INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team moscova

Mobility, security, concurrence, verification and analysis

Paris - Rocquencourt

Theme : Programs, Verification and Proofs

Table of contents

1. Team

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

Jade Alglave [MESR grant, Paris 7, until 1/11/2010]
Nataliya Guts [INRIA-MSR grant, Paris 7]
Jérémy Planul [ENS-Lyon, Ecole Polytechnique]

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.

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 2010 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 2010, Khartik Bhargavan won an ERC Starting Grant. His CRYSP project, described further in this report, aims at Collaboratively Building Cryptographically Secure Programs and their Proofs. Pierre-Malo Deniélou defended his thesis (he is now postdoc at Imperial College), Jade Alglave defended her thesis (she is now postdoc at Oxford University), and Nataliya Guts will defend her PhD on 11 January 2011 (and then will start a postdoc at U. of Maryland). Jérémy Planul (ENS-Lyon and MPRI) accomplished his 2nd year of PhD.

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 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 finish at end of 2010 the project PARSEC, funded by the ANR (*Agence Nationale de la Recherche*), together with MIMOSA and LANDE project-teams of INRIA and the team of Roberto Amadio at CNRS-PPS, U. of Paris 7. This project is coordinated by G. Boudol.

2.2. Highlights

Moscova is proud of producing the following important results in 2010:

- 2 PhDs were defended, another one will be defended on January 11, 2011.
- 1 ERC Starting Grant by K. Bhargavan.
- 2 papers presented at POPL 2010.
- 5 new original released softwares.

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 [34]. 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 [35] 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 [36] 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 multitasking and networking. For instance, Microsoft is much interested by these topics.

5. Software

5.1. Memevents-Litmus-Diy-Dont

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

We developed a tool suite for our project about *Weak Memory Models* (see the relevant section).

In 2009, we implemented three tools: *memevents* (which checks our models), *litmus* (which runs tests on actual machines) and *diy* (which generates tests from concise specifications). This year, we introduced a new tool and three official releases (with documentation), see [31]. Our releases comprises all tools but *memevents* which is kept for our internal usage. The new tool is:

dont (included in *diy* with a total of 8500 lines of Ocaml, co-developed by Jade Alglave and Luc Maranget) which automates the analysis of the memory model (for x86 or Power). It outputs a memory model following the framework of [24].

We still access to the Power6 supercomputer in IDRIS (CNRS) for our tests, but we also bought a Power7 machine in late 2010. It will allow the new *dont* to perform deeper explorations of the Power memory model, and hopefully confirm our model.

A description of *litmus* will be presented at TACAS'2011 (March–April, Saarbrücken).

A common work by J. Alglave, Delphine Longuet (LRI, Paris Sud) and L. Maranget on Automatic Memory Model Exploration has also been submitted to conference. This paper is oriented towards testing with the *dont* tool.

5.2. F7 Typechecker

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

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 2010, in collaboration with Andrew D. Gordon and Cédric Fournet, we developed the second version of the typechecker with a completely new set of libraries, as described in our paper "Modular Verification of Security Protocol Code by Typing".

In 2010, in collaboration with Nataliya Guts, we extended the typechecker to support the automated verification of programs that use both cryptography and higher-order functions, as described in our paper "Typechecking Higher-Order Security Libraries".

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. Of these, the majority of the library and the samples were written in 2010. The second version of F7 is due to be released in December 2010.

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.

The current version of Ott is about 25000 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 2010 we re-engineered the tool, which now reflects the structure of the source definition in the generated output. This enables interleaving Ott and theorem-prover definitions, granting extra flexibility to the user and allowing complex languages to be defined.

A paper describing the metalanguage used by Ott and its semantics (including the new backend) appeared in the Journal of Functional Programming [23].

5.4. Secure Multi-party Sessions

Participants: Karthikeyan Bhargavan, Ricardo Corin [Universidad Nacional de Cordoba, Argentina], Pierre-Malo Deniélou [correspondant, Imperial College], Cédric Fournet [MSR Cambridge].

Web page at http://www.msr-inria.inria.fr/projects/sec/sessions/.

We have designed and implementated a compiler that, given high-level multiparty session descriptions, generates custom cryptographic protocols.

Our sessions specify pre-arranged patterns of message exchanges and data accesses between distributed participants. They provide each participant with strong security guarantees for all their messages.

Our compiler generates code for sending and receiving these messages, with cryptographic operations and checks, in order to enforce these guarantees against any adversary that may control both the network and some session participants. We furthermore verify that the generated code is secure by relying on a recent type system for cryptography. Most of the proof is performed by mechanized type checking, of the generated code, and does not rely on the correctness of our compiler. We obtain the strongest session security guarantees to date in a model that captures the actual details of protocol code.

Two central design goals guide our work on session implementation. First, all the cryptography required to protect compromised participants is completely hidden from the application programmer, who may reason about the behaviour of a distributed system as if it followed precisely the high level specification. (Thus all correspondence properties at the abstract level carry through to any distributed execution.) Second, all low-level network activity is in a one-to-one relationship with high-level communication, thus no additional messages are introduced.

Our compiler translates our session language to custom cryptographic protocols, coded as ML modules (both for $F\#$ with .NET cryptographic libraries, and for Ocaml with OpenSSL libraries), which can be linked to application code for each party of the protocol. Our compiler combines a variety of cryptographic techniques and primitives to produce compact message formats and fast processing.

The compiler consists of about 6000 lines of $F\#$. The trusted libraries for networking, cryptographic primitives, and principals shared by all session implementations have 780 lines of code (although their concrete implementation mostly relies on much-larger system libraries).

5.5. 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. Jocaml is avaible at http://jocaml.inria.fr. The current version is 3.12.0 (released in September [33]).

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.

5.6. 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.

Hevea (first release in 1997) is still maintained and developed by Luc Maranget. A continuous (although informal) collaboration exists around Hevea, including Philip H. Viton (Ohio State University) for the Windows port and Ralf Treinen (ENS Cachan) for all Debian developments. For the record, Hevea consists in about 20000 lines of Ocaml and about 5000 lines of packages sources written in “almost TeX” (the language understood by Hevea).

This year saw a few developpements around Hevea, mostly for maintenance.

6. New Results

6.1. Weak Memory Models

Participants: Jade Alglave, Luc Maranget, Francesco Zappa Nardelli.

Multiprocessors are now dominant, but real multiprocessors do not provide the sequentially consistent memory that is assumed by most work on semantics and verification. Instead, they have subtle relaxed (or weak) memory models, usually described only in ambiguous prose, leading to widespread confusion.

We developed rigourous and accurate semantics for multiprocessor programs above three architectures: x86, Power, and ARM. Each covers the relaxed memory model, instruction semantics, and instruction decoding, for fragments of the instruction sets, and is mechanised in HOL or Coq.

A paper resuming our understanding of the x86 memory model appeared in the Communications of the ACM [22]. Pankaj Pawan, during his 2 month long internship in Moscova, implement a devilish emulator of x86-TSO. See also a survey at http://moscova.inria.fr/~zappa/projects/weakmemory.

Main of our research effort in 2010 was related with the achievement of Jade Alglave’s PhD and the design and implementation of corresponding tools. All the axiomatic definition of weak memory models covers both the x86 and the more relaxed Power model. These definitions are stated in terms of events structures, which capture the causality of reads and writes in memory.

We extended the Diy tool suite (http://diy.inria.fr).

This tool suite has three components:

- Litmus: this is a tool to run small tests in PowerPC or x86 assembly code against real hardware. It collects the memory states observed during the execution of these tests, allowing to observe which optimisations are enabled by the hardware. A tool paper about Litmus has been submitted to TACAS 2011.
- Diy: this is a tool to automatically generate such small tests from concise specifications. The strength of this tool lies in the fact that it allows to generate tests highlighting a particular optimisation of a processor. Thus, we can precisely state what a processor optimises or not. We used this tool to establish an experimentally sound model for the PowerPC architecture (published in CAV 2010), which is the subtler existing architecture.
- Dont: this is an automated front-end, addressing two issues (a paper about this automated tool will be presented at TACAS'2011):
  - conformance of a machine to a given memory model: the tool automatically generally the (provably) exact set of tests to run to ensure that a given piece of hardware conforms to a given memory model (i.e. that the model embraces indeed all the behaviours observable on the hardware)/
  - automatic exploration of a machine: the tool learns, by incremental testing, the weak memory model exhibited by a given piece of hardware. This allows to do model exploration without knowing a priori the model of the machine, as in the PowerPC case.

In addition, we have implemented all the proofs described in Jade Alglave's thesis in the Coq proof assistant (see the development at http://moscova.inria.fr/~alglave/wmm).

6.2. Relaxed-Memory Concurrency and Verified Compilation

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

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 and validated with correctness proofs a certifying compiler from ClightTSO to x86. A paper describing our approach has been accepted in POPL'2011 [30], while the development is available from http://www.cl.cam.ac.uk/~pes20/CompCertTSO.

6.3. Models of Audit Logs

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

In an optimistic approach to security, one can often simplify protocol design by relying on audit logs, which can be analyzed a posteriori. Such auditing is widely used in practice, but no formal studies guarantee that the log information suffices to reconstruct past runs of the protocol, to reliably detect, and provide evidence of, any cheating.

In 2009 we developed a generic setup for auditability, and we understood how type-checking can be used to check auditability.

In 2010 research focused on extending the F7 type-checker for $F\#$, so that it could easily deal with programs that use cryptography and recursive data structures. Applications such as X.509 certificate chains, secure logs for multi-party games, and XML digital signatures are beyond the reach of automated cryptographic verifiers such as ProVerif, since they require some form of induction. They can be verified using refinement types (that is, types with embedded logical formulas, tracking security events). However, this entails replicating higher-order library functions and annotating each instance with its own logical pre- and post-conditions. Instead, we equip higher-order functions with precise, yet reusable types that can refer to the pre- and post-conditions of their functional arguments, using generic logical predicates. We implemented our method by extending the F7 typechecker with automated support for these predicates and we evaluated our approach experimentally by verifying a series of security libraries and protocols. Our results have been published in FCS-PrivMod [32] and APLAS [27].

Nataliya Guts completed her thesis [21] on "Auditability for security protocols"; the defense is planned for January 2011.

6.4. Compiling Information Flow Security to minimal Trusting Computing Bases

Participants: Cédric Fournet [MSR Cambridge], Jérémy Planul.

Modern computer architectures provide hardware support for reducing dependencies upon Trusting Computing Bases (TCB) and for protecting privileged operations. We present a security model for these mechanisms in an imperative language with dynamic loading of code of programs to make more secure. Then, on top of a preceeding work on the Cflow (Cryptographic Implementations of Information-Flow Security) compiler (see last year), we define program transformations to simulate trusted hosts on untrusted machines by exploiting these hardware security mechanisms available on modern architectures such as trusted platform modules (TPM). Thus, we greatly reduce needed trust assumptions for distributed programs.

A prototype compiler has been developed during a 3-month long internship at Microsoft Research laboratory in Cambridge (April-July 2010) and is built on top of the Cflow compiler originally developed by Gurvan Le Guernic (see last year).

This work will be presented at ESOP'2011 (Saarbrücken).

6.5. Modular Verification of Security Protocol Code by Typing

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

We propose a method for verifying the security of 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 present a theory to justify the soundness of modular code verification via our method.

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.

We evaluate the method on a series of larger case studies of protocol code, previously checked using whole-program analyses based on ProVerif, a leading verifier for cryptographic protocols. Our results indicate that compositional verification by typechecking with refinement types is more scalable than the best domainspecific analysis currently available for cryptographic code.

6.6. Security Programming with Refinement Types & Mobile Proofs

Participants: Nikhil Swamy [MSR Redmond], Juan Chen [MSR Redmond], Cédric Fournet [MSR Cambridge], Karthikeyan Bhargavan, Jean Yang [MIT].

Several recent papers develop lightweight refinement type systems for ML-like languages and use them to verify the security of distributed programs. In these languages, logical formulas flexibly express policies for confidentiality, integrity, and access control. Typechecking can then automatically verify programs meant to enforce these policies; this involves constructing (or checking) a proof for each of their logical refinement obligations.

These type systems offer diverse flavors of erasure for refinements and their proofs. In source programs, refinement formulas may be weakened by subtyping, to enable code reuse in less demanding contexts. After typechecking, most proofs may be erased for efficiency. Others may be replaced with cryptographic evidence. Still, some proofs must be kept at runtime, so that they may be communicated, stored, and re-checked by other programs.

We propose FERN, a core typed lambda-calculus that support programming with refinement types, security proofs and erasure. FERN enables powerful new security programming idioms, and facilitates the side-by-side comparison of existing flavors of refinement types and erasure. We prove type soundness and logical consistency, by embedding FERN formulas into CiC. We also provide a typed embedding of F7 into FERN.

We have implemented a compiler and runtime system for FERN. In addition to programs that use the new features of FERN, our compiler fully supports both F7 and Fine source code. FERN has runtime support for mobile proofs, thereby enabling refinement type-safe distributed programming. Selective erasure also produces efficiency gains—our compiler produces verifiable binaries with 60% code size overhead for proofs and types, as much as a 45x improvement over prior compilers, while still enabling efficient bytecode verification.

We evaluate FERN on several applications, including, among others: data provenance tracking, illustrating the use of proofs at run-time with a custom proof kernel; verification of higher-order libraries and their use in the construction of secure implementations of a variety of cryptographic protocols, including zero-knowledge privacy schemes and secure multi-party sessions; and several prior benchmarks, for a quantitative assessment of proofs at runtime.

This paper has been submitted to PLDI'2011.

6.7. Secure Sessions

Participants: Karthikeyan Bhargavan, Ricardo Corin [Universidad Nacional de Cordoba, Argentina], Pierre-Malo Deniélou [Imperial College, London].

Distributed applications can be structured as parties that exchange messages according to some pre-arranged communication patterns. These sessions (or contracts, or protocols) simplify distributed programming: when coding a role for a given session, each party just has to follow the intended message flow, under the assumption that the other parties are also compliant.

In an adversarial setting, remote parties may not be trusted to play their role. Hence, defensive implementations also have to monitor one another, in order to detect any deviation from the assigned roles of a session. This task involves low-level coding below session abstractions, thus giving up most of their benefits.

We explore language-based support for sessions. We extend the ML language with session types that express flows of messages between roles, such that well-typed programs always play their roles. We compile session type declarations to cryptographic communication protocols that can shield programs from any low-level attempt by coalitions of remote peers to deviate from their roles. Our main result is that, when reasoning about programs that use our session implementation, one can safely assume that all session peers comply with their roles—without trusting their remote implementations.

Initial work was presented at CSF'07 [7], TGC'07 [8] and in a special issue of the Journal of Computer Security [10].

We have added support for integrity and secrecy support for a global store, and dynamic principal selection, which enables simple, abstract reasoning on global control and data flows. In this setting, we developed novel, mostly-automated security proof techniques, where our compiler generates type annotations (from a predicate logic), which are then mechanically checked against actual executable code.

This work was presented in [6]

6.8. Integrating Typed and Untyped Code in a Scripting Language

Participants: Sylvain Lebresne [Purdue U.], Johan Ostlund [Purdue U.], Vitek Vitek [Purdue U.], Tobias Wrigstad [Purdue U.], Francesco Zappa Nardelli.

Many large software systems originate from untyped scripting language code. While good for initial development, the lack of static type annotations can impact code-quality and performance in the long run.

In collaboration with Jan Vitek and Tobias Wrigstad (Purdue University), we studied an approach for integrating untyped code and typed code in the same system to allow an initial prototype to smoothly evolve into an efficient and robust program. We introduced like types, a novel intermediate point between dynamic and static typing. Occurrences of like types variables are checked statically within their scope but, as they may be bound to dynamic values, their usage is checked dynamically. Thus like types provide some of the benefits of static typing without decreasing the expressiveness of the language.

We provided a formal account of like types in a core object calculus and evaluated their applicability in the context of a new scripting language.

A paper describing our approach appeared in POPL'2010 [29], while the implementation in the Thorn programming language (developed by Purdue University and IBM) is available from http://www.thorn-lang.org.

7. Contracts and Grants with Industry

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

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.

The five main challenges to be addressed by this project can be summarized as follows.

1. Design a new collaborative security specification framework. Security specifications for large applications are either non-existent or monolithic (developed by a single security expert.) Although collaborative software development has become the norm for application development, there is no existing framework that enables developers to collaboratively specify the security of their code. Our first goal is to design a specification language that can be used to annotate the security-critical components of source programs. The language must be expressive enough to capture precise security assumptions for trusted libraries and precise security requirements for untrusted libraries and application code. Our second goal is to implement a source code specification framework where collaborating programmers can add and modify security specifications collaboratively and incrementally while keeping their specifications consistent.
2. Design and implement new incremental and modular security verification techniques. The second challenge is to implement new verification tools that can prove that a source program meets its collaborative security specification as both program and specification evolve. General program verification techniques are unsuitable for this purpose since they typically require the whole program and its specification to be verified together and hence do not scale to large dynamic programs. We require that the verification techniques be modular, that is, that it be possible for a single developer to verify a new module without having to verify the full application. We require that the verification techniques be incremental, that is, that it be possible to use previous verification result to avoid re-verifying a particular component unless a change in code or specification makes it likely to fail.

3. Design and implement tools to extract security-critical components Large parts of real applications are irrelevant for security. Writing security specifications for these parts is unnecessary and a waste of effort. Moreover, by eliminating these parts, we can improve the scalability of our verification techniques and simplify the program that needs to be analyzed. Hence, the third challenge of this proposal is to develop techniques that can automatically identify the security-critical components of programs. As proof of correctness for these techniques, we require a theorem that states that any attack on the source program can be reduced to an attack on one of these security-critical components.
4. Build the first verified open source cryptographic protocol library. Most secure networked applications are based upon implementations of standard cryptographic protocols such as TLS, SSH, IPSec, or Kerberos. Although the security of all such applications depends upon the correctness of these protocol implementations, there is no protocol library for which we have a proof of correctness. Indeed, implementation flaws are often found even in widely-distributed and thoroughly-vetted libraries, such as OpenSSL2. Our fourth challenge is to build the first cryptographic protocol library with a formal proof of security. Such a library is a necessary component for building any secure web application. It is also an excellent test case for collaborative specification and incremental verification, since each protocol implementation often has multiple modes and versions implemented by different developers. Moreover, implementing multiple protocols within the same library gives us the opportunity to share code and specifications across protocols.
5. Build the first web applications with formal security proofs. Our final challenge is the culmination of the goals of this proposal. We aim to use our specification framework, verification tools, and verified libraries to build and verify secure web applications. A web application introduces new challenges for the scalability of our verification tools. Browsers and web servers are very large programs and are typically impossible to verify, but the security of an application should depend only on a small part of these programs. They are therefore a good test case for our techniques for extracting security-critical components from web applications.

This grant was awarded by the ERC under its 2010 Starting Independent Researcher Grant scheme. The amount awarded was 1.4 M Euros over 5 years, beginnind in November 2009. We aim to hire 4 Ph.D. students, 2 Post-docs, and several interns over the course of the project.

7.2. Grants with Industry

see section on Joint Centre with Microsoft Research.

7.3. National Initiatives

7.3.1. PARSEC

The project PARCSEC, started at end of 2006, is finishing at end 2010. It is a project funded by the ANR (*Agence Nationale de la Recherche*), together with MIMOSA, EVEREST, LANDE project-teams of INRIA and the team of Roberto Amadio at CNRS-PPS, U. of Paris 7. This project is coordinated by G. Boudol. This project is about the design of programming languages for distributed applications and their security properties.

7.4. European actions

7.4.1. Collaboration with Microsoft

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), Benjamin Grégoire and Santiago Zanella (who defended his PhD in December 2010). 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 2010.

7.5. International Initiatives

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

8. Dissemination

8.1. Animation of the scientific community

+ K. Bhargavan and J. Leifer started a new MSR-INRIA Security Seminar Series in Nov 2010 to comprise talks by high profile speakers whose research directly impacts real systems: the hacking of significant protocols, the design and construction of security infrastructure, and the security proofs of software and hardware. Talks include: "Strategic Healthcare Advanced Research Projects for Security" by Carl Gunter (University of Illinois), "Attacking ballot secrecy in Helios" by Ben Smyth (University of Birmingham), and "Baaz – A System for Detecting Access Control Misconfigurations" by Prasad Naldurg (Microsoft Research India) for details).
+ J. Leifer was a Program Committee member for JFLA 2011, Journées francophones des langages applicatifs, which will be held 29 Jan-1 Feb 2011 in La Bresse, France.
+ J.-J. Lévy is director of the Microsoft Research-INRIA Joint Centre, see http://www.msr-inria.inria.fr. He participated to the renewal of the Microsoft Research-INRIA agreement for next 4 years. In 2010, he participated to the renewal and creation of new projects of so-called Track B (3 renewals + creation of 2 new projects: AzureBrain by G. Antoniu and B. Thirion and CardiacIndex by A. Crimisini and N. Ayache).
+ 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.
+ J.-J. Lévy participated to the jury of the PhD of Pierre-Malo Deniélou (Paris 7, January 25), Jade Alglave (Paris 7, November 26), Gustavo Petri (Sophia-Antipolis, November 29), Santiago Zanella (ENSMP, December 9).
+ L. Maranget is elected member of *Comité technique paritaire* of INRIA, 1 meeting every 2 months about the general politics of INRIA.
+ F. Zappa Nardelli is member of the *Comité Directeur* of the CEA-EDF-INRIA summer schools.
+ F. Zappa Nardelli is the correspondent of the ANR ParSec Project for the Moscova project-team.
+ F. Zappa Nardelli is the correspondent of the *Équipes associées MM*.
+ F. Zappa Nardelli is member of the *Comité de Suivi Doctoral* of INRIA Saclay–Île-de-France.
+ F. Zappa Nardelli served in the recruiting committee of a *maîtres de conférences* position at PPS, Université Denis Diderot Paris 7.
+ F. Zappa Nardelli served in the PhD Jury of Eleonora Sibilio, University of Verona, Italy.
+ F. Zappa Nardelli served in the PhD Jury of Serguei Lenglet, Université de Grenoble.

8.2. Teaching

Our project-team participates to the following courses:

- K. Bhargavan was Chargé d'Enseignement at École polytechnique, teaching undergraduate courses on programming languages (2009-2010).
- October, K. Bhargavan was a visiting lecturer at the Indian Institute of Technology, New Delhi. He taught a four-week course on "Formal Security Analysis of Cryptographic Protocol Code".
- "Concurrency", Master Parisien de Recherche en informatique (MPRI), 2010-2011, at U. of Paris 7, 23 students, (J. Leifer taught the pi-calculus semantics: 15 hours plus the mid-term exam)
- "Concurrency", Master Parisien de Recherche en informatique (MPRI), 2009-2010, at U. of Paris 7, 30 students, (F. Zappa Nardelli taught Proof Methods for Concurrent Programs: 12 hours and the final examination);
- June, 7-11, F. Zappa Nardelli was teaching assistant at the *Introduction to the Coq proof assistant*, CEA-EDF-INRIA summer school (20 hours, about 30 students), Paris.
- March 22-26, F. Zappa Nardelli lectured about Weak-Memory Models at the ECOOP summer school (2 hours, about 30 attendants), Istanbul.
- March-September, F. Zappa Nardelli supervised the 6 months long M1 internship of Yinjie XU (ENS Ulm) on Automatic Test Generation for ML.
- J.-J. Lévy participated to the 2nd Asian-Pacific Summer School on Formal Methods (23-24 August). He taught "Type free lambda-calculus" and extended these lectures to a Master Course, August-September, Tsinghua University, Beijing (15 students, 12h) http://jeanjacqueslevy.net/courses/tsinghua/lambda.

We also had the following activity related to teaching:

- L. Maranget coordinates the 3 computer science problems (4h+2h+2h) of the entrance examination at the Ecole polytechnique in 2010.
- April 26, J.-J. Lévy gave a talk on *Un petit bogue, un grand boum* for students of lycée at Science Ouverte (invited by François Gaudel), Institut Henri Poincaré, Paris.

8.3. Participations to conferences, Seminars, Invitations

8.3.1. Participations to conferences

- January, 29, J.-J. Lévy, F. Zappa Nardelli attended the ANR ParSec meeting in Paris.
- January 19-23, J. Alglave, N. Guts, J.-J. Lévy, L. Maranget attended the POPL conference in Madrid (Spain). F. Zappa Nardelli presented his paper.
- January, K. Bhargavan gave a lecture at the INRIA Rocquencourt Gallium seminar.
- March, 25-27, F. Zappa Nardelli visited Cambridge University for collaboration with Peter Sewell and Suresh Jagannathan.
- March, K. Bhargavan gave a lecture at the ENS Cachan security protocols groupe de travail.
- April 12-16, N. Guts, J.-J. Lévy attended the 37th CosyProofs (Computational and Symbolic Proofs of Security) School on theoretical computer science and French-Japanese collaboration workshop), Barbizon.
- May, 5-6, F. Zappa Nardelli visited the University of Verona for collaboration with Massimo Merro.
- May, 25, N.Guts attended the CryptoForma workshop in Paris. She gave a talk on Pre- and Post-conditions for Security Typechecking.

- May, 28, F. Zappa Nardelli gave a talk on *Ma mémoire est faible* at the "Unithé ou Café" meeting at INRIA Saclay.
- June, 22-25, F. Zappa Nardelli attended the ECOOP conference in Maribor, Slovenia. He lectured at the ECOOP summer school.
- June 28-29, N. Guts, F. Zappa Nardelli and Pankaj Pawan attended the ANR ParSec meeting in Sophia-Antipolis. N. Guts gave a talk on Type Inference for F7. Pankaj Pawan gave a talk on its prototype of a devilish emulator for x86TSO.
- June 30, F. Zappa Nardelli attended the Multicore day in Paris.
- July 9-21, J. Alglave, N. Guts, J. Leifer, J.-J. Lévy, L. Maranget attended the FLOC conference in Edinburgh. J. Alglave presented her paper at CAV (co-authored with L. Maranget). N. Guts gave a talk on Pre- and Post-conditions for Security Typechecking at FCS PrivMod workshop.
- September 15-16, J. Leifer attended as an invited speaker SAS 2010, the 17th International Static Analysis Symposium, Perpignan, France, September 14-16, 2010. He gave a talk “Remembering Robin...and a short introduction to secure sessions” at a special session in memory of Robin Milner and Amir Pnueli.
- September, J. Alglave visited Aquinas Hobor during 2 weeks at the U. of Singapore. She presented the Diy tool suite, and worked on with the design of a logic to reason about racy programs with weak memory models.
- November, J. Alglave and L. Maranget visited U. of Cambridge (Peter Sewell, Susmit Sarkar, Derek Williams [engineer at IBM]).
- November 28, December 1, K. Bhargavan presented a paper at APLAS 2010, Shanghai.
- December 12-14, F. Zappa Nardelli visited U. of Cambridge.

8.3.2. Seminars

- January 26, J.-J. Lévy talked about “Un petit bogue, un grand boum!” at Ecole Normale Supérieure, Paris. http://www.diffusion.ens.fr/index.php?res=conf&idconf=2737#, vidéo en http://www.diffusion.ens.fr/video_inc.php?video=2010_01_26_levy.mov
- February 9, J.-J. Lévy talked about “Preuves de programmes et méthodes formelles” at the Microsoft TechDays, Paris.
- September 15, J.-J. Lévy talked about “the MSR-INRIA Joint Centre” at Microsoft Research Asia, Beijing.
- December 3, J.-J. Lévy talked about “Proofs, Security, and Computational Sciences” at the MSR Joint Institutes Workshop, affiliated to Merging Knowledge, 5th Anniversary, Cosbi, Trento.

8.3.3. Invited visitors

- February 17-19, Peter Sewell, Suresh Jagannathan, and Jaroslav Ševčík visited Rocquencourt for collaboration with F. Zappa Nardelli.
- June-July, Pankaj Pawan (IIT Kanpur) was intern student in the Moscova project team under the supervision of F. Zappa Nardelli.
- May-September, Yingjie Xu (ENS Ulm) was intern student in the Moscova project team under the supervision of F. Zappa Nardelli.
- September, 21-24, Jaroslav Ševčík visited Rocquencourt for collaboration with F. Zappa Nardelli.
- November, 14-16, M. Batty (University of Cambridge) visited Rocquencourt for collaboration with F. Zappa Nardelli. He gave a talk on the C++ memory model.
- November 22-December 17, Carl Gunter (University of Illinois) visited Rocquencourt for collaboration with K. Bhargavan.

- November, 24-27, Peter Sewell visited Rocquencourt for collaboration with F. Zappa Nardelli and serving in J. Alglave PhD Jury.

9. 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] J. ALGLAVE. *A Shared Memory Poetics*, Université Denis Diderot Paris 7, December 2010.

[20] P.-M. DENIÉLOU. *Sûreté des abstractions et Sessions sécurisées dans les langages distribués*, Université Denis Diderot Paris 7, January 2010.

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

Articles in International Peer-Reviewed Journal

[22] P. SEWELL, S. SARKAR, S. OWENS, F. ZAPPA NARDELLI, M. O. MYREEN. *x86-TSO: A Rigorous and Usable Programmer's Model for x86 Multiprocessors*, in "Communications of the ACM", July 2010, vol. 53, n^o 7, p. 89–97, (Research Highlights).

[23] P. SEWELL, F. ZAPPA NARDELLI, S. OWENS, G. PESKINE, T. RIDGE, S. SARKAR, R. STRNIŠA. *Ott: Effective Tool Support for the Working Semanticist*, in "Journal of Functional Programming", January 2010, vol. 20, n^o 1, p. 70–122, Invited submission from ICFP 2007.

International Peer-Reviewed Conference/Proceedings

[24] J. ALGLAVE, L. MARANGET, S. SARKAR, P. SEWELL. *Fences in Weak Memory Models*, in "CAV", 2010.

[25] J. ALGLAVE, L. MARANGET, S. SARKAR, P. SEWELL. *Litmus: Running Tests Against Hardware*, in "TACAS", 2011.

[26] K. BHARGAVAN, C. FOURNET, A. D. GORDON. *Modular Verification of Security Protocol Code by Typing*, in "ACM Symposium on Principles of Programming Languages (POPL'10)", 2010, p. 445–456.

[27] K. BHARGAVAN, C. FOURNET, N. GUTS. *Typechecking Higher-Order Security Libraries*, in "Proc. APLAS", Lecture Notes in Computer Science, Springer-Verlag, 2010.

[28] C. FOURNET, J. PLANUL. *Compiling Information Flow Security to minimal Trusting Computing Bases*, in "European Symposium on Programming (ESOP'11)", 2011.

[29] T. WRIGSTAD, F. ZAPPA NARDELLI, S. LEBRESNE, J. OSTLUND, J. VITEK. *Integrating Typed and Untyped Code in a Scripting Language*, in "POPL", 2010, p. 377-388.

[30] J. ŠEVČÍK, V. VAFEIADIS, F. ZAPPA NARDELLI, P. SEWELL, S. JAGANNATHAN. *Relaxed-Memory Concurrency and Verified Compilation*, in "POPL 2011", 2011, to appear.

Other Publications

[31] J. ALGLAVE, L. MARANGET, S. SARKAR, P. SEWELL. *diy, release 3.0*, October 2010, Software and documentation available at, http://diy.inria.fr/.

[32] K. BHARGAVAN, C. FOURNET, N. GUTS. *Pre- and Post-Conditions for Security Typechecking*, in "Workshop on Foundations of Security and Privacy (FCS-PrivMod 2010)", 2010.

[33] L. MARANGET, L. MANDEL, M. QIN. *JoCaml release 3.12.0*, September 2010, http://jocaml.inria.fr/.

References in notes

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

[35] 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.

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