Activity Report 2011

Project-Team MOSCOVA

Mobility, security, concurrence, verification and analysis

RESEARCH CENTER
Paris - Rocquencourt

THEME
Programs, Verification and Proofs

Table of contents

Project-Team MOSCOVA

Keywords: Programming Languages, Security, Formal Methods, Cryptography, Processors, Distributed System

Beginning of the Team: 01/01/2000.

1. Members

Research Scientists

Jean-Jacques Lévy [Team leader, Senior Researcher (DR) Inria]
Luc Maranget [Research Associate (CR) Inria]
Karthikeyan Bhargavan [Research Associate (CR) Inria]
James Leifer [Research Associate (CR) Inria]
Francesco Zappa Nardelli [Research Associate (CR) Inria]

PhD Students

Nataliya Guts [INRIA-MSR grant, Paris 7, until 15/1/2011]
Jérémy Planul [ENS-Lyon, Ecole Polytechnique]

Post-Doctoral Fellow

Alfredo Pironti [Crysp]

Administrative Assistant

Stéphanie Aubin [Assistant (TR) Inria]

2. Overall Objectives

2.1. Introduction

The research in Moscova which was traditionaly centered around the theory and practice of concurrent programming in the context of distributed and mobile systems, is now retargeted around two main themes: weak memory models and secure distributed computations. Our ambitious long-term goal is still to program safely concurrent applications on top of new multi-core architectures and to program secure global computations on top of wide-area networks.

In in the past years, we designed several concurrent programming languages (Jocaml, Acute) together with tools to study their semantics (OTT). Our languages are not as used as the known Java or $C\#$ which allow downloading of programs, but our languages also allow migrations of active programs. Moreover the join primitives of Jocaml have been implemented inside polyphonic $C\#$, in $F\#$ and Visual Basic. These studies and implementations demonstrated that there is still a need for a deep understanding of the underlying hardware and for an intensive use of programming primitives for security.

On the concurrency side, our implementations relied on locks and sequential consistency. This means that the semantics of our concurrent languages were defined in terms of interleaving of sequential instruction steps in each thread of programs. However this is no longer correct with modern architectures which allow delayed write operations in memory and permutation of reads and writes not related to the same memory location (TSO, PSO, RMO architectures). Therefore the semantics of concurrent languages need be revisited to take into account these new features. In this area, we have pursued our productive cooperation with U. of Cambridge (Peter Sewell) and with J. Alglave (now at Oxford).

For security, we already looked at programming primitives to define contracts for multiparty secure sessions or for logs auditing which could resist to malicious participants in strongly typed programming languages (Ocaml or $F\#$). Thus the use of these high-level primitives inside programming languages lighten the burden of close inspection for complex security protocols. All the effort is now carried by the correctness proof of the compiling phase from the source text of programs to low-level exchanges of messages using cryptographic operations. These correctness proofs are incredibly complex, and therefore one needs tools to help them. This is a great part of the work done in 2011 in our project-team. We developed several versions of the F7 language on top of $F\#$, a Caml-like language with types annotated by logical formulas. We used F7 to develop the work on secure audits of logs and to prove the correctness of implementations of SSL/TLS.

On the software side, we pursue the maintenance for the development of Jocaml with additional constructs for object-oriented programming, and for Hevea (a fast LaTeX to HTML translator). We also continue the development of OTT – one tool for the working semanticist – which has a growing set of users. More innovative are the softwares developed along the two main themes of our present research: the *Memevents-Litmus-Diy-Dont* suite of tools for efficient analyses of weak memory models, the F7 typechecker, and the secure multi-party sessions packages.

In 2011, the ERC Starting Grant (CRYSP) of Karthikeyan Bhargavan started with visit of Sergio Maffeis (6 months) and arrival of Alfredo Pironti (as postdoc). Nataliya Guts defended her PhD on 11 January 2011 and then started a postdoc at U. of Maryland with Mike Hicks. Jérémy Planul (ENS-Lyon and MPRI) will defend his PhD on 8 February 2012.

Since August 2006, J.-J. Lévy is also director of the Microsoft Research-INRIA Joint Centre, in Orsay. K. Bhargavan, N. Guts, J. Leifer, J. Planul, A. Pironti and F. Zappa Nardelli are also active in this centre, as members of the *Secure Distributed Computations and their Proofs*, headed by C. Fournet (Many members of the Joint Centre are former members of project-team Moscova).

Finally, we are in a phase of remodelling our project-team, since Moscova is 12-year old and J.-J. Lévy will retire on February 2012. A proposal for a new project (Prosecco) about Security headed by K. Bhargavan is under discussions.

2.2. Highlights of the Year

Moscova is proud of producing the following important results in 2011:

- 1 PhD was defended, another one will be defended on February 8, 2012.
- 1 paper accepted at POPL 2012 (1 paper was accepted at POPL 2011).

3. Scientific Foundations

3.1. Concurrency theory

Milner started the theory of concurrency in 1980 at Edinburgh. He proposed the calculus of communicating systems (CCS) as an algebra modeling interaction [36]. This theory was amongst the most important to present a compositional process language. Furthermore, it included a novel definition of operational equivalence, which has been the source of many articles, most of them quite subtle. In 1989, R. Milner, J. Parrow and D. Walker [37] introduced a new calculus, the *pi-calculus*, capable of handling reconfigurable systems. This theory has been refined by D. Sangiorgi (Edinburgh/INRIA Sophia/Bologna) and others. Many variants of the pi-calculus have been developed since 1989.

We developed a variant, called the Join-calculus [11], [12], a variant easier to implement in a distributed environment. Its purpose is to avoid the use of atomic broadcast to implement fair scheduling of processes. The Join-calculus allows concurrent and distributed programming, and simple communication between remote processes. It was designed with locations of processes and channels. It leads smoothly to the design and implementation of high-level languages which take into account low-level features such as the locations of objects.

The Join-calculus has higher-order channels as in the pi-calculus; channels names can be passed as values. However there are several restrictions: a channel name passed as argument cannot become a receiver; a receiver is permanent and has a single location, which allows one to identify channel names with their receivers. The loss of expressibility induced by these restrictions is compensated by joined receivers. A guard may wait on several receivers before triggering a new action. This is the way to achieve rendez-vous between processes. In fact, the notation of the Join-calculus is very near the natural description of distributed algorithms.

The second important feature of the Join-calculus is the concept of location. A location is a set of channels co-residing at the same place. The unit of migration is the location. Locations are structured as trees. When a location migrates, all of its sub-locations move too.

The Join-calculus, renamed Jocaml, has been fully integrated into Ocaml. Locations and channels are new features; they may be manipulated by or can handle any Ocaml values. Unfortunately the newer versions of Ocaml do not support them. We are still planning for both systems to converge.

3.2. Type systems

Types [38] are used in the theory of programming languages to guarantee (usually static) integrity of computations. Types are also used for static analysis of programs. The theory of types is used in Moscova to ensure safety properties about abstract values exchanged by two run-time environments; to define inheritance on concurrent objects in current extensions of Jocaml; to guarantee access control policies in Java- or $C\#$-like libraries.

3.3. Formal security

Formal properties for security in distributed systems started in the 90's with the BAN (Burrows, Abadi, Needham) logic paper. It became since a very active theory dealing with usual properties such as privacy, integrity, authentication, anonymity, repudiation, deniability, etc. This theory, which is not far from Concurrency theory, is relevant with the new activity of Moscova in the Microsoft Research-INRIA Joint Centre.

4. Application Domains

4.1. Telecoms and Interfaces

Distributed programming with mobility appears in the programming of the web and in autonomous mobile systems. It can be used for customization of user interfaces and for communications between several clients. Telecommunications is an other example application, with active networks, hot reconfigurations, and intelligent systems. For instance, France Telecom (Lannion) designs a system programmed in mobile Erlang.

4.2. Software Engineering

Security and Concurrency are two critical issues in software engineering. Any methodology based on formal verification of secure operations is fundamental in the ease of development of applications based on multi-tasking and networking. For instance, Microsoft is much interested by these topics.

5. Software

5.1. F7: Refinement Types for F#

Participants: Karthikeyan Bhargavan [correspondant], Cédric Fournet [MSR Cambridge], Andrew D. Gordon [MSR Cambridge].

F7 is an enhanced typechecker for the F# programming language that enables static checking of properties expressed as refinement types.

A refinement type is a base type qualified with a logical formula; the formula can express invariants, preconditions, and postconditions. F7 relies on type annotations, including refinements, provided in specific interface files. While checking code, F7 generates many logical problems which it solves by submitting to Z3, an external theorem prover for first-order logic (de Moura and Bjørner 2008). Finally, F7 erases all refinements and yields ordinary F# modules and interfaces.

Our main aim is to use F7 for the verification of security-critical programs. We have used it to verify implementations of access control mechanisms, multi-party secure sessions, cryptographic protocols for web services security and federated authentication, and secure audit logs.

A first version of F7 was released in 2008. In 2011, we revised the F7 libraries and typechecker and ported it to the released version of F# for .NET 4.0. The second version of F7 was released in December 2011.

The typechecker is written in 16000 lines of F#, with an additional cryptographic library of 9000 lines, and sample code of more than 12000 lines.

5.2. JSTY: Logical Auditing of JavaScript Programs

Participants: Karthikeyan Bhargavan [correspondant], Sergio Maffeis [Imperial College], Ravinder Shankesi [U. of Illinois at Urbana Champain].

JSTY is a runtime monitoring and logical auditing framework for JavaScript web applications. It has three components: (1) a contract language for JavaScript that enables programmers to annotate their scripts with assumptions and goals written as first-order logic pre- and post-conditions; (2) a runtime monitor implemented as a browser extension in the web browser Chrome that interprets these contracts at runtime and generates proof obligations for an SMT solver; (3) a logical auditor that checks proof obligations and maps counterexamples to violations of program correctness goals.

The target applications for JSTY include browser extensions as well as website scripts. In the case of browser extensions, our goal is to help extension writers to test their code by annotating it with logical contracts and auditing the code with JSTY. For website scripts, our goal is to check whether a website script obeys a generic security policy expressed as pre-conditions on functions in the browser or DOM API. We have used JSTY to analyze a variety of security-critical browser extensions and website scripts and found several vulnerabilities. We are currently incorpotating static checking into JSTY.

JSTY is written in about 1000 lines of JavaScript and we plan a public release in 2012.

5.3. OTT: Tool support for the working semanticist

Participants: Peter Sewell [U. of Cambridge], Francesco Zappa Nardelli [correspondant].

Ott is a tool for writing definitions of programming languages and calculi. It takes as input a definition of a language syntax and semantics, in a concise and readable ASCII notation that is close to what one would write in informal mathematics. It generates output:

1. a LaTeX source file that defines commands to build a typeset version of the definition;
2. a Coq version of the definition;
3. an Isabelle version of the definition; and
4. a HOL version of the definition.

Additionally, it can be run as a filter, taking a LaTeX/Coq/Isabelle/HOL source file with embedded (symbolic) terms of the defined language, parsing them and replacing them by typeset terms.

The main goal of the Ott tool is to support work on large programming language definitions, where the scale makes it hard to keep a definition internally consistent, and to keep a tight correspondence between a definition and implementations. We also wish to ease rapid prototyping work with smaller calculi, and to make it easier to exchange definitions and definition fragments between groups. The theorem-prover backends should enable a smooth transition between use of informal and formal mathematics.

In collaboration with Peter Sewell (Cambridge University).

The current version of Ott is about 30000 lines of OCaml. The tool is available from http://moscova.inria.fr/~zappa/software/ott (BSD licence).

Since its release in December 2007, the tool has been used in several projects, including a large proof of type preservation for the OCaml language (without modules) done by Scott Owens.

In 2011, apart from minor bug-fixes and features added, we implemented several performance improvements which result in a up-to 6x speed-up, and kept the Isabelle and Coq backend up-to date with the theorem prover evolution.

The currently released version is 0.21.1.

5.4. Lem, a tool for lightweight executable mathematics

Participants: Scott Owens [U. of Cambridge], Peter Sewell [U. of Cambridge], Francesco Zappa Nardelli [correspondant].

Lem is a lightweight tool for writing, managing, and publishing large scale semantic definitions. It is also intended as an intermediate language for generating definitions from domain-specific tools, and for porting definitions between interactive theorem proving systems (such as Coq, HOL4, and Isabelle). As such it is a complementary tool to Ott.

Lem resembles a pure subset of Objective Caml, supporting typical functional programming constructs, including top-level parametric polymorphism, datatypes, records, higher-order functions, and pattern matching. It also supports common logical mechanisms including list and set comprehensions, universal and existential quantifiers, and inductively defined relations. From this, Lem generates OCaml, HOL4 and Isabelle code; the OCaml backend uses a finite set library (and does not yet support inductive relations). A Coq backend is in development.

Lem is already in use at Cambridge and INRIA for research on relaxed-memory concurrency. We are currently preparing a feature-complete release with back-ends for HOL4, Isabelle/HOL, Coq, OCaml, and LaTeX. The project web-page is http://www.cl.cam.ac.uk/~so294/lem/. A paper on a Lem prototype appeared in ITP 2011, in the “rough diamond” category [25].

5.5. Memevents-Litmus-Diy-Dont

Participants: Jade Alglave, Luc Maranget [correspondant], Susmit Sarkar [U. of Cambridge, UK], Peter Sewell [U. of Cambridge, UK].

Luc Maranget is the main developer of the tools suite of project “Weak Memory Models” (cf. the relevant section).

This suite features three subtools *memevents* (model checker), *litmus* (runs tests on actual machines) and *diy* (generate tests from concise specifications). This year saw a new tool and one official releases (with documentation) [33] — see also http://diy.inria.fr. The releases feature all tools except *memevents*, which we wish to keep for ourselves.

This year main extensions are the handling of the ARM architecture and more collaboration between tools. For the latter, the test generator *diy* enrichs tests with meta-data that are exploited by *litmus*, so as to perform binding of test threads to machine processors, intelligent prefetch of data etc. We plan a new release of the tool suiteearly next year.

A new, independant, “proof of concept” tool, *offence* was written by J. Alglave and L. Maranget, as a support of our publication [30].

This software is available at http://diy.inria.fr/offence.

5.6. Jocaml

Participants: Luc Maranget, Xavier Clerc [correspondant].

Jocaml is an implementation of the join-calculus integrated into Ocaml. With respect to previous join-language prototypes, the most salient feature of the new prototype is a better integration into Ocaml. We achieve binary compatibility with Ocaml, moreover Jocaml releases now follow Ocaml releases. See previous year reports for details on Jocaml. The current version is 3.12.1 (released in September [34]) is available at http://jocaml.inria.fr/.

This new release features an extended Jocaml specific library that provide programmers with an easier access to concurrency and distribution:

1. Some utilities to parse command line, organize client-server connection, etc. This code was written partly by Xavier Clerc, engineer at INRIA SED department.
2. Some new abstractions of text channels help for writing text oriented applications.

5.7. Hevea

Participant: Luc Maranget [correspondant].

Hevea is a fast translator from full LaTeX to HTML, written in Ocaml. Hevea is highly configurable with commands written in LaTeX. Mathematics are rendered with UNICODE characters for symbols and HTML tables for formatting. Hevea produces HTML 4.0, enriched by css files. Hevea comes with Hacha companion, which produces a set of HTML pages (for instance, one page per chapter). Since it is very efficient and configurable, Hevea is adequate for on-line manuals or teaching courses.

This year saw a few developpements around Hevea, mostly for maintenance. Hevea is available at http://hevea.inria.fr/.

6. New Results

6.1. Weak Memory Models, Litmus-PPC-WMM tools

Participants: Jade Alglave, Luc Maranget, Pankaj Pawan, Susmit Sarkar [U. of Cambridge], Peter Sewell [U. of Cambridge], Francesco Zappa Nardelli.

Shared memory multiprocessors typically expose subtle, poorly understood and poorly specified relaxed-memory semantics to programmers. To understand them, and to develop formal models to use in program verification, we find it essential to take an empirical approach, testing what results parallel programs can actually produce when executed on the hardware. We describe a key ingredient of our approach, our litmus tool, which takes small "litmus test" programs and runs them for many iterations to find interesting behaviour. It embodies various techniques for making such interesting behaviour appear more frequently. We presented a tool, *litmus*, to run "litmus tests". on real hardware at TACAS'11 [31].

An automated exploration of machine memory models (on the *dont* tool) is submitted to TACAS'12.

During a two month long intership, Pankaj Pawan (IIT Kanpur, India) ported the PPCMEM application from OCaml to JavaScript, and developed a suitable web-interface. This enabled a wide dissemination (including at IBM) of the PPCMEM tool. The tool is available online at: http://www.cl.cam.ac.uk/~pes20/ppcmem/help.html.

6.2. Operational semantics for the memory model of Power (IBM) machines

Participants: Jade Alglave, Luc Maranget, Susmit Sarkar [U. of Cambridge], Peter Sewell [U. of Cambridge], Derek Williams [IBM, Austin].

Exploiting today's multiprocessors requires high-performance and correct concurrent systems code (optimising compilers, language runtimes, OS kernels, etc.), which in turn requires a good understanding of the observable processor behaviour that can be relied on. Unfortunately this critical hardware/software interface is not at all clear for several current multiprocessors.

We characterise the behaviour of IBM POWER multiprocessors, which have a subtle and highly relaxed memory model (ARM multiprocessors have a very similar architecture in this respect). We have conducted extensive experiments on several generations of processors: POWER G5, 5, 6, and 7. Based on these, on published details of the microarchitectures, and on discussions with IBM staff, we give an abstract-machine semantics that abstracts from most of the implementation detail but explains the behaviour of a range of subtle examples. Our semantics is explained in prose but defined in rigorous machine-processed mathematics; we also confirm that it captures the observable processor behaviour, or the architectural intent, for our examples with an executable checker. While not officially sanctioned by the vendor, we believe that this model gives a reasonable basis for reasoning about current POWER multiprocessors. Our work should bring new clarity to concurrent systems programming for these architectures, and is a necessary precondition for any analysis or verification. It should also inform the design of languages such as C and C++, where the language memory model is constrained by what can be efficiently compiled to such multiprocessors.

This work was presented at PLDI'11 [26]. This operational model is now being enriched for Power machine with loads and link/store conditionals (the Power primitives to write locks), and connected to C++ semantics. It is submitted to PLDI'2012 with many co-authors from Cambridge.

6.3. Restoring Sequential Consistency for x86 and Power machines

Participants: Jade Alglave, Luc Maranget.

Concurrent programs running on weak memory models exhibit relaxed behaviours, making them hard to understand and to debug. To use standard verification techniques on such programs, we can force them to behave as if running on a Sequentially Consistent (SC) model. Thus, we examine how to constrain the behaviour of such programs via synchronisation to ensure what we call their stability, i.e. that they behave as if they were running on a stronger model than the actual one, e.g. SC. First, we define sufficient conditions ensuring stability to a program, and show that Power's locks and read-modify-write primitives meet them. Second, we minimise the amount of required synchronisation by characterising which parts of a given execution should be synchronised. Third, we characterise the programs stable from a weak architecture to SC. Finally, we present our offence tool which places either lock-based or lock-free synchronisation in a x86 or Power program to ensure its stability.

This work was presented at CAV'11 [30]

6.4. Relaxed-Memory Concurrency and Verified Compilation

Participants: Jaroslav Ševčík [U. of Cambridge], Peter Sewell [U. of Cambridge], Jagannathan Suresh [U. of Cambridge], Viktor Vafeiadis [U. of Cambridge], Francesco Zappa Nardelli.

We studied the semantic design and verified compilation of a C-like programming language for concurrent shared-memory computation above x86 multiprocessors. The design of such a language is made surprisingly subtle by several factors: the relaxed-memory behaviour of the hardware, the effects of compiler optimisation on concurrent code, the need to support high-performance concurrent algorithms, and the desire for a reasonably simple programming model. In turn, this complexity makes verified (or verifying) compilation both essential and challenging.

This project started in 2010, when we defined a concurrent relaxed-memory semantics for ClightTSO, an extension of CompCert's Clight in which the processor memory model is exposed for high-performance code, and, building on CompCert, we implemented a certifying compiler from ClightTSO to x86, and proved in Coq several compilation phases. A paper describing our approach has been accepted in POPL [29] 2011.

During 2011 we completed this project by developping correctness proofs for all the compilation phases, and made a public distribution of the compiler, available from http://www.cl.cam.ac.uk/~pes20/CompCertTSO.

In 2011 Zappa Nardelli and Vafeiadis investigated the soundness of fence elimination optimisations for x86TSO. They implemented and proved correct two optimisations that remove redundant fence instructions as compiler passes over RTL in CompCertTSO. Despite an apparent simplicity, these optimisations generate almost optimal code for several standard concurrent algorithms (CompCertTSO does not implement escape analysis, which would enhance the effectiveness of the optimisations), and since they only perform data-flow analysis over the code of each thread (without analysing the full-system thread interactions) they do not suffer form the finite-state and finite control limitation of other approaches. The proof of correctness of one optimisation was challenging has required some for prophecy variable simulation. This work has been published in SAS 2011 [28] and the code is part of CompCertTSO.

A journal version, describing the correctness proof of all the phases of CompCertTSO (including the fence eliminations) as been submitted to the Journal of the ACM [35].

6.5. Compiling C/C++ Concurrency: from C++11 to POWER

Participants: Kayvan Memarian, Francesco Zappa Nardelli.

The upcoming C and C++ revised standards add concurrency to the languages, for the first time, in the form of a subtle relaxed memory model (the C++11 model). This aims to permit compiler optimisation and to accommodate the differing relaxed-memory behaviours of mainstream multiprocessors, combining simple semantics for most code with high-performance low-level atomics for concurrency libraries.

We studied the the correctness of two proposed compilation schemes for the C++11 load and store concurrency primitives to Power assembly, having noted that an earlier proposal was flawed. (The main ideas apply also to ARM, which has a similar relaxed memory architecture.)

This should inform the ongoing development of production compilers for C++11 and C1x, clarifies what properties of the machine architecture are required, and builds confidence in the C++11 and Power semantics.

A paper describing this work will appear in POPL 2012 [22].

6.6. F*: Secure Distributed Programming with Value-Dependent Types

Participants: Nikhil Swamy [MSR Redmond], Juan Chen [MSR Redmond], Cédric Fournet [MSR Cambridge], Pierre-Yves Strub [MSR-INRIA], Karthikeyan Bhargavan [correspondant], Jean Yang [MIT].

Distributed applications are difficult to program reliably and securely. Dependently typed functional languages promise to prevent broad classes of errors and vulnerabilities, and to enable program verification to proceed side-by-side with development. However, as recursion, effects, and rich libraries are added, using types to reason about programs, specifications, and proofs becomes challenging.

We present F*, a full-fledged design and implementation of a new dependently typed language for secure distributed programming. Unlike prior languages, F* provides arbitrary recursion while maintaining a logically consistent core; it enables modular reasoning about state and other effects using affine types; and it supports proofs of refinement properties using a mixture of cryptographic evidence and logical proof terms. The key mechanism is a new kind system that tracks several sub-languages within F* and controls their interaction. F* subsumes two previous languages, F7 and Fine. We prove type soundness (with proofs mechanized in Coq) and logical consistency for F*.

We have implemented a compiler that translates F* to .NET bytecode, based on a prototype for Fine. F* provides access to libraries for concurrency, networking, cryptography, and interoperability with C#, F#, and the other .NET languages. The compiler produces verifiable binaries with 60size overhead for proofs and types, as much as a 45x improvement over the Fine compiler, while still enabling efficient bytecode verification.

To date, we have programmed and verified more than 20,000 lines of F* including (1) new schemes for multi-party sessions; (2) a zero-knowledge privacy-preserving payment protocol; (3) a provenance-aware curated database; (4) a suite of 17 web-browser extensions verified for authorization properties; and (5) a cloud-hosted multi-tier web application with a verified reference monitor.

This paper was published at ICFP 2011 [27].

6.7. Security by Typing for Cryptographic Protocol Implementations

Participants: Karthikeyan Bhargavan [correspondant], Cédric Fournet [MSR Cambridge], Andrew D. Gordon [MSR Cambridge], Alfredo Pironti.

We propose to use refinement typing to verify the security of cryptographic protocol implementations. Our method is based on declaring and enforcing invariants on the usage of cryptography. We develop cryptographic libraries that embed a logic model of their cryptographic structures and that specify preconditions and postconditions on their functions so as to maintain their invariants.

We implement the method for protocols coded in F# and verified using F7, our SMT-based typechecker for refinement types, that is, types carrying formulas to record invariants. As illustrated by a series of programming examples, our method can flexibly deal with a range of different cryptographic constructions and protocols [24].

We are currently evaluating this method on a fully-fledged implementation of TLS. While previous uses of typing for cryptographic protocol implementations focused on the symbolic model of cryptography, we use a new technique by Fournet et al to develop computational proofs for our implementations. Our TLS implementation consists of 6000 lines of code. We have currently annotated and verified about half of this implentation.

We recently published a tutorial on our verification method as part of the proceedings of FOSAD 2010, and a journal paper on our type system at TOPLAS [20].

6.8. Authorization for the Social Web: from Formal Analysis to Concrete Attacks

Participants: Chetan Bansal [BITS Goa], Karthikeyan Bhargavan [correspondant], Sergio Maffeis [Imperial College].

Social sign-on and social sharing are becoming an ever more popular feature of web applications. This success is caused in part by the APIs support offered by leading websites such as Facebook, Twitter and Google, and by the openness of standards such as OAuth 2.0. A formal analysis of such protocols must account for malicious websites and their JavaScript, and common website vulnerabilities, such as cross site request forgery and open redirectors.

We present a formal model for web application protocols called WebSpi, implemented as a library for the protocol verification tool ProVerif. We use WebSpi to model and verify several configurations of the OAuth 2.0 protocol. We show that our formal analysis can be used to reconstruct concrete website attacks. Our approach is validated by finding dozens of previously unknown vulnerabilities in popular websites such as Yahoo and Wordpress, when they connect to social networks such as Twitter and Facebook.

We are in discussion with Facebook, Twitter and other websites to address the vulnerabilites found by our analysis. We have submitted a paper describing this work, and plan to release the WebSpi library in 2012.

6.9. Verified Android Cryptographic Applications

Participants: Karthikeyan Bhargavan [correspondant], Quentin Lefebvre [MPRI].

With the emergence of application markets for smartphones, hundreds of third-party applications now use cryptography to protect sensitive user data before storing it on disk or sending it out on the network. However, using cryptographic mechanisms correctly to fulfill a desired security goal is challenging and error-prone, even for experts. Our goal is to build verification tools that developers may use to develop security proofs for their applications.

We show how to verify the security of third-party cryptographic applications written in Java for the Android platform. We first develop symbolic security specifications for classes in the JCA. We can then verify that applications that use these libraries satisfy their security goals, even in the presence of a Dolev-Yao adversary who controls the network, the disk, and potentially other applications on the device. We report preliminary verification results using the Krakatoa verification tool for Java programs.

We presented this work at the ASA workshop 2011 [23].

6.10. Logical Auditing of JavaScript Programs for Security

Participants: Karthikeyan Bhargavan [correspondant], Sergio Maffeis [Imperial College], Ravinder Shankesi [UIUC].

Client side web applications are error-prone and hard to secure, as proven by frequent vulnerability reports. We experiment with using logical annotations as a means to specify inlined security policies for web pages, and we implement a run-time monitoring system that generates a logical trace of the program execution. Feeding the logical trace to external theorem provers, it is possible to detect, post-facto, violations of the security policies, helping the on-line debugging of web applications.

We present JSTY a browser-based logical auditing framework for JavaScript programs. We show how first-order logic contracts can be used to express cryptographic assumptions and security goals for JavaScript ptograms that use cryptography. We demonstrate our approach on realistic examples, including browser extensions for password management. We find security vulnerabilities in commercial products by logical auditing. This work is currently unpublished.

6.11. Secure Interpreters for Sessions

Participants: James Leifer, Jean Pichon [intern from ENST Telecom ParisTech].

We developed an interpreter for decentralised multi-party sessions. The interpreter takes a graph-based description of a session and provides a high-level interface for sending and receiving messages permissible in the session. The interpreter protects the integrity of session execution in a realistic security setting where an adversary has the ability to: (1) control the network to capture and reinject messages at will, and read and forge messages using leaked cryptographic keys; (2) compromise arbitrary session participants. By producing and verifying cryptographic signatures, the interpreter ensures that all non-compromised participants have consistent views of the session's execution history.

We previously worked on a session compiler. The compiler took as input "local graphs" described in process-calculus fashion. It produced one F# library for each role, and an F7 interface/specification for this library. If the library typechecked (with the F7 type-checker) against its interface, then it was secure. However, the production of the library was not itself verified, and could fail.

By contrast, this present work is concerned with a session interpreter. This interpreter works like an ML functor (i.e. a compile-time function from modules to modules). It takes as input a module describing a global graph by exposing a specific interface. The application of the interpreter functor to a graph module yields an F☆ module that contains the interface/specification. The interpreter functor, being checked against an abstract description of a graph, is typechecked "once and for all". Because the interpreter functor typechecks against its interface (that contains specifications), it is secure (through refinements). The interpreter functor can then be applied to a concrete session graph to be used.

The interpreter is written in F☆, a dialect of ML enriched with refinement types, and its correctness is proven by type annotations.

The interpreter consists of approximately 3000 lines of code.

6.12. Models of Audit Logs

Participants: Karthikeyan Bhargavan, Cédric Fournet [MSR Cambridge], Nataliya Guts, Francesco Zappa Nardelli.

This line of research was accurately described in last year activity report of Moscova. Here we just mention that Nataliya Guts defended her PhD [19] on "Auditability for security protocols" on January 11th, 2011.

7. Contracts and Grants with Industry

7.1. Grants with Industry

In 2006, we started to work at the Microsoft Research-INRIA Joint Centre in a common project with Cédric Fournet (MSR Cambridge), Gilles Barthe (now at IMDEA), Nataliya Guts (who defended her PhD in January 2011) and Jérémy Planul (who will defend on February 2012). The project is named *Secure Distributed Computations and their Proofs* and deals with security, programming languages theory and formal proofs. This work is still under active collaboration within all year 2011.

8. Partnerships and Cooperations

8.1. European Initiatives

8.1.1. FP7 Projet

8.1.1.1. CRYSP

Title:CRYSP: A Novel Framework for Collaboratively Building Cryptographically Secure Programs and their Proofs

Type: IDEAS ()

Instrument: ERC Starting Grant (Starting)

Duration: November 2010 - October 2015

Coordinator: Karthikeyan Bhargavan, INRIA (France)

Abstract: The goal of this grant proposal is to develop a collaborative specification framework and to build incremental, modular, scalable verification techniques that enable a group of collaborating programmers to build an application and its security proof side-by-side. We propose to validate this framework by developing the first large-scale web application and full-featured cryptographic protocol libraries with formal proofs of security.

8.2. International Initiatives

We are Équipe Associée with Computer lab at University of Cambridge (P. Sewell et al).

9. Dissemination

9.1. Animation of the scientific community

+ J. Leifer was a member of the Program Committee of JFLA'11.
+ J.-J. Lévy is director of the Microsoft Research-INRIA Joint Centre, see http://msr-inria.inria.fr.
+ J.-J. Lévy is member of the Scientific Committee of the "Fondation Sciences Mathématiques de Paris" and participates to corresponding meetings and juries.
+ J.-J. Lévy is member of the Program Committee of Digiteo as representative of INRIA–Paris-Rocquencourt.
+ L. Maranget is elected member of the INRIA *Comité technique paritaire*. He participates to one meeting every two months, discussing about the politics of Inria at the highest level.

+ L. Maranget is president of *CUMIR-R Commission des utilisateurs des moyens informatiques de Rocquencourt-Recherche*.
+ F. Zappa Nardelli is member of the *Comité Directeur* of the CEA-EDF-INRIA summer schools.
+ F. Zappa Nardelli is the correspondent of the *Équipes associates MM*.
+ F. Zappa Nardelli is member of the *comité de suivi doctoral* de l'INRIA Saclay.
+ F. Zappa Nardelli served in the POPL 2012 PC. He attended the PC meeting on September, 29-2 at U. Maryland, USA.
+ F. Zappa Nardelli organised the POPL PC Workshop.
+ January 11, F. Zappa Nardelli served in Nataliya Guts PhD Jury.
+ K. Bhargavan and J. Leifer organised the security seminar series. Leifer continued in 2011 with a series of talks by invited researchers: 11-01 (Graham Steel, Nataliya Guts), 09-03 (François Dupressoir), 15-03 (David Cadé), 28-04 (Nikhil Swamy), 01-06 (Kristin Lauter).

9.2. Teaching

Our project-team participates to the following courses:

- October-February, K. Bhargavan taught courses in programming as a Chargé d'Enseignement at the École Polytechnique in 2011.
- October-November, J. Leifer participated to the "Concurrency" course, Master Parisien de Recherche en informatique (MPRI), 2011-2012, at U. of Paris 7, 23 students, (J. Leifer taught the pi-calculus semantics: 15 hours plus the mid-term exam)
- J.-J. Lévy taught "Reductions and Causality", an advanced course on the lambda-calculus in October-November, Tsinghua University, Beijing (15 students, 12h) http://jeanjacqueslevy.net/courses/tsinghua/reductions.
- J.-J. Lévy wrote a vulgarisation article on "Les dominos de Wang" in *Quadrature*, journal for the students of the French *Classes Préparatoires*.
- L. Maranget is the coordinator of the computer science examinations of the *Concours d'entrée à l'École polytechnique*.
- January, F. Zappa Nardelli was responsible of a quarter of the Master course "Concurrency", Master Parisien de Recherche en Informatique, where he lectured about proof methods for concurrent programs (12 hours of lecture plus a final exam, about 30 students).
- February, F. Zappa Nardelli lectured at ENS Lyon on "Shared memory: an elusive abstraction" (4h, about 15 students).
- June, 20-23, F. Zappa Nardelli lectured on "Shared memory: an elusive abstraction" at the UPMARC Summer School, Sweden (4h, about 60 students).
- November, 7-11, F. Zappa Nardelli was teaching assistant at the "Introduction to the Coq proof assistant" CEA-EDF-INRIA summer school (20 hours, about 25 students).

Training and Supervisions:

- K. Bhargavan supervised 2 Masters internships (stagiaires) in 2011: Quentin Lefebvre and Evmorfia-Iro Bartzia.
- K. Bhargavan supervised 2 summer internships: Chetan Bansal and Ravinder Shankesi.
- K. Bhargavan is supervising Evmorfia-Iro Bartzia as a PhD. student since October 2011.
- K. Bhargavan is supervising Alfredo Pironti as a post-doc since 2010.
- J. Leifer supervised Jean Pichon, a Masters (M1) research intern from ENST Telecom ParisTech, from 1 July - 31 December 2011. The topic of the research project was "A verified interpreter for secure multiparty sessions". Pichon will hold his defence in January 2012.
- F. Zappa Nardelli supervised the 6 months long M2 internship of Kayvan Memarian (ENS Cachan) on "Correctness of compilation of C++11 low-level atomics".

9.3. Participations to conferences, Seminars, Invitations

9.3.1. Participations to conferences

- January 24-30, F. Zappa Nardelli attended the POPL conference in Austin, Texas (US).
- January 29-Feb. 1, J. Leifer, J.-J. Lévy, L. Maranget attended the JFLA'11, La Bresse France. L. Maranget gave a tutorial on Jocaml, 5 hours, for a public of about 40 reseachers.
- February 2-5, L. Maranget attended the GGJJ 2011, in Gérardmer. He gave an invited talk with Jade Alglave.
- February 2-5, F. Zappa Nardelli attended the GGJJ 2011, in Gérardmer. He gave a talk on "relaxed memory models must be rigorous".
- March 22-24, All Moscova attended the INRIA evaluation seminar, Paris.
- March 26 - April 3, L. Maranget attended TACAS'11, in Saarbrücken. He gave a talk and a tool presentation [31].
- April 18-22, J. Leifer attended the "Behavioural Types Workshop", Lisbon, Portugal. He presented a talk "Secure protocol synthesis for multi-party sessions" and was an invited member of a panel on "Comparing approaches to behavioural types".
- June 6–8, L. Maranget attended PLDI'11, San Jose, California. His co-author S. Sarkar gave a talk of this article [26].
- July 16–20, L. Maranget attended CAV'11, in Snowbird, Utah. His co-author J. Alglave gave a talk of this article [30].
- June 26-29: K.Bhargavan, J. Leifer, J.-J. Lévy attended CSF 2011, "The 24th IEEE Computer Security Foundations Symposium", Domaine de l'Abbaye des Vaux de Cernay, France.
- September, 2, F. Zappa Nardelli gave a talk on "Veryifing Fence elimination Optimisations" at the POPL PC workshop.

9.3.2. Seminars

- January 31, F. Zappa Nardelli gave a talk on relaxed memory models at the Parkas seminar, Paris.
- March 3, F. Zappa Nardelli gave a talk on relaxed memory models at the "groupe de travail Programmation", Paris.
- April, 7-10, F. Zappa Nardelli visited Cambridge University for collaboration with Peter Sewell.
- April, 7-10, K. Memarian visited Cambridge University for collaboration with Peter Sewell.
- May 5, K. Bhargavan gave a seminar as part of the *le Modèle et l'Algorithme* series in INRIA Rocquencourt.
- June 1–4, L. Maranget visited IBM in Austin, Texas, for discussion with D. Williams (IBM), J. Alglave (U. of Oxford), S. Sarkar and P. Sewell (U. of Cambridge).
- July, K. Memarian visited Cambridge University for collaboration with Peter Sewell.

9.3.3. Invited visitors

- Sergio Maffeis (Imperial College) visited the Moscova EPI as invited professor from June to November and worked with K. Bhargavan.
- June 24, Mike Hicks (U. Maryland) visited the Moscova EPI and gave a talk on "monads in OCaml".
- June-July, Pankaj Pawan (IIT Kanpur) was intern student in the Moscova project team under the supervision of F. Zappa Nardelli.
- May-September, Kayvan Memarian (ENS Cachan) was intern student in the Moscova project team under the supervision of F. Zappa Nardelli.
- May-August, Jan Vitek (U. Purdue) visited the Moscova project as Invited Professor. He collaborated with F. Zappa Nardelli.

- June, Gregor Richards (U. Purdue) visited the Moscova project for collaboration with F. Zappa Nardelli and Jan Vitek.

10. Bibliography

Major publications by the team in recent years

[1] G. BARTHE, C. FOURNET (editors). *Trustworthy Global Computing, Third Symposium, TGC 2007, Sophia-Antipolis, France, November 5-6, 2007, Revised Selected Papers*, Lecture Notes in Computer Science, Springer, 2008, vol. 4912.

[2] *20th IEEE Computer Security Foundations Symposium, CSF 2007, 6-8 July 2007, Venice, Italy*, IEEE Computer Society, 2007.

[3] *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*, IEEE Computer Society, 2009.

[4] P. NING, P. F. SYVERSON, S. JHA (editors). *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, ACM, 2008.

[5] F. BESSON, T. BLANC, C. FOURNET, A. D. GORDON. *From Stack Inspction to Access Control: A Security Analysis for Libraries*, in "17th IEEE Computer Security Foundations Workshop", June 2004, p. 61–75.

[6] K. BHARGAVAN, R. CORIN, P.-M. DENIÉLOU, C. FOURNET, J. J. LEIFER. *Cryptographic Protocol Synthesis and Verification for Multiparty Sessions*, in "CSF", IEEE Computer Society, 2009, p. 124-140.

[7] R. CORIN, P.-M. DENIÉLOU, C. FOURNET, K. BHARGAVAN, J. J. LEIFER. *Secure Implementations for Typed Session Abstractions*, in "20th IEEE Computer Security Foundations Symposium (CSF'07)", Venice, Italy, IEEE, July 2007, p. 170–186, http://www.msr-inria.inria.fr/projects/sec/sessions/.

[8] R. CORIN, P.-M. DENIÉLOU. *A Protocol Compiler for Secure Sessions in ML*, in "TGC", G. BARTHE, C. FOURNET (editors), Lecture Notes in Computer Science, Springer, 2007, vol. 4912, p. 276-293.

[9] R. CORIN, P.-M. DENIÉLOU, C. FOURNET, K. BHARGAVAN, J. J. LEIFER. *Secure Implementations for Typed Session Abstractions*, in "CSF", IEEE Computer Society, 2007, p. 170-186.

[10] R. CORIN, P.-M. DENIÉLOU, C. FOURNET, K. BHARGAVAN, J. J. LEIFER. *A secure compiler for session abstractions*, in "Journal of Computer Security", 2008, vol. 16, n^{o} 5, p. 573-636.

[11] C. FOURNET, G. GONTHIER. *The Reflexive Chemical Abstract Machine and the Join-Calculus*, in "Proceedings of the 23rd Annual Symposium on Principles of Programming Languages (POPL)", (St. Petersburg Beach, Florida), ACM, January 1996, p. 372–385.

[12] C. FOURNET, G. GONTHIER, J.-J. LÉVY, L. MARANGET, D. RÉMY. *A Calculus of Mobile Agents*, in "CONCUR '96: Concurrency Theory (7th International Conference)", Pisa, Italy, U. MONTANARI, V. SASSONE (editors), LNCS, Springer, August 1996, vol. 1119, p. 406–421.

[13] C. FOURNET, C. LANEVE, L. MARANGET, D. RÉMY. *Inheritance in the join calculus*, in "Journal of Logics and Algebraic Programming", September 2003, vol. 57, n^{o} 1–2, p. 23–29.

[14] A. HOBOR, A. APPEL, F. ZAPPA NARDELLI. *Oracle Semantics for Concurrent Separation Logic*, in "17th European Symposium on Programming (ESOP'08)", April 2007.

[15] J. J. LEIFER, G. PESKINE, P. SEWELL, K. WANSBROUGH. *Global abstraction-safe marshalling with hash types*, in "Proc. 8th ICFP", 2003, Extended Abstract of INRIA Research Report 4851, http://hal.inria.fr/inria-00071732/fr/.

[16] M. MERRO, F. ZAPPA NARDELLI. *Behavioural theory for Mobile Ambients*, in "Journal of ACM", November 2005, vol. 52, n^{o} 6, p. 961–1023.

[17] M. QIN, L. MARANGET. *Expressive Synchronization Types for Inheritance in the Join Calculus*, in "Proceedings of APLAS'03", Beijing China, LNCS, Springer, November 2003.

[18] S. SARKAR, P. SEWELL, F. ZAPPA NARDELLI, S. OWENS, T. RIDGE, T. BRAIBANT, M. O. MYREEN, J. ALGLAVE. *The semantics of x86-CC multiprocessor machine code*, in "POPL", 2009, p. 379-391.

Publications of the year

Doctoral Dissertations and Habilitation Theses

[19] N. GUTS. *Auditability for security protocols*, Université Denis Diderot Paris 7, 2011.

Articles in International Peer-Reviewed Journal

[20] J. BENGTSON, K. BHARGAVAN, C. FOURNET, A. D. GORDON, S. MAFFEIS. *Refinement types for secure implementations*, in "ACM Trans. Program. Lang. Syst.", 2011, vol. 33, n^{o} 2, 8, http://research.microsoft.com/pubs/70624/MSR-TR-2008-118-SP2.pdf.

Articles in National Peer-Reviewed Journal

[21] J.-J. LÉVY. *Les dominos de Wang*, in "Revue Quadrature, “Magazine de mathématiques pures et épicées”", Octobre-novembre-décembre 2011, vol. 82, 5, http://jeanjacqueslevy.net/pubs/11quadrature.pdf.

International Conferences with Proceedings

[22] M. BATTY, K. MEMARIAN, S. OWENS, S. SARKAR, P. SEWELL. *Clarifying and Compiling C/C++ Concurrency: from C++11 to POWER*, in "Proc. POPL 2012", 2011, to appear.

[23] K. BHARGAVAN, Q. LEFEBVRE. *Verified Android Cryptographic Applications*, in "5th International Workshop on Analysis of Security APIs (ASA-5)", July 2011.

[24] C. FOURNET, K. BHARGAVAN, A. D. GORDON. *Cryptographic Verification by Typing for a Sample Protocol Implementation*, in "Foundations of Security Analysis and Design VI (FOSAD'10)", 2011, p. 66-100.

[25] S. OWENS, P. BÖHM, F. ZAPPA NARDELLI, P. SEWELL. *Lem: A Lightweight Tool for Heavyweight Semantics*, in "ITP", 2011, http://www.cl.cam.ac.uk/~so294/documents/itp11.pdf.

[26] S. SARKAR, P. SEWELL, J. ALGLAVE, L. MARANGET, D. WILLIAMS. *Understanding POWER Multiprocessors*, in "Proceedings of 32nd ACMSIGPLAN Conference on Programming Language Design and Implementation", June 2011, p. 175–186, http://www.cl.cam.ac.uk/~pes20/ppc-supplemental/pldi105-sarkar.pdf.

[27] N. SWAMY, J. CHEN, C. FOURNET, P.-Y. STRUB, K. BHARGAVAN, J. YANG. *Secure Distributed Programming with Value-dependent Types*, in "16th ACM SIGPLAN International Conference on Functional Programming", Tokyo, Japan, 2011, p. 266-278, Related Projects * F*: A Verifying ML Compiler for Distributed Programming, http://hal.inria.fr/inria-00596715/en.

[28] V. VAFEIADIS, F. ZAPPA NARDELLI. *Verifying Fence Elimination Optimisations*, in "in Proc. SAS 2011", 2011, http://www.cl.cam.ac.uk/~pes20/CompCertTSO/doc/fenceelim.pdf.

[29] J. ŠEVČÍK, V. VAFEIADIS, F. ZAPPA NARDELLI, P. SEWELL, S. JAGANNATHAN. *Relaxed-Memory Concurrency and Verified Compilation*, in "Proc. POPL", 2011, http://www.cl.cam.ac.uk/~pes20/CompCertTSO/doc/paper.pdf.

Scientific Books (or Scientific Book chapters)

[30] J. ALGLAVE, L. MARANGET. *Stability in Weak Memory Models*, in "Computer Aided Verification", G. GOPALAKRISHNAN, S. QADEER (editors), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2011, vol. 6806, p. 50–66, http://dx.doi.org/10.1007/978-3-642-22110-1_6.

[31] J. ALGLAVE, L. MARANGET, S. SARKAR, P. SEWELL. *Litmus Running Tests against Hardware*, in "Tools and Algorithms for the Construction and Analysis of Systems", P. ABDULLA, K. LEINO (editors), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2011, vol. 6605, p. 41-44.

Other Publications

[32] J. ALGLAVE, A. MAHBOUBI. *A Generic Formalised Framework for Reasoning About Weak Memory Models*, 2011, http://hal.inria.fr/inria-00604656/en.

[33] J. ALGLAVE, L. MARANGET, S. SARKAR, P. SEWELL. *diy, release 4.0*, January 2011, Software and documentation available, http://diy.inria.fr/.

[34] L. MARANGET, L. MANDEL, M. QIN. *JoCaml, release 3.12.1*, July 2011, Software and documentation available, http://jocaml.inria.fr/.

[35] J. ŠEVČÍK, V. VAFEIADIS, F. ZAPPA NARDELLI, P. SEWELL, S. JAGANNATHAN. *CompCertTSO: A Verified Compiler for Relaxed-Memory Concurrency*, 2011, submitted, http://moscova.inria.fr/~zappa/readings/compcerttso-long.pdf.

References in notes

[36] R. MILNER. *Communication and Concurrency*, International Series on Computer Science, Prentice Hall, 1989.

[37] R. MILNER, J. PARROW, D. WALKER. *A Calculus of Mobile Processes, Parts I and II*, in "Journal of Information and Computation", September 1992, vol. 100, p. 1–77.

[38] B. C. PIERCE. *Types and Programming Languages*, The MIT Press, 2002.