Information and Computation 238 (2014) 68–105

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Typing access control and secure information flow in sessions ☆

Sara Capecchi [a], Ilaria Castellani [b,*], Mariangiola Dezani-Ciancaglini [a]

[a] Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino, Italy
[b] INRIA, 2004 route des Lucioles, 06902 Sophia Antipolis, France

ARTICLE INFO

Article history:
Received 10 March 2011
Available online 15 July 2014

Keywords:
Communication-centred computing
Session types
Access control
Secure information flow

ABSTRACT

We consider a calculus for multiparty sessions with delegation, enriched with security levels for session participants and data. We propose a type system that guarantees both session safety and a form of access control. Moreover, this type system ensures secure information flow, including controlled forms of declassification. In particular, it prevents information leaks due to the specific control constructs of the calculus, such as session opening, selection, branching and delegation. We illustrate the use of our type system with a number of examples, which reveal an interesting interplay between the constraints of security type systems and those used in session types to ensure properties like communication safety and session fidelity.

1. Introduction

With the advent of web technologies and the proliferation of programmable and interconnectable devices, we are faced today with a powerful and heterogeneous computing environment. This environment is inherently parallel and distributed and, unlike previous computing environments, it heavily relies on communication. It therefore calls for a new programming paradigm which is sometimes called *communication-centred*. Moreover, since computations take place concurrently in all kinds of different devices, controlled by parties which possibly do not trust each other, security properties such as the confidentiality and integrity of data become of crucial importance. The issue is then to develop models, as well as programming abstractions and methodologies, to be able to exploit the rich potential of this new computing environment, while making sure that we can contain its complexity and get around its security vulnerabilities. To this end, calculi and languages for communication-centred programming have to be *security-minded* from their very conception, and make use of specifications not only for data structures, but also for communication interfaces and for security properties.

The aim of this paper is to investigate type systems for safe and secure sessions. A *session* is an abstraction for various forms of "structured communication" that may occur in a parallel and distributed computing environment. Examples of sessions are a client-service negotiation, a financial transaction, or a multiparty interaction among different services within a web application.

Language-based support for sessions has now become the subject of active research. Primitives for enabling programmers to code sessions in a flexible way, as well as type systems ensuring the compliance of programs to session specifications (session types), have been studied in a variety of calculi and languages in the last decade. *Session types* were originally

☆ Work partially funded by ASCENS 257414 Project, by ICT COST Action IC1201 BETTY, MIUR PRIN Project CINA Prot. 2010LHT4KM and Torino University/Compagnia San Paolo Project SALT.

* Corresponding author.

http://dx.doi.org/10.1016/j.ic.2014.07.005
0890-5401/

introduced in a variant of the pi-calculus [29]. We refer to [12] for a survey on the session type literature. The key properties ensured by session types are *communication safety*, namely the consistency of the communication patterns exhibited by the partners (implying the absence of communication errors), and *session fidelity*, ensuring that channels which carry messages of different types do it in a specific order.

Enforcement of security properties via session types has been studied in [3,24]. These papers propose a compiler which, given a multiparty session description, implements cryptographic protocols that guarantee *session execution integrity*. A type system assuring that services always comply with the policy regulating data exchange is presented in [20]. The question of ensuring *access control* in binary sessions has been recently addressed in [19] for the Calculus of Services with Pipelines and Sessions of [4], where delegation is absent. On the other hand, the property of *secure information flow* has not been investigated within session calculi so far. This property, first studied in the early eighties [14], has regained interest in the last decade, due to the evolution of the computing environment. It has now been thoroughly studied for both programming languages (cf. [25] for a review) and process calculi [13,15,18].

In this paper, we address the question of incorporating mandatory access control and secure information flow within session types. We consider a calculus for multiparty sessions with delegation, enriched with security levels for both session participants and data, and providing a form of declassification for data [27], as required by most practical applications. We propose a type system that ensures access control, namely that each participant receives data of security level less than or equal to its own. For instance, in a well-typed session involving a Customer, a Seller and a Bank, the secret credit card number of the Customer will be communicated to the Bank, but not to the Seller. Moreover, our type system prevents insecure flows that could occur via the specific constructs of the language, such as session opening, selection, branching and delegation. Finally, we show that it allows controlled forms of declassification during communication, namely those explicitly permitted by the sender and respecting the access control policy in the receiver. Our work reveals an interesting interplay between the constraints of security type systems and those used in session types to ensure properties like communication safety and session fidelity.

The rest of the paper is organised as follows. Section 2 motivates our access control and declassification policies with an example. Sections 3 and 4 introduce the syntax and semantics of our calculus. Section 5 defines the secure information flow property. Section 6 illustrates this property by means of examples. Section 7 presents our type system and Section 8 establishes its soundness. Section 9 discusses future work. Appendix A discusses a more expressive notion of security (called *robust security*) and shows that our type system remains sound for this new notion. Appendix B summarises various features of the running example introduced in Section 2.

This paper is a deeply revised version of [9], with improved definitions, new examples and results and complete proofs. In a companion paper [8] we give a monitored semantics, which only assures secure information flow for a simplification of the present calculus without declassification and delegation.

2. An example on access control and declassification

In this section we illustrate by an example the basic features of our typed calculus, as well as our access control policy and our specific form of declassification. The question of secure information flow will only be briefly touched upon here, while it will be discussed in depth in Sections 5 and 6, once the semantics of the calculus has been formally defined. Suffice it to say, for the time being, that a process has secure information flow if none of its actions depends on some *input action* of higher or incomparable level.

For the sake of simplicity, in this introductory example we shall only consider binary sessions, and two security levels for data and participants, namely high (secret) and low (public).[1] In our calculus, the level of data represents their degree of confidentiality, while the level of participants represents their reading rights: thus a high participant can read both high and low data, whereas a low participant can only read low data.

Consider the following scenario, involving a bookseller, a client and a bank. A client C sends the title of a book to a bookseller S. The seller S *delegates* to a bank B both the reception of C's credit card number and its validity check. This frequently happens in online shopping, when clients are redirected to a new page for a secure payment. Delegation is crucial for assuring the secrecy of the credit card number, which should be read by B but not by S. To express this access control constraint, the typing will assign high security level to the credit card and to B, and low security level to S. Then B notifies S about the result of the validity check: since this result is low and the card is high, for this information flow to be admissible a *declassification* is needed. Finally, if the credit card is valid, C receives a delivery date from S, otherwise the deal falls through. More precisely, the protocol is as follows:

1. C opens a connection with S and sends a title to S;
2. S opens a connection with B and *delegates* to B part of his conversation with C;
3. C sends his secret credit card number *apparently* to the low party S but *really* – thanks to delegation – to the high party B; moreover C authorises the high recipient of his card number to declassify it (although not knowing B, C knows that a delegation to a high party must take place since S is low);

[1] However, our formal treatment will cover the general case of a finite lattice of security levels, and a number of examples involving multiparty sessions will be discussed in Section 6.