RESEARCH CENTRE

Sophia Antipolis - Méditerranée

2021 ACTIVITY REPORT

Project-Team INDES

Secure Diffuse Programming

DOMAIN

Networks, Systems and Services, Distributed Computing

THEME

Distributed programming and Software engineering

Contents

Project-Team INDES

Creation of the Project-Team: 2010 July 01

Keywords

Computer sciences and digital sciences

A1.3. – Distributed Systems

A2. – Software

A2.1. – Programming Languages

A2.1.1. – Semantics of programming languages

A2.1.3. – Object-oriented programming

A2.1.4. – Functional programming

A2.1.7. – Distributed programming

A2.1.9. – Synchronous languages

A2.1.12. – Dynamic languages

A2.2.1. – Static analysis

A2.2.5. – Run-time systems

A2.2.9. – Security by compilation

A4.3.3. – Cryptographic protocols

A4.6. – Authentication

A4.7. – Access control

Other research topics and application domains

B6.3.1. – Web

B6.4. – Internet of things

B9.5.1. – Computer science

B9.10. – Privacy

1 Team members, visitors, external collaborators

Research Scientists

- Manuel Serrano [Team leader, Inria, Senior Researcher, HDR]
- Ilaria Castellani [Inria, Researcher]
- Tamara Rezk [Inria, Senior Researcher, HDR]

PhD Students

- Lesly Ann Daniel [CEA, until Sep 2021]
- Mohamad El Laz [Inria]
- Jayanth Krishnamurthy [Inria]
- Heloise Maurel [Inria]

Technical Staff

- Yoon Seok Ko [Inria, Engineer, until Oct 2021]

Administrative Assistant

- Nathalie Bellesso [Inria]

External Collaborators

- Gérard Berry [Collège de France, HDR]
- Marc Feeley [Université de Montréal - Canada]

2 Overall objectives

The goal of the Indes team is to study models for diffuse computing and develop languages for secure diffuse applications. Diffuse applications, of which Web 2.0 applications are a notable example, are the new applications emerging from the convergence of broad network accessibility, rich personal digital environment, and vast sources of information. Strong security guarantees are required for these applications, which intrinsically rely on sharing private information over networks of mutually distrustful nodes connected by unreliable media.

Diffuse computing requires an original combination of nearly all previous computing paradigms, ranging from classical sequential computing to parallel and concurrent computing in both their synchronous / reactive and asynchronous variants. It also benefits from the recent advances in mobile computing, since devices involved in diffuse applications are often mobile or portable.

The Indes team contributes to the whole chain of research on models and languages for diffuse computing, going from the study of foundational models and formal semantics to the design and implementation of new languages to be put to work on concrete applications. Emphasis is placed on correct-by-construction mechanisms to guarantee correct, efficient and secure implementation of high-level programs. The research is partly inspired by and built around Hop, the web programming model proposed by the former Mimosa team, which takes the web as its execution platform and targets interactive and multimedia applications.

3 Research program

3.1 Parallelism, concurrency, and distribution

Concurrency management is at the heart of diffuse programming. Since the execution platforms are highly heterogeneous, many different concurrency principles and models may be involved. Asynchronous concurrency is the basis of shared-memory process handling within multiprocessor or multicore computers, of direct or fifo-based message passing in distributed networks, and of fifo- or interrupt-based event handling in web-based human-machine interaction or sensor handling. Synchronous or quasi-synchronous concurrency is the basis of signal processing, of real-time control, and of safety-critical information acquisition and display. Interfacing existing devices based on these different concurrency principles within Hop or other diffuse programming languages will require better understanding of the underlying concurrency models and of the way they can nicely cooperate, a currently ill-resolved problem.

3.2 Web, functional, and reactive programming

We are studying new paradigms for programming Web applications that rely on multi-tier functional programming. We have created a Web programming environment named Hop. It relies on a single formalism for programming the server-side and the client-side of the applications as well as for configuring the execution engine.

Hop is a functional language based on the SCHEME programming language. That is, it is a strict functional language, fully polymorphic, supporting side effects, and dynamically type-checked. Hop is implemented as an extension of the BIGLOO Scheme compiler that we develop. In the past, we have extensively studied static analyses (type systems and inference, abstract interpretations, as well as classical compiler optimizations) to improve the efficiency of compilation in both space and time.

As a Hop DSL, we have created HipHop, a synchronous orchestration language for web and IoT applications. HipHop facilitates the design and programming of complex web/IoT applications by smoothly integrating three computation models and programming styles that have been historically developed in different communities and for different purposes: *i) Transformational programs* that simply compute output values from input values, with comparatively simple interaction with their environment; *ii)* asynchronous concurrent programs that perform interactions between their components or with their environment with uncontrollable timing, using typically network-based communication; and *iii)* synchronous reactive programs that react to external events in a conceptually instantaneous and deterministic way.

3.3 Security of diffuse programs

The main goal of our security research is to provide scalable and rigorous language-based techniques that can be integrated into multi-tier compilers to enforce the security of diffuse programs. Research on language-based security has been carried on before in former Inria teams. In particular previous research has focused on controlling information flow to ensure confidentiality.

Typical language-based solutions to these problems are founded on static analysis, logics, provable cryptography, and compilers that generate correct code by construction. Relying on the multi-tier programming language Hop that tames the complexity of writing and analysing secure diffuse applications, we are studying language-based solutions to prominent web security problems such as code injection and cross-site scripting, to name a few.

4 Application domains

4.1 Web

The Web is the natural application domain of the team. We are designing and implementing multitier languages for helping the development of Web applications. We are creating static and dynamic analyses for Web security. We are conducting empirical studies about privacy preservation on the Web.

4.2 Internet of Things

More recently, we have started focusing on *Internet of Things* (IoT) applications. They share many similarities with Web applications so most of the methodologies and expertises we have developed for the Web apply to IoT but the restricted hardware resources made available by many IoT devices demand new developments and new research explorations.

5 Highlights of the year

5.1 Awards

- Bertrand Petit and Manuel Serrano received the *Reviewers' Choice Award* [16] of the Programming Journal for their work on interactive music using reactive programming languages.
- Tamara Rezk and her coauthors outside Indes received a honorable mention from Intel: the Intel Security Award 2021 for her work done in 2020 on microarchitectural semantics [14].

6 New software and platforms

Let us describe new/updated software.

6.1 New software

6.1.1 Bigloo

Keyword: Compilers

Functional Description: Bigloo is a Scheme implementation devoted to one goal: enabling Scheme based programming style where C(++) is usually required. Bigloo attempts to make Scheme practical by offering features usually presented by traditional programming languages but not offered by Scheme and functional programming. Bigloo compiles Scheme modules. It delivers small and fast stand alone binary executables. Bigloo enables full connections between Scheme and C programs, between Scheme and Java programs.

Release Contributions: modification of the object system (language design and implementation), new APIs (alsa, flac, mpg123, avahi, csv parsing), new library functions (UDP support), new regular expressions support, new garbage collector (Boehm's collection 7.3alpha1).

URL: http://www-sop.inria.fr/teams/indes/fp/Bigloo/

Contact: Manuel Serrano

Participant: Manuel Serrano

6.1.2 Hop

Keywords: Programming language, Multimedia, Iot, Web 2.0, Functional programming

Scientific Description: The Hop programming environment consists in a web broker that intuitively combines in a single architecture a web server and a web proxy. The broker embeds a Hop interpreter for executing server-side code and a Hop client-side compiler for generating the code that will get executed by the client.

An important effort is devoted to providing Hop with a realistic and efficient implementation. The Hop implementation is validated against web applications that are used on a daily-basis. In particular, we have developed Hop applications for authoring and projecting slides, editing calendars, reading RSS streams, or managing blogs.

Functional Description: Multitier web programming language and runtime environment.

URL: http://hop.inria.fr

Contact: Manuel Serrano

Participant: Manuel Serrano

6.1.3 IFJS

Name: Infomation Flow monitor inlining for JavaScript

Keyword: Cybersecurity

Functional Description: The IFJS compiler is applied to JavaScript code. The compiler generates JavaScript code instrumented with checks to secure code. The compiler takes into account special features of JavaScript such as implicit type coercions and programs that actively try to bypass the inlined enforcement mechanisms. The compiler guarantees that third-party programs cannot (1) access the compiler internal state by randomizing the names of the resources through which it is accessed and (2) change the behaviour of native functions that are used by the enforcement mechanisms inlined in the compiled code.

URL: http://www-sop.inria.fr/indes/ifJS/

Contact: Tamara Rezk

6.1.4 Hiphop.js

Name: Hiphop.js

Keywords: Web 2.0, Synchronous Language, Programming language

Functional Description: HipHop.js is an Hop.js DLS for orchestrating web applications. HipHop.js helps programming and maintaining Web applications where the orchestration of asynchronous tasks is complex.

URL: http://hop-dev.inria.fr/hiphop

Contact: Manuel Serrano

6.1.5 Server-Side Protection against Third Party Web Tracking

Keywords: Privacy, Web Application, Web, Architecture, Security by design, Program rewriting techniques

Functional Description: We present a new web application architecture that allows web developers to gain control over certain types of third party content. In the traditional web application architecture, a web application developer has no control over third party content. This allows the exchange of tracking information between the browser and the third party content provider.

To prevent this, our solution is based on the automatic rewriting of the web application in such a way that the third party requests are redirected to a trusted third party server, called the Middle Party Server. It may be either controlled by a trusted party, or by a main site owner and automatically eliminates third-party tracking cookies and other technologies that may be exchanged by the browser and third party server

URL: http://www-sop.inria.fr/members/Doliere.Some/essos/

Contact: Francis Doliére Some

6.1.6 webstats

Name: Webstats

Keywords: Web Usage Mining, Statistic analysis, Security

Functional Description: The goal of this tool is to perform a large-scale monthly crawl of the top Alexa sites, collecting both inline scripts (written by web developers) and remote scripts, and establishing the popularity of remote scripts (such as Google Analytics and jQuery). With this data, we establish whether the collected scripts are actually written in a subset of JavaScript by analyzing the different constructs used in those scripts. Finally, we collect and analyze the HTTP headers of the different sites visited, and provide statistics about the usage of HTTPOnly and Secure cookies, and the Content Security Policy in top sites.

URL: https://webstats.inria.fr

Contact: Francis Doliére Some

6.1.7 Skini Node.js (ISS)

Name: Platform for creation and execution for audience participative music

Keywords: Music, Interaction, Web Application, Synchronous Language

Functional Description: Skini is a platform form designing et performing collaborative music. It is based on two musical concept: pattern and orchestration. The orchestration is design using HipHop.js.

Release Contributions: Can be use for performance and création.

Author: Bertrand Petit

Contact: Bertrand Petit

7 New results

We have pursued the development of Hop and our study on efficient JavaScript implementations as well as our development of analyses for distributed language sessions and security.

7.1 Web Reactive Programming

Participants: Jayanth Krishnamurthy, Manuel Serrano.

Computer systems that react continuously to their environment at a rate set by the environment form a class of the so-called *reactive systems*. They differ from classical computing systems which takes the input at the start of execution and produce output before terminating. Furthermore, they also differ from traditional interactive systems like operating systems which endlessly interact with their environment at their own speed (in contrast to the speed determined by the environment). A reactive system can be perceived as a black box that perpetually receives some input events as external stimuli and reacts to them by producing some output events as their behavior. This output may successively affect the production of later stimuli by the environment.

HipHop, which is the language used in this study is based on Esterel's semantics. It is a synchronous reactive DSL for JavaScript, built on top of Hop.js for applying the Esterel programming model to the world of web and IoT where programs are sent over networks. HipHop can be used to develop complex web application interfaces and IoT controllers, which are dynamic in nature. HipHop blends Esterel's synchrony with JavaScript's asynchrony, simplifying the cooperation between synchronous and asynchronous activities that are typical in these application domains. HipHop differs from Esterel in having

its own syntax and programming model adapted to the web. For example, HipHop supports partial reconfiguration of programs between two synchronous reactions, while maintaining consistency of the control state.

The expressiveness and the flexibility of Esterel dialects come with a downside: the debugging, and more precisely the error reporting is difficult because errors detected by the runtime system are loosely connected with locations in the program source code. This is a major difficulty, especially for programmers not deeply accustomed with the programming model. Improving the error messages the compiler and the runtime system report is then a major issue and is the subject of ongoing researches in the team. This year we have developed and implemented a technique that isolates the fragments of the program that are responsible for an error when it occurs. The technique we propose applies to the compilation technique HipHop uses to transform a source program into an equivalent electric circuit using techniques developed for the Esterel programming language. The improved error messages are built by isolating parts in the generated circuit - minimizing the size of causality error cycles using an iterative process.

The method of causality error analysis and debugging proceeds by building on classical graph algorithms, which are applied to the graph of nets composing the circuit generated by the HipHop compiler. This enables programmers to narrow down to smaller error positions in source code. We have shown the results and advantages of application of our debugging approach in a real life project developed using HipHop. This work has been presented at the *Principles and Practice of Declarative Programming* conference [5].

7.2 Implementation of Dynamic Languages

Participants: Manuel Serrano.

Dynamic languages are particularly difficult to implement efficiently because most of their expressions have all sorts of different meanings that involve all sorts of different executions that are not distinguished by any syntactic or type annotation. For instance considering JavaScript, "`obj.prop`" might *i)* fetch property `prop` from `obj`, *ii)* scan the linked list of `obj`'s prototype chain and fetch `prop` from another object, *iii)* call a user defined function if `prop` is an accessor, *iv)* allocate a fresh object if `obj` is a primitive value, or *v)* evaluate yet another user function if `obj` is a proxy object. Checking all the possible interpretations and executing the appropriate one literally, that is treating the language specification as an algorithm, delivers unacceptably slow performance. All fast implementations use alternative strategies. Amongst all the possible interpretations, they favor the one that corresponds to the most frequent situation, for which they elaborate a faster execution plan, and, as importantly, for which they elaborate a fast guard that ensures the preservation of the language semantics. Typically, that is what *inline caches* and *hidden classes* achieve. Using a single test, the comparison of the object's hidden class with the inline cache, we know if the property is to be read directly from the object and, if so, at which offset. The common intuition is that only dynamic compilers, *a.k.a.*, JIT compilers, can handle dynamic languages efficiently because this heuristic-based strategy requires having the program *and* the data on hand in order to generate efficient code. We view this position as too extreme, as it is oblivious to other characteristics of static compilers (AoT) that might make them competitive.

- AoT compilers can allocate conceptually infinite resources for analyzing and optimizing the program because they run before execution. This opens opportunities to conceive and deploy new optimizations that are out of reach of JIT compilers for which compilation time and compilation resource consumption matter.
- AoT compilers are efficient even for brief executions while JIT compilers need the execution to last sufficiently long to benefit from gathered profiled data. This should give AoT compilers an advantage for executing programs such as shell commands or cloud computing microservices.
- New techniques have been proposed to adapt JIT-style heuristics-based approach to AoT compilation. These studies have shown that if indeed a JavaScript expression involves many different

interpretations, typically only one is used over and over again and guessing it before the execution is not too difficult.

As few are committed to developing optimizing AoT JavaScript compilers, we rely too heavily on our intuition to answer the question whether AoT compilers can deliver performance comparable to JIT compilers. To provide the elements of a proper scientific comparison, we have built Hopc, an AoT compiler for JavaScript. We have compared its performance with those of production JIT compilers and we have shown that on many new tests, its performance is close to those of JIT compilers [1]. We read this as a strong indication that an AoT compiler that optimizes the whole core language and the whole set of libraries could compete with the fastest JIT compilers. We intend to pursue this exploration in the coming years.

7.3 Session Types

Participants: Ilaria Castellani.

Session types describe communication protocols involving two or more participants by specifying the sequence of exchanged messages and their functionality (sender, receiver and type of carried data). They may be viewed as the analogue, for concurrency and distribution, of data types for sequential computation. Originally conceived as a static analysis technique for an enhanced version of the π-calculus, session types have been subsequently embedded into a range of functional, concurrent, and object-oriented programming languages.

The aim of session types is to ensure safety properties for sessions, such as the *absence of communication errors* (no type mismatch in exchanged data) and *deadlock-freedom* (no standstill until all participants are terminated). Multiparty session types often target also the liveness property of *progress* or *lock-freedom* (no participant waits forever), which is stronger than deadlock-freedom.

While binary sessions can be described by a single session type, multiparty sessions require two kinds of types: a *global type* that describes the whole session protocol, and *local types* that describe the individual contributions of the participants to the protocol. The key requirement to achieve safety properties such as deadlock-freedom is that the local types of the processes implementing the participants be obtained as projections from the same global type. To ensure progress, global types must satisfy additional well-formedness requirements.

What makes session types particularly attractive is that they offer several advantages at once: 1) static safety guarantees, 2) automatic check of protocol implementation correctness, based on local types, and 3) a strong connection with linear logics and with concurrency models such as communicating automata, graphical choreographies and message-sequence charts.

During the past year we have further investigated the relationship between multiparty session types and concurrency models, focussing on Event Structures [17], a canonical model for concurrent computation. As most of our previous work on this subject, this research has been pursued in collaboration with colleagues from the Universities of Eastern Piedmont and Turin.

Event Structure Semantics for Synchronous and Asynchronous Multiparty Sessions In the two papers [10] and [11], we explored the relationship between multiparty session calculi and Event Structures (ESs), a well-known concurrency model introduced in the early 80's [18, 15].

In the first paper [10], we considered a core multiparty session calculus with *synchronous communication*, where sessions are described as networks of sequential processes (each process implementing a participant), equipped with standard global types. We proposed an interpretation of networks as *Flow Event Structures* (FESs) [13], a subclass of Winskel's Stable Event Structures [18], as well as an interpretation of global types as *Prime Event Structures* (PESs) [15], the simplest class of ESs. Since global types are sequential specifications, which are not able to explicitly represent the concurrency among communications, the events of the associated PES need to be defined as equivalence classes of communication sequences up to *permutation equivalence*. We showed that when a network is typable with a global type,

the FES semantics of the former is equivalent, in a precise technical sense, to the PES semantics of its type.

In the second paper [11], we undertook a similar endeavour for *asynchronous communication*. This led us to devise a new notion of global type for asynchronous sessions. The type system for asynchronous sessions is expected to be more permissive than the one for synchronous sessions. For instance, consider a session with two participants each of which wishes to first send a message and then receive a message. This session is stuck if communication is synchronous but not if communication is asynchronous. Hence it should be typable in the latter case but not in the former.

We started by considering a core session calculus as in the synchronous case, where networks are now endowed with a queue, and they act on this queue by performing outputs or inputs: an output stores a message in the queue, while an input fetches a message from the queue. Then, the idea for our *asynchronous global types* is quite simple: to split communications in the type into outputs and inputs, and to equip the type with a queue, thus mimicking very closely the behaviour of asynchronous networks. The well-formedness conditions for global types must now take into account also the queue. Essentially, this amounts to requiring that each input appearing in the type be justified by a preceding output in the type or by a message in the queue, and vice versa.

The contribution of [11] is twofold: 1) We propose an original type system for asynchronous multiparty sessions, which accounts for asynchronous communication more directly than existing approaches, while remaining decidable; 2) We present an Event Structure semantics for asynchronous sessions and asynchronous global types, and we show that these two semantics agree.

Both these papers have been submitted for journal publication.

7.4 Micro-architectural attacks

Participants: Lesly-Ann Daniel, Tamara Rezk.

Previously, in 2020, we had developed an analyzer for constant-time called Binsec/Rel. Binsec/Rel analyses timing-leaks attacks. These attacks can be captured via a security property called constant-time, which states that the execution time of an application does not depend on the dynamic path of the execution. Our analyzer works at binary-level and is based on symbolic execution with dedicated optimizations for constant-time analysis. In particular, we complement relational symbolic execution with a new on-the-fly simplification to maximize sharing in the memory and formally prove that our analysis is correct for bug-finding and bounded-verification. In 2021, we extended this analyzer to handle microarchitectural attacks. The new analyzer is called Binsec/Haunted [3]. We first modeled the semantics of hardware with microarchitectural features to model timing-side channels and attacks such as Spectre that can be used *e.g.*, in the cloud, to learn all kind of confidential information from the cloud's customers. Our obtained hardware semantics supports out-of-order and speculative execution by modeling reorder buffers and transient instructions, respectively. It assumes that attackers have complete control over microarchitectural features (*e.g.*, the branch target predictor), and uses adversarial execution directives to model the adversarys control over predictors. The Binsec/Haunted analyzer was based on this semantics and scales to detect Spectre-PHT vulnerabilities in binaries of cryptographic libraries. It also works for Spectre-STL attacks and helped to uncover inconsistencies between different Spectre defenses. The analyzers have helped to disclose that popular compilers cannot be trusted to preserve constant-time and that popular counter-measures for Spectre vulnerabilities may also introduce other variants of the vulnerability. Binsec/Rel and Binsec/Haunted constitute the contributions in the PhD thesis of Lesly-Ann Daniel, who defended on November 12th, 2021. We have also worked in adding phases to the Jazmin compiler in order to obtain code certified as free of Spectre PHT and STL attacks. Our works on microarchitectural attacks have been presented in major security conferences [3, 2].

7.5 JavaScript security

Participants: Yoonseok Ko, Heloise Maurel, Tamara Rezk, Manuel Serrano.

Modern client-side web applications often include external third-party code, namely gadgets such as advertisement banners.

Previously to 2021, we developed a compiler, called Mashic, that takes a client-side web application as input to transform it in such a way that gadgets are included with iframes. The guarantees of the compiler are: if the gadget is not malicious, then the functionality of the compiled code is the same as the original one. If the mashup is malicious, all the attacks are neutralized and left with no effect on trusted code. During the current period, with the intention of obtaining a compiler applicable to IoT applications, we have generalized the Mashic compiler to be independent from the browsers and, in particular, of any browser built-in isolation mechanisms. We called the generalization SecureJS [4], and it can used for JavaScript running in IoT devices. SecureJS passes the test suites of the ECMAScript5 specification in diverse JavaScript engines such as V8, Hop, and JerryScript. As a continuation of this work, we have studied more flexible JavaScript security policies, in particular a security policy that will allow us to express sharing of JavaScript objects via declassification. This kind of policies can be implemented for example by membrane patterns from the object capability model and could be used to prove formally the security property that the secure subset of EcmaScript provides. We also studied JavaScript and PHP security by means of analyzers to detect XSS vulnerabilities using neural networks [6]. We compare two different code representations based on Natural Language Processing (NLP) and Programming Language Processing (PLP) and experiment with models based on different neural network architectures for static analysis detection of vulnerabilities in PHP and Node.js. We train and evaluate the models using synthetic databases. Using the generated PHP and Node.js databases, we compare our results with a well-known static analyzer for PHP code, ProgPilot, and a known scanner for Node.js, AppScan static mode. Our analyzers using neural networks overcome the results of existing tools in all cases but are limited to only a reduced number of lines of code.

8 Partnerships and cooperations

8.1 International initiatives

8.1.1 Inria associate team not involved in an IIL or an international program

HipHopSec

Participants: Tamara Rezk, Manuel Serrano.

Title: *Secure Reactive IoT Programming*

Duration: 2021 - 2022

Coordinator: Manuel Serrano

Partners: Northwestern University (Chicago, United States).

Inria contact: Manuel Serrano

Summary: Nowadays most applications are distributed, that is, they run on several computers: a mobile device for the graphical user interface a gateway for storing data in a local area; a remote server of a large cloud platform for resource demanding computing; an object connected to Internet in the IoT (Internet of Things); etc.

For many different reasons, this makes programming much more difficult than it was when only a single computer was involved:

- Applications are composed of extensive lists of diverse components, each coming with their own specification and imposing its own constraints on application development.
- Due to the distributed nature of the applications, developers have to implement appropriate communication protocols, which is difficult to do correctly and securely.
- Communicating applications need to resort to parallelism to handle requests from their clients with acceptable latency. No matter whether it is multi-threading (as in Java) or asynchronous programming (as in JavaScript/Node.js), this style of programming is notoriously difficult and error-prone.

The Indes, Northwestern, and College de France teams are studying programming languages and have each created complementary solutions that address the aforementioned problems. Combined together, they could lead to a robust and secure execution environment for the web and IoT programming. Indes will bring its expertise in secure web programming, College de France its expertise in synchronous reactive programming, Northwestern its expertise secure execution environments and run-time validation of security properties of program executions. Finally Northwestern will contribute with its expertise in medical descriptions, which will be the main application domain of the secure execution environment the participants aim to develop.

The main objective of the collaboration is the development of a robust and secure integrated programming environment for reactive applications suitable for web and IoT applications. The programming of medical prescriptions will be our favored application domain. We will base our work on three pillars: Hop.js, the contract system designed for the Racket language, and HipHop.js, a domain specific language for reactive programming within Hop.js.

- HipHop.js has currently minimal integration with Hop.js and a rudimentary programming environment. We will continue the development of HipHop.js with the goal of turning it into a a usable and reliable platform.
- The formal semantics of HipHop.js is based on rewriting logics, automata theory and Boolean equations. Thus, HipHop.js programs can be verified using existing techniques based on the satisfiability of logic formulas. Such techniques have been widely used for synchronous reactive programs, but never before in the more dynamic world of web or medical applications.
- Supporting medical prescriptions as programs requires not only a language with special syntactic abstractions to match the notations of the medical domain, but also a fundamentally new way to think about prescription vs. computer programs. For example, medical personnel often modifies prescriptions in the middle of a treatment. In linguistic terms this requires that the programming language in use supports the ability to pause a program while it is running, modify its code, and restart it from the point of the pause but with the modified version of the code, this in a guaranteed consistent way. We hope to build such a programming language, with a semantics inspired by synchronous-reactive programming in the style of HipHop.js but tailored to the medical domain.
- Contracts state precise properties of the interfaces of components and validate them at run time. Over the last fifteen years, Racket developers, including those dealing with the language itself, have used contracts extensively to validate properties that range from simple type-like constraints to partial functional correctness and even security. Our goal is to design and implement a contract system for Hop/HipHop.js that is as expressive as that of Racket. Hop/HipHop.js is based on Javascript, a different linguistic setting than that of Racket; however, existing work on Javascript proxies and macros has resulted in encouraging preliminary results on contracts for higher-order functions and objects in Javascript. We aim at lifting and extending these result to Hop/HipHop.js. Given an expressive contract system for Hop/HipHop.js, we will investigate: (i) how to state and enforce security policies for Hop/HipHop.js applications with contracts; and (ii) how different compilation and implementation techniques can alleviate existing performance issues of applications, a current weakness that impedes the widespread adoption of contracts.
- Improving the quality of the code requires support from testing. S. You (working with C. Dimoulas and R. Findler) is working on improving automated testing techniques. So far he has discovered a

new theoretical result showing how to use concolic testing for higher-order functions. This result may have applications for testing in JavaScript and we are hopeful that we can leverage it to Hop.js.

8.2 European initiatives

8.2.1 FP7 & H2020 projects

SPARTA

Participants: Tamara Rezk, Manuel Serrano.

Title: Strategic Programs for Advanced Research and Technology in Europe

Duration: *(2019) - (2022)*

Coordinator: *(CEA)*

Partners:

- CENTRE D'EXCELLENCE EN TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION (Belgium)
- CESNET ZAJMOVE SDRUZENI PRAVNICKYCH OSOB (Czech Republic)
- COMMISSARIAT A L ENERGIE ATOMIQUE ET AUX ENERGIES ALTERNATIVES (France)
- CONSIGLIO NAZIONALE DELLE RICERCHE (Italy)
- CONSORZIO INTERUNIVERSITARIO NAZIONALE PER L'INFORMATICA (Italy)
- CONSORZIO NAZIONALE INTERUNIVERSITARIO PER LE TELECOMUNICAZIONI (Italy)
- CZ.NIC, ZSPO (Czech Republic)
- DIREZIONE GENERALE PER LE TECNOLOGIE DELLE COMUNICAZIONI E LA SICUREZZA INFORMATICA - ISTITUTO SUPERIORE DELLE COMUNICAZIONI E DELLE TECNOLOGIE DELL'INFORMAZIONE (Italy)
- FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V. (Germany)
- FUNDACIO EURECAT (Spain)
- FUNDACION CENTRO DE TECNOLOGIAS DE INTERACCION VISUAL Y COMUNICACIONES VICOMTECH (Spain)
- FUNDACION TECNALIA RESEARCH & INNOVATION (Spain)
- GENEROLO JONO ZEMAICIO LIETUVOS KARO AKADEMIJA (Lithuania)
- INDRA SISTEMAS SA (Spain)
- INOV INESC INOVACAO - INSTITUTO DE NOVAS TECNOLOGIAS (Portugal)
- INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON (France)
- INSTITUTO SUPERIOR TECNICO (Portugal)
- ITTI SP ZOO (Poland)
- JOANNEUM RESEARCH FORSCHUNGSGESELLSCHAFT MBH (Austria)
- KAUNO TECHNOLOGIJOS UNIVERSITETAS (Lithuania)
- KENTRO MELETON ASFALEIAS (Greece)
- LEONARDO - SOCIETA PER AZIONI (Italy)
- LIETUVOS KIBERNETINIU NUSIKALTIMU KOMPETENCIJU IR TYRIMU CENTRAS (Lithuania)

- LUXEMBOURG INSTITUTE OF SCIENCE AND TECHNOLOGY (Luxembourg)
- MYKOLO ROMERIO UNIVERSITETAS (Lithuania)
- NATIONAL CENTER FOR SCIENTIFIC RESEARCH "DEMOKRITOS" (Greece)
- NAUKOWA I AKADEMICKA SIEC KOMPUTEROWA - PANSTWOWY INSTYTUT BADAWCZY (Poland)
- SECRETARIAT GENERAL DE LA DEFENSE ET DE LA SECURITE NATIONALE (France)
- STOWARZYSZENIE POLSKA PLATFORMA BEZPIECZENSTWA WEWNETRZNEGO (Poland)
- TARTU ULIKOOL (Estonia)
- TECHNIKON FORSCHUNGS- UND PLANUNGSGESELLSCHAFT MBH (Austria)
- TECHNISCHE UNIVERSITAET MUENCHEN (Germany)
- THALES SIX GTS FRANCE SAS (France)
- UNIVERSITAT KONSTANZ (Germany)
- UNIVERSITE DE NAMUR ASBL (Belgium)
- UNIVERSITE DU LUXEMBOURG (Luxembourg)
- VYSOKE UCENI TECHNICKE V BRNE (Czech Republic)

Inria contact: Tamara Rezk and Manuel Serrano

Summary: SPARTA establishes a strategic research and innovation roadmap to stimulate the development and deployment of key technologies in cybersecurity and to retain digital sovereignty and autonomy of the European industries.

SPARTA Roadmap serves as common ground for the alignment of research, education and certification priorities of the European Cybersecurity Competence Network.

8.3 National initiatives

8.3.1 ANR CISC

Participants: Ilaria Castellani, Tamara Rezk, Manuel Serrano.

The CISC project (Certified IoT Secure Compilation) is funded by the ANR for 42 months, starting in April 2018. The goal of the CISC project is to provide strong security and privacy guarantees for IoT applications by means of a language to orchestrate IoT applicatoins from the microcontroller to the cloud. Tamara Rezk coordinates this project, and Manuel Serrano, Ilaria Castellani and Nataliia Bielova participate in the project. The partners of this project are Inria teams Celtique, Indes and Privatics, and Collège de France.

9 Dissemination

Participants: Ilaria Castellani, Tamara Rezk, Manuel Serrano.

9.1 Promoting scientific activities

Tamara Rezk gave a talk for undergraduates at University of Córdoba in May 2021 to promote scientific careers.

9.1.1 Scientific events: organisation

Member of the organizing committees Tamara Rezk co-organized PLMW at PLDI'21.

Member of the conference program committees

- Manuel Serrano participated in the program committee of the 20th International Conference on Generative Programming: Concepts & Experiences.
- Tamara Rezk participated in the program committees of MADWeb'21, PriSC@POPL'21, FM'21, ACM CCS'21 and PLDI'21.

9.1.2 Journal

Member of the editorial boards

- Manuel Serrano is member of the Programming Journal Steering Committee.

9.1.3 Invited talks

- Ilaria Castellani gave an invited talk at the TRENDS 2021 Workshop, August 2021.
- Tamara Rezk gave an invited talk at GDR Sécurité, July 2021.

9.1.4 Leadership within the scientific community

- Tamara Rezk is a steering Committee member of Principles of Secure Compilation Workshop (PriSC) 2019-2022
- Tamara Rezk is a steering Committee member of Foundation of Computer Security Workshop (FCS) 2021-2025
- Tamara Rezk was leader of the Panel on "Software Side-Channel Attacks" at High-Assurance Crypto Software Workshop, May 2021

9.1.5 Research administration

- Tamara Rezk is member of the Bureau de CP since March 2021.
- Manuel Serrano is vice-head of the Inria Evaluation Committee. As such he co-organizes all the grants, promotions juries and the juries of the national recruiting campaigns. He also co-organizes all the team evaluation seminars.

9.2 Teaching - Supervision - Juries

9.2.1 Teaching

- Tamara Rezk taught two courses (master level) at Université Côte d'Azur: Web security (28 ETD) and Cryptographic proofs (28 ETD).
- Manuel Serrano gave a course on the implementation of JavaScript at the *Programming Language Implementation Summer School*.

9.2.2 Supervision

- PhD in progress: Jayanth Krishnamurthy, Secure Reactive Web Programming, 12/09/2018, Manuel Serrano.
- PhD in progress: Heloise Maurel, Machine Learning and Security Analysis, 01/10/2018, Tamara Rezk.
- PhD in progress: Mohamad El Laz, Cryptography, 01/12/2017, Tamara Rezk (co-supervision).
- PhD in progress: Ignacio Tiraboschi, Symbolic Analysis for IoT Security Analysis, 01/10/2020, Tamara Rezk (co-supervision).
- PhD in progress: Adam Khayam, Program Semantics, 01/07/2019, Tamara Rezk (co-supervision).
- PhD completed: Lesly-Ann Daniel, Symbolic Binary-Level Code Analysis for Security, 01/10/2018, Tamara Rezk (co-supervised).

9.2.3 Juries

Ilaria Castellani participated in the following PhD jury:

- Phd Jury Member (Reviewer): Eva Graversen (supervisors: Iain Phillips and Nobuko Yoshida), Imperial College London, March 2021.

Tamara Rezk participated in the following PhD juries:

- Phd Jury Member (Rapporteur) : Itsaka Rakotonirina (supervisor : Steve Kremer), Université de Lorraine 2021
- Phd Jury : Maximilian Algehed (supervisor : Mary Sheeran), Chalmers 2021
- Phd Jury (Rapporteur) : Rémy Hutin (supervisors : Sandrine Blazy and David Pichardie), ENS Rennes 2021

10 Scientific production

10.1 Publications of the year

International journals

[1] M. Serrano. 'Of JavaScript AOT compilation performance'. In: *Proceedings of the ACM on Programming Languages* (Aug. 2021). URL: https://hal.inria.fr/hal-03351860.

International peer-reviewed conferences

[2] G. Barthe, S. Cauligi, B. Grégoire, A. Koutsos, K. Liao, T. Oliveira, S. Priya, T. Rezk and P. Schwabe. 'High-Assurance Cryptography in the Spectre Era'. In: IEEE Symposium of Security and Privacy (S&P'21). Virtual, France, 24th May 2021. URL: https://hal.inria.fr/hal-03352062.

[3] L.-A. Daniel, S. Bardin and T. Rezk. 'Hunting the Haunter -Efficient Relational Symbolic Execution for Spectre with Haunted RelSE'. In: NDSS. Virtual, France, 2021. DOI: 10.14722/ndss.2021.24286. URL: https://hal.inria.fr/hal-03363263.

[4] Y. Ko, T. Rezk and M. Serrano. 'SecureJS Compiler: Portable Memory Isolation in JavaScript'. In: SAC 2021 - 36th ACM/SIGAPP Symposium On Applied Computing. Gwangju / Virtual, South Korea, 22nd Mar. 2021. URL: https://hal.inria.fr/hal-03090348.

[5] J. Krishnamurthy and M. Serrano. 'Causality Error Tracing in HipHop'. In: Principles and Practice of Declarative Programming. Tallinn, Estonia, 6th Sept. 2021. URL: https://hal.inria.fr/hal-03351887.

[6] H. Maurel, S. Vidal and T. Rezk. 'Statically Identifying XSS using Deep Learning'. In: SECRYPT 2021 - 18th International Conference on Security and Cryptography. Virtual, France, 6th July 2021. URL: https://hal.inria.fr/hal-03273564.

[7] B. Petit and M. Serrano. 'Generative Music Using Reactive Programming'. In: *Proceedings of the International Computer Music Conference 2021*. International Computer Music Conférence. Santiago, Chile, 21st July 2021. URL: https://hal.archives-ouvertes.fr/hal-03105666.

Reports & preprints

[8] A. Canteaut, M. A. Fernández, L. Maranget, S. Perin, M. Ricchiuto, M. Serrano and E. Thomé. *Évaluation des Logiciels*. Inria, 14th Jan. 2021. URL: https://hal.inria.fr/hal-03110723.

[9] A. Canteaut, M. A. Fernández, L. Maranget, S. Perin, M. Ricchiuto, M. Serrano and E. Thomé. *Software Evaluation*. Inria, 14th Jan. 2021. URL: https://hal.inria.fr/hal-03110728.

[10] I. Castellani, M. Dezani-Ciancaglini and P. Giannini. *Event structure semantics for multiparty sessions*. 3rd Feb. 2022. URL: https://hal.inria.fr/hal-03554668.

[11] I. Castellani, M. Dezani-Ciancaglini and P. Giannini. *Global types and event structure semantics for asynchronous multiparty sessions*. 1st Feb. 2021. URL: https://hal.inria.fr/hal-03126627.

[12] M. El Laz, A. Hevia and R. Tamara. *Tracking Information Flow by Mapping Broadcast Encryption Subgroups to Security Lattices*. Inria, 19th Jan. 2022. URL: https://hal.inria.fr/hal-03537962.

10.2 Cited publications

[13] G. Boudol and I. Castellani. 'Permutation of transitions: an event structure semantics for CCS and SCCS'. In: *REX: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*. Ed. by J. W. de Bakker, W. P. de Roever and G. Rozenberg. Vol. 354. LNCS. Springer, 1988, pp. 411–427.

[14] S. Cauligi, C. Disselkoen, K. V. Gleissenthall, D. Tullsen, D. Stefan, T. Rezk and G. Barthe. 'Constant-Time Foundations for the New Spectre Era'. In: *2020 Programming Language Design and Implementation (PLDI'20)*. PLDI '20. London, United Kingdom, 2020. URL: https://hal.inria.fr/hal-03141383.

[15] M. Nielsen, G. Plotkin and G. Winskel. 'Petri Nets, Event Structures and Domains, Part I'. In: *Theoretical Computer Science* 13.1 (1981), pp. 85–108.

[16] B. Petit and M. Serrano. 'Skini:Reactive Programming for Interactive Structured Music'. In: *The Art, Science, and Engineering of Programming* (June 2020). URL: https://hal.archives-ouvertes.fr/hal-03105643.

[17] G. Winskel. 'An introduction to event structures'. In: *REX: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*. Ed. by J. W. de Bakker, W. P. de Roever and G. Rozenberg. Vol. 354. LNCS. Heidelberg: Springer, 1988, pp. 364–397.

[18] G. Winskel. 'Events in Computation'. PhD thesis. University of Edinburgh, 1980.