applications. In *ACSAC*, 2006.

[197] Gurvan Le Guernic. *Confidentiality Enforcement Using Dynamic Information Flow Analyses*. PhD thesis, Kansas State University, 2007.

[198] Sebastian Lekies, Ben Stock, and Martin Johns. 25 million flows later: large-scale detection of DOM-based XSS. In *ACM Conference on Computer and Communications Security (CCS 2013)*, pages 1193–1204. ACM, 2013.

[199] Li Li, Alexandre Bartel, Tegawendé F. Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. Iccta: Detecting inter-component privacy leaks in android apps. In *ICSE (1)*. IEEE, 2015.

[200] Peixuan Li and Danfeng Zhang. Towards a flow-and path-sensitive information flow analysis. In *Proc. IEEE CSF*, pages 53–67, 2017.

[201] Peng Li and Steve Zdancewic. Downgrading policies and relaxed noninterference. In *POPL*, 2005.

[202] Peng Li and Steve Zdancewic. Practical information flow control in web-based information systems. In *CSF*, 2005.

[203] Xiaowei Li and Yuan Xue. A survey on server-side approaches to securing web applications. *ACM Surv.*, 2014.

[204] Ximeng Li, Heiko Mantel, and Markus Tasch. Taming message-passing communication in compositional reasoning about confidentiality. In *Asian Symposium on Programming Languages and Systems (APLAS)*, pages 45–66, 2017.

[205] B. Livshits. Dynamic taint tracking in managed runtimes. Technical Report MSR-TR-2012-114, Microsoft, November 2012.

[206] B. Livshits and S. Chong. Towards fully automatic placement of security sanitizers and declassifiers. In *POPL*, 2013.

[207] V. Benjamin Livshits, Aditya V. Nori, Sriram K. Rajamani, and Anindya Banerjee. Merlin: specification inference for explicit information flow problems. In *PLDI*, 2009.

[208] Steffen Lortz, Heiko Mantel, Artem Starostin, Timo Bähr, David Schneider, and Alexandra Weber. Cassandra: Towards a Certifying App Store for Android. In *SPSM*, 2014.

[209] Luísa Lourenço and Luís Caires. Dependent information flow types. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 317–328, Mumbai, India, January 2015.

[210] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. The seaview security model. *TSE*, 16(6), 1990.

[211] Alexander Lux, Heiko Mantel, and Matthias Perner. Scheduler-independent declassification. In *International Conference on Mathematics of Program Construction*, pages 25–47, 2012.

[212] Jonas Magazinius, Alejandro Russo, and Andrei Sabelfeld. On-the-fly inlining of dynamic security monitors. pages 173–186. Springer Berlin Heidelberg, 2010.

[213] Heiko Mantel. Possibilistic definitions of security - an assembly kit. In *CSFW*, 2000.

[214] Heiko Mantel. On the composition of secure systems. In *Security & Privacy*, pages 88–101, 2002.

[215] Heiko Mantel and Alexander Reinhard. Controlling the What and Where of declassification in language-based security. In *Proc. European Symp. on Programming*, pages 141–156, 2007.

[216] Heiko Mantel, David Sands, and Henning Sudbrock. Assumptions and guarantees for compositional noninterference. In *Proc. IEEE CSF*, pages 218–232, Cernay-la-Ville, France, Jun 2011.

[217] Wes Masri and Andy Podgurski. Measuring the strength of information flows in programs. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 19(2), 2009.

[218] Isabella Mastroeni. Abstract interpretation-based approaches to security - a survey on abstract non-interference and its challenging applications. In *Festschrift for Dave Schmidt*, pages 41–65, 2013.

[219] Daniel Matichuk, Toby Murray, and Makarius Wenzel. Eisbach: A proof method language for Isabelle. *Journal of Automated Reasoning*, 56(3):261–282, 2016.

[220] Laurent Mazaré. Using unification for opacity properties. In *WITS*, 2004.

[221] Daryl McCullough. Specifications for multi-level security and a hook-up. In *Security & Privacy*, pages 161–161, 1987.

[222] John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *1994 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 16-18, 1994*, pages 79–93. IEEE Computer Society, 1994.

[223] William Melicher, Anupam Das, Mahmood Sharif, Lujo Bauer, and Limin Jia. Riding out DOMsday: Toward detecting and preventing DOM cross-

site scripting. In *Proceedings of the 25th Network and Distributed System Security Symposium*, 2018. To appear.

[224] Robin Milner. A theory of type polymorphism in programming. *J. Comput. Syst. Sci.*, 17(3):348–375, 1978.

[225] Robin Milner. *Communication and concurrency*. Prentice Hall, 1989.

[226] Scott Moore and Stephen Chong. Static analysis for efficient hybrid information-flow control. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, pages 146–160, 2011.

[227] Toby Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, Timothy Bourke, Sean Seefried, Corey Lewis, Xin Gao, and Gerwin Klein. seL4: from general purpose to a proof of information flow enforcement. In *Security & Privacy*, pages 415–429, May 2013.

[228] Toby Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, and Gerwin Klein. Noninterference for operating system kernels. pages 126–142, December 2012.

[229] Toby Murray, Robert Sison, and Kai Engelhardt. COVERN: A logic for compositional verification of information flow control. London, United Kingdom, April 2018.

[230] Toby Murray, Robert Sison, Edward Pierzchalski, and Christine Rizkallah. Compositional verification and refinement of concurrent value-dependent noninterference. In *Proc. IEEE CSF*, pages 417–431, June 2016.

[231] Toby Murray and Paul C. van Oorschot. BP: Formal proofs, the fine print and side effects. In *IEEE Cybersecurity Development Conference (SecDev)*, 2018. To appear.

[232] A. C. Myers, L. Zheng, S. Zdancewic, S. Chong, and N. Nystrom. Jif: Java Information Flow. Software release. `http://www.cs.cornell.edu/jif`, 2001.

[233] Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(4):410–442, 2000.

[234] Jasvir Nagra and Christian Collberg. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Pearson Education, 2009.

[235] A. Nanevski, A. Banerjee, and D. Garg. Verification of information flow and access control policies with dependent types. In *Security & Privacy*, pages 165–179. IEEE Computer Society, May 2011.

[236] Aleksandar Nanevski, Anindya Banerjee, and Deepak Garg. Dependent type theory for verification of information flow and access control policies. *ACM Trans. Program. Lang. Syst.*, 35(2):6, 2013.

[237] Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *TODS*, 35(3):21, 2010.

[238] Netscape. Using data tainting for security. http://www.aisystech.com/resources/advtopic.htm, 2006.

[239] J. Newsome and D. X. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *NDSS*, 2005.

[240] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of program analysis*. Springer, 1999.

[241] Hanne Riis Nielson and Flemming Nielson. Content dependent information flow control. *Journal of Logical and Algebraic Methods in Programming*, 87:6–32, 2017.

[242] Hanne Riis Nielson, Flemming Nielson, and Ximeng Li. Hoare logic for disjunctive information flow. In *Programming Languages with Applications to Biology and Security*, pages 47–65. 2015.

[243] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 541–555, 2013.

[244] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[245] Martin Odersky and Tiark Rompf. Unifying functional and object-oriented programming with scala. *Commun. ACM*, 57(4), 2014.

[246] Colin O'Halloran. A Calculus of Information Flow. In *ESORICS*, 1990.

[247] Susan Owicki and David Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6:319–340, 1976.

[248] `https://f-droid.org/repository/browse/?fdid=name.bagi.levente.pedometer`. Accessed: 2016-07-05.

[249] François Pottier and Vincent Simonet. Information flow inference for ML. In *POPL*, 2002.

[250] Leonor Prensa Nieto. *Verification of parallel programs with the Owicki-Gries and rely-guarantee methods in Isabelle/HOL*. PhD thesis, Technische Universität München, 2002.

[251] Leonor Prensa Nieto and Javier Esparza. Verifying single- and multi-mutator garbage collectors with Owicki/Gries in Isabelle/HOL. In *Mathematical Foundations of Computer Science (MFCS)*, volume 1893 of *LNCS*, pages 619–628, 2000.

[252] Willard Rafnsson and Andrei Sabelfeld. Secure multi-execution: Fine-grained, declassification-aware, and transparent. In *CSF*, 2013.

[253] Willard Rafnsson and Andrei Sabelfeld. Compositional information-flow security for interactive systems. In *Proc. IEEE CSF*, pages 277–292, 2014.

[254] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. A machine-learning approach for classifying and categorizing android sources and sinks. In *NDSS*, 2014.

[255] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 74:358–366, 1953.

[256] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, 2004.

[257] William K. Robertson and Giovanni Vigna. Static enforcement of web application integrity through strong typing. In *USENIX*, 2009.

[258] Franziska Roesner, Tadayoshi Kohno, Alexander Moshchuk, Bryan Parno, Helen J Wang, and Crispin Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In *Security & Privacy*, pages 224–238, 2012.

[259] A. Russo, A. Sabelfeld, and K. Li. Implicit flows in malicious and nonmalicious code. *Marktoberdorf Summer School (IOS Press)*, 2009.

[260] Alejandro Russo and Andrei Sabelfeld. Dynamic vs. static flow-sensitive security analysis. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 186–199. IEEE Computer Society, 2010.

[261] Joanna Rutkowska and Rafal Wojtczuk. Qubes OS architecture. Technical report, Invisible Things Lab, January 2010.

[262] P. Ryan. Mathematical models of computer security—tutorial lectures. In *FOSAD*. Springer, 2001.

[263] P. Y. A. Ryan and T. Peacock. Opacity - further insights on an information flow property. Technical Report CS-TR-958, University of Newcastle upon Tyne, 2006.

[264] A. Sabelfeld and D. Sands. A per model of secure information flow in sequential programs. *Higher Order and Symbolic Computation*, 14(1), March 2001.

[265] A. Sabelfeld and D. Sands. Declassification: Dimensions and principles. *JCS*, 17, 2009.

[266] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[267] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *JSAC*, 2003.

[268] Andrei Sabelfeld and Andrew C. Myers. A model for delimited information release. In *ISSS*, volume 3233 of *Lecture Notes in Computer Science*, pages 174–191. Springer, 2003.

[269] Andrei Sabelfeld and Alejandro Russo. From dynamic to static and back: Riding the roller coaster of information-flow control research. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 352–365. Springer, 2009.

[270] Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *CSFW*, pages 200–214. IEEE Computer Society, 2000.

[271] Andrei Sabelfeld and David Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.

[272] Pierangela Samarati. Recursive revoke. In *Encyclopedia of Cryptography and Security*, pages 1035–1037. Springer, 2011.

[273] Pierangela Samarati and Sabrina Capitani de Vimercati. Access Control: Policies, Models, and Mechanisms. *Springer LNCS*, 2171, 2001.

[274] Ravi Sandhu and Fang Chen. The multilevel relational (MLR) data model. *TISSEC*, 1(1), 1998.

[275] Prateek Saxena, Steve Hanna, Pongsin Poosankam, and Dawn Song. FLAX: Systematic discovery of client-side validation vulnerabilities in rich web applications. In *Network and Distributed System Security Symposium (NDSS 2010)*. The Internet Society, 2010.

[276] Thomas Schmitz, Dustin Rhodes, Thomas H. Austin, Kenneth Knowles, and Cormac Flanagan. Faceted dynamic information flow via control and data monads. In *POST*, 2016.

[277] D. Schoepe, M. Balliu, B. C. Pierce, and A. Sabelfeld. Explicit secrecy: A policy for taint tracking. In *EuroS&P*, 2016.

[278] Daniel Schoepe. Isabelle formalizations. https://schoepe.org/~daniel/phd/isabelle, 2018.

[279] Daniel Schoepe, Musard Balliu, Frank Piessens, and A. Sabelfeld. Let's face it: Faceted values for taint tracking. In *ESORICS*, 2016. To Appear.

[280] Daniel Schoepe, Daniel Hedin, and Andrei Sabelfeld. SeLINQ: tracking information across application-database boundaries. In *ICFP*, 2014.

[281] B. Scholz, C. Zhang, and C. Cifuentes. User-input dependence analysis via graph reachability. In *SCAM, 2008*, 2008.

[282] David A. Schultz and Barbara Liskov. IFDB: decentralized information flow control for databases. In *EuroSys*, 2013.

[283] E. J. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *S&P 2010*, 2010.

[284] Koushik Sen, Swaroop Kalasapur, Tasneem Brutch, and Simon Gibbs. Jalangi: A selective record-replay and dynamic analysis framework for JavaScript. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2013)*, pages 488–498. ACM, 2013.

[285] Koushik Sen, Darko Marinov, and Gul Agha. Cute: a concolic unit testing engine for c. In *ESEC/SIGSOFT FSE*, pages 263–272, 2005.

[286] Haichen Shen, Aruna Balasubramanian, Anthony LaMarca, and David Wetherall. Enhancing mobile apps to use sensor hubs without programmer effort. In *UbiComp*, 2015.

[287] Paritosh Shroff, Scott Smith, and Mark Thober. Dynamic dependency monitoring to secure information flow. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, CSF '07, pages 203–217, 2007.

[288] Vincent Simonet. The Flow Caml system. Software. http://cristal.inria.fr/~simonet/soft/flowcaml, 2003.

[289] Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. Discovering browser extensions via web accessible resources. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017*, pages 329–336, 2017.

[290] Geoffrey Smith and Dennis Volpano. Secure information flow in a multithreaded imperative language. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 355–364, 1998.

[291] SnoopWall. Flashlight Apps Threat Assessment Report. https://www.snoopwall.com/reports, 2014.

[292] D. X. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. BitBlaze: A new approach to computer security via binary analysis. In *ICISS*, 2008.

[293] M. Sridharan, S. Artzi, M. Pistoia, S. Guarnieri, O. Tripp, and R. Berg. F4F: taint analysis of framework-based web applications. In *OOPSLA*, 2011.

[294] Cristian-Alexandru Staicu and Michael Pradel. Freezing the web: A study of ReDoS vulnerabilities in JavaScript-based web servers. Technical Report TUD-CS-2017-0305, TU Darmstadt, 2017.

[295] Cristian-Alexandru Staicu, Michael Pradel, and Ben Livshits. Understanding and automatically preventing injection attacks on Node.js. In *NDSS*, 2018.

[296] CristianAlexandru Staicu and Michael Pradel. An empirical study of implicit information flow, 2015. Poster at PLDI. https://www.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_SOLA/Papers/poster-pldi2015-src.pdf.

[297] Deian Stefan, Pablo Buiras, Edward Z. Yang, Amit Levy, David Terei, Alejandro Russo, and David Mazières. Eliminating cache-based timing attacks with instruction-based scheduling. In *Proc. ESORICS*, pages 718–735, Sep 2013.

[298] Alley Stoughton, Andrew Johnson, Samuel Beller, Karishma Chadha, Dennis Chen, Kenneth Foner, and Michael Zhivich. You sank my battleship!: A case study in secure programming. 2014.

[299] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. In *ASPLOS*, 2004.

[300] D. Sutherland. A model of information. In *NCSC*, 1986.

[301] Nikhil Swamy, Juan Chen, and Ravi Chugh. Enforcing stateful authorization and information flow policies in Fine. In *Proc. European Symp. on Programming*, March 2010.

[302] Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. Secure distributed programming with value-dependent types. In *Proc. ACM International Conference on Functional Programming*, pages 266–278, 2011.

[303] Nikhil Swamy, Brian J. Corcoran, and Michael Hicks. Fable: A language for enforcing user-defined security policies. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 369–383. IEEE, 2008.

[304] Don Syme. Leveraging .NET Meta-programming Components from F#: Integrated Queries and Interoperable Heterogeneous Execution. In *ML*, 2006.

[305] Filippo Del Tedesco, Sebastian Hunt, and David Sands. A semantic hierarchy for erasure policies. In *ICISS*, 2011.

[306] Tachio Terauchi and Alexander Aiken. Secure information flow as a safety problem. In *Static Analysis, 12th International Symposium, SAS 2005, London, UK, September 7-9, 2005, Proceedings*, pages 352–367, 2005.

[307] Manolis Terrovitis. Privacy preservation in the dissemination of location data. *SIGKDD Explorations*, 2011.

[308] O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, and O. Weisman. Taj: effective taint analysis of web applications. In *PLDI*, 2009.

[309] Omer Tripp, Pietro Ferrara, and Marco Pistoia. Hybrid security analysis of web javascript code via dynamic partial evaluation. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 49–59, 2014.

[310] Omer Tripp, Marco Pistoia, Patrick Cousot, Radhia Cousot, and Salvatore Guarnieri. Andromeda: Accurate and scalable security analysis of web applications. In *FASE*, 2013.

[311] Neil Vachharajani, Matthew J. Bridges, Jonathan Chang, Ram Rangan, Guilherme Ottoni, Jason A. Blome, George A. Reis, Manish Vachharajani, and David I. August. RIFLE: An Architectural Framework for User-Centric Information-Flow Security. In *MICRO*, 2004.

[312] R. Vallée-Rai, P. Co, E. Gagnon, L. J. Hendren, P. Lam, and V. Sundaresan. Soot - a java bytecode optimization framework. In *Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative Research*. IBM, 1999.

[313] B. van Delft, S. Hunt, and D. Sands. Very static enforcement of dynamic policies. In *POST*, pages 32–52, 2015.

[314] Mathy Vanhoef, Willem De Groef, Dominique Devriese, Frank Piessens, and Tamara Rezk. Stateful declassification policies for event-driven programs. In *CSF*, 2014.

[315] V. N. Venkatakrishnan, Wei Xu, Daniel C. DuVarney, and R. Sekar. Provably correct runtime enforcement of non-interference properties. In *Proceedings of the 8th International Conference on Information and Communications Security*, ICICS'06, pages 332–351, 2006.

[316] Sabrina De Capitani di Vimercati and Giovanni Livraga. Sql access control model. In *Encyclopedia of Cryptography and Security*, pages 1248–1251. Springer, 2011.

[317] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Krügel, and Giovanni Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2007, San Diego, California, USA, 28th February - 2nd March 2007*, 2007.

[318] D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. *J. Computer Security*, 7(2,3):231–253, 1999.

[319] D. M. Volpano. Safety versus secrecy. In *SAS*, 1999.

[320] Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.

[321] Dennis M. Volpano and Geoffrey Smith. Eliminating covert flows with minimum typings. In *CSFW*, 1997.

[322] David von Oheimb. Information flow control revisited: Noninfluence = noninterference + nonleakage. In *ESORICS*, 2004.

[323] Qihua Wang, Ting Yu, Ninghui Li, Jorge Lobo, Elisa Bertino, Keith Irwin, and Ji-Won Byun. On the correctness criteria of fine-grained access control in relational databases. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 555–566, 2007.

[324] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *CCS*, 2014.

[325] Zachary Weinberg, Eric Y Chen, Pavithra Ramesh Jayaraman, and Collin Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 147–161. IEEE, 2011.

[326] J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, 1990.

[327] Yi-Chin Wu and Stéphane Lafortune. Comparative analysis of related notions of opacity in centralized and coordinated architectures. *DEDS*, 2013.

[328] Jean Yang, Travis Hance, Thomas H Austin, Armando Solar-Lezama, Cormac Flanagan, and Stephen Chong. Precise, dynamic information flow for database-backed applications. In *Proc. ACM SIGPLAN Conference on Programming language Design and Implementation*, volume 51, pages 631–647, 2016.

[329] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. A language for automatically enforcing privacy policies. In *POPL*, 2012.

[330] Aris Zakinthinos and E Stewart Lee. A general theory of security properties. In *Security & Privacy*, pages 94–102, 1997.

[331] Dante Zanarini, Mauro Jaskelioff, and Alejandro Russo. Precise enforcement of confidentiality for reactive systems. In *CSF*, 2013.

[332] Stephan Arthur Zdancewic. *Programming Languages for Information Security*. PhD thesis, Cornell University, Ithaca, NY, USA, 2002.

[333] Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazières. Securing distributed systems with information flow control. In *5th USENIX Symposium on Networked Systems Design & Implementation, NSDI 2008, April 16-18, 2008, San Francisco, CA, USA, Proceedings*, pages 293–308, 2008.

[334] Chenyi Zhang. Conditional information flow policies and unwinding relations. pages 227–241, 2011.

[335] Danfeng Zhang, Yao Wang, G. Edward Suh, and Andrew C. Myers. A hardware design language for timing-sensitive information-flow security. 2015.

[336] Lantian Zheng and Andrew C. Myers. Dynamic security labels and static information flow control. *International Journal of Information Security*, 6(2–3), March 2007.

[337] Lantian Zheng and Andrew C. Myers. Dynamic security labels and static information flow control. *Int. J. Inf. Sec.*, 2007.