Network of Excellence

Deliverable D9.2

Initial Solutions for Security Assurance for Services
Abstract

The main objective of this work package is to enable assurance in the development of software-based services in order to increase confidence in their security. The core goal is to incept a transverse methodology that enables to manage assurance throughout the software development life cycle (SDLC). Our research is divided into two main sub-domains: early assurance at the level of requirements, architecture and design using techniques such as refinement and model checking and complementary implementation-based assurance techniques such as testing and runtime verification.

This deliverable summarizes the results obtained during the first year of the project. We cover most of the tasks and activities of the work package and address the majority of the challenges set out in the preceding deliverable about the state-of-the-art in assurance for services. Building on this state-of-the-art, we have obtained a wide range of strong results and we were able to produce a number of high-rated publications, most notably in the areas of refinement, model checking, testing and debugging, and runtime verification.

Keyword List

Assurance for services; software verification, refinement, model checking; testing, penetration testing, model-based testing, run-time verification, usage control; security metrics.
## Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Type of Change</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Initial setup</td>
<td>Christoph Sprenger, ETH Zurich</td>
</tr>
<tr>
<td>1.0</td>
<td>Complete draft</td>
<td>Tigran Avanesov, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Marianne Busch, LMU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fabrice Bouquet, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Juan Caballero, IMDEA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Frédéric Dadeau, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simone Frau, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Thai Son Hoang, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Felix Klaedtke, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nora Koch, LMU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eda Marchetti, CNR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Michael Rusinowitch, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Christoph Sprenger, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mohammad Torabi Dashti, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Artsiom Yautsiukhin, CNR</td>
</tr>
<tr>
<td>2.0</td>
<td>Under internal review</td>
<td>Tigran Avanesov, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Marianne Busch, LMU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fabrice Bouquet, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Juan Caballero, IMDEA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Frédéric Dadeau, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simone Frau, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Thai Son Hoang, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Felix Klaedtke, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nora Koch, LMU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eda Marchetti, CNR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Michael Rusinowitch, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Christoph Sprenger, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mohammad Torabi Dashti, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Artsiom Yautsiukhin, CNR</td>
</tr>
<tr>
<td>2.1</td>
<td>Revised version</td>
<td>Tigran Avanesov, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Marianne Busch, LMU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fabrice Bouquet, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Juan Caballero, IMDEA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Frédéric Dadeau, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nora Koch, LMU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eda Marchetti, CNR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Michael Rusinowitch, INRIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Christoph Sprenger, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mohammad Torabi Dashti, ETH Zurich</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Artsiom Yautsiukhin, CNR</td>
</tr>
<tr>
<td>3.0</td>
<td>Approved by the internal reviewers</td>
<td>Christoph Sprenger, ETH Zurich</td>
</tr>
</tbody>
</table>
Document Review

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Reviewer</th>
<th>Comment</th>
</tr>
</thead>
</table>
| Sep 22, 2011    | 2.0     | Marinella Petrocchi (CNR) | - Non-uniform presentation style throughout the document.  
- Found Section 3.1.2 a bit too technical for non-experts.                                                                                                                                             |
| Sep 23, 2011    | 2.0     | Bjørnar Solhaug (SINTEF)  | - Presentation somewhat verbose and hard to follow in some passages, e.g. Section 3.1.1 is very compressed, as is also Section 3.1.2.  
- Introduction lacks overall description of how the various contributions fit together or complement each other in the SDLC.  
- Every chapter should begin with an overview including one sentence highlighting the contribution. Structure of the chapters and sections should be made more uniform, for example like Section 2.1.1 and 2.3.1 that have a separate “Contributions” section. |

(For the editor: please, list a resume of the most relevant reviewers' comments)
# Table of Contents

**LIST OF FIGURES** ................................................................. 9

**LIST OF TABLES** .................................................................. 11

1 **INTRODUCTION** ............................................................. 13
1.1 Overview of results and highlights ........................................ 13
1.2 Assurance along the SDLC .................................................. 15
1.3 Document structure and NESSoS tasks ................................. 15

2 **EARLY ASSURANCE** ....................................................... 17
2.1 Stepwise refinement .......................................................... 17
  2.1.1 Developing security protocols using classical refinement .......... 17
  2.1.2 Automated analysis of non-interference security by refinement .... 19
2.2 Mapping security-design models to enable formal analysis ............ 19
2.3 Algorithmic verification .................................................... 21
  2.3.1 Modeling multiple intruders and XML rewriting attacks with general deducibility constrains ....................................................... 21
  2.3.2 Integrated specification and verification of security protocols and policies .......................................................... 22

3 **ASSURANCE IN IMPLEMENTATION** ............................. 25
3.1 Testing and debugging ....................................................... 25
  3.1.1 Model-based testing ...................................................... 25
  3.1.2 Testing of access control policies .................................... 29
  3.1.3 Debugging ................................................................. 32
3.2 Run-time verification: Monitoring usage control properties ............ 35
  3.2.1 Interval-based and point-based properties ......................... 35
  3.2.2 Monitoring tool .......................................................... 35
  3.2.3 Monitoring of concurrent and distributed systems .................. 36

4 **SECURITY METRICS** ..................................................... 37
4.1 Formal approach to security metrics. What does “more secure” mean for you? 37
4.2 Formal analysis of security metrics and risk ............................. 38
4.3 A general method for assessment of security in complex services .... 39

5 **INTERACTIONS WITH OTHER WORK PACKAGES** .......... 41

6 **CONCLUSIONS** ............................................................. 43

NESSoS WP9 **BIBLIOGRAPHY** ........................................... 45

**GENERAL BIBLIOGRAPHY** ................................................ 47
List of Figures

Figure 1.1: Assurance techniques and development phases ........................................ 15

Figure 2.1: Address book example. Navigation state diagram.................................... 21

Figure 3.1: Syntax of the sequence and model layers ........................................... 27
Figure 3.2: Syntax of the test generation directive layer ........................................ 27
Figure 3.3: Integration of the MBT and UMLsec approaches..................................... 28

Figure 4.1: Example of transformation of a business process into a graph ...................... 40
List of Tables

Table 1.1: Mapping tasks to sections of this document ....................................................... 15

Table 3.1: Mutant-kill ratios achieved by test suites of Simple Combinatorial and XACML Context Schema-based .......................................................... 31

Table 4.1: Relations between metrics .................................................................................. 38
1 Introduction

The Future Internet is characterized by its highly dynamic and distributed nature. There is a strong tendency to rapidly and dynamically construct new applications and services from a given set of services that is already available. These services may be spread over many locations and involve a large number of stakeholders that do not necessarily trust each other. It is therefore natural that this evolution is paralleled by increasingly stronger security requirements on these applications. These are formulated in the form of security policies, the enforcement of which makes sure that, for example, services, agents, and other entities are properly authenticated and that sensitive data is only accessible to authorized players.

While the assurance of correctness and security properties is already a challenging task for centralized systems, this challenge is aggravated by the highly adaptive and rapidly evolving distributed systems and applications that populate the current and Future Internet. It is therefore indispensable that a suitable collection of rigorous, mathematically founded methods are developed that support the developer at each stage of the software development life cycle (SDLC). This goal is reflected in the task composition of Work Package 9 (WP 9) on Security Assurance for Services.

Rigorous development and verification techniques such as stepwise refinement, theorem proving, and model checking are adopted at the early phases of the SDLC, where they are most cost-effective (Task 9.1). Testing, debugging, and run-time verification methods (Task 9.2), used in the later phases of the SDLC, complement the previously mentioned techniques. Transversal methods (Task 9.3) aim to provide assurance support throughout the development process and improve the integration and communication between the different phases. Finally, security and assurance metrics (Task 9.4) measure the overall security of the software-based services in order to provide developers with a tool to obtain objective feedback on the achieved level of security.

This deliverable reports on initial results obtained in WP 9 during the first year of the NESSoS project. Our results build on the state-of-the-art in security assurance as described in the NESSoS Deliverable ID9.1 [29]. In our gap analysis, we had mapped out a number of specific research directions among which we have identified the following four recurring research challenges.

C1 – Expressiveness We need more expressive modeling languages and attacker models in order to faithfully represent the manifold aspects of Future Internet applications (e.g., for modeling XML data, authorization policies, usage control properties, compromising intruders, new kinds of attacks such as XML rewriting attacks, trust models, and distribution aspects).

C2 – Distribution In order to obtain better coverage and stronger guarantees testing and runtime verification techniques have to consider the entire distributed system instead of the prevalent approach of separately testing and monitoring individual components or services of distributed applications (e.g., client side/server side).

C3 – Linking abstraction levels / SDLC phases The integration between different phases of the SDLC needs to be improved. In particular, the different abstraction levels need to be related in a semantically sound and practically useful way.

C4 – Modularization Modularization is a natural way to decompose complex systems into simpler parts. Unfortunately, security is not compositional, that is, new vulnerabilities may arise from the composition of modules or services, even if each of these is secure individually. Further study is needed to identify sufficient conditions for secure module and service composition.

Our conclusion was that useful basic techniques do exist in many cases, but they require substantial extensions to tackle the distributed and highly dynamic Future Internet applications and their demanding security requirements.

1.1 Overview of results and highlights

Rather than trying to be exhaustive in our choice of topics and challenges we address, we have focused on a number of strategic issues that we believe to be essential for advancing the current state-of-the-art
and for laying a solid foundation for future research in the area. Below we summarize our results from the first year, relate them to the challenges C1-C4 above, and point out some publication highlights.

**Stepwise refinement (Section 2.1)** We propose a new method for the development of security protocols by stepwise refinement. The method is based on a refinement strategy that spans four different levels of abstraction starting from abstract models of the security properties to be established and reaching down to protocols secure against a standard Dolev-Yao adversary. The method is implemented in the Isabelle/HOL theorem prover. This work has resulted in a ACM CCS 2010 publication [26].

Challenges addressed: C3.

**Algorithmic verification (Section 2.3)** We have extended model checking techniques for security protocols along several lines. First, we have studied attacker models that are relevant for Future Internet applications, namely, multiple non-communicating attackers and a class of attacks that arise from the XML representation of messages, called XML rewriting attacks. We give a decision procedure for these cases based on a generalized form of intruder constraints.

Second, we have augmented model checking techniques with new primitives, namely, we have combined security protocol specifications with authorization policies, thus providing a framework for the modeling and automatic verification of service-oriented architectures. This work is reported in a IEEE CSF 2011 publication [17].

Challenges addressed: C1.

**Testing and debugging (Section 3.1)** Differential slicing is an efficient technique for debugging security vulnerabilities. The idea is to compare two execution traces, a correct and a faulty one, extract the reason for their difference, and concisely present it to the program analyst so that he can quickly understand the cause of the faulty behavior. This technique has been applied to analyze crashes of benign programs as well as the triggering mechanisms of malware. We have published this work at IEEE S&P 2011 [19].

In our work on mutation-based penetration testing for security protocols, we generate test cases for security protocol implementations from high-level models of these protocols. We apply mutation operators to these models, check the model for executability, and use a model checker to verify its security properties. Any attack trace on an insecure mutant found in this manner can serve as a test case for the protocol implementation.

Moreover, in order to test XACML access control policies, we have proposed and compared two strategies for the derivation of XACML requests.

Challenges addressed: C3.

**Runtime verification (Section 3.2)** Our work in runtime verification has progressed along three axes. First, we have studied the impact of different time models on monitoring algorithms. In particular, we compare monitoring under point-based vs interval-based semantics of propositional metric temporal logic (MTL). Second, we have proposed a tool for monitoring properties expressed in an expressive safety fragment of first-order MTL. Third, we provide solutions for efficiently monitoring partially ordered event logs with millions of log entries produced per day, which is a central problem in monitoring real-time distributed systems. The work on the latter two axes is published at TIME 2011 [8].

Challenges addressed: C1, C2, and C3.

**Security metrics (Section 4)** We report on preliminary results on a formal framework for the modeling and analysis of security metrics. We have focused on the metrics’ capacity to detect changes in the security level and the comparison and aggregation of different security metrics.

Challenges addressed: C3.

In Section 5, we discuss interactions with other NESSoS work packages. In the final Section 6, we draw conclusions and provide an outlook on future work. We have split the bibliography into two parts. The first part contains our NESSoS WP 9 publications including some that are not discussed in this deliverable. The second part lists other publications cited in this deliverable.
1.2 Assurance along the SDLC

The contributed assurance methods described above are associated with different stages of the SDLC as illustrated in Figure 1.1. Algorithmic verification is (in the present context) associated with the design phase, testing and debugging are related to the implementation phase, and runtime verification is mainly applied to deployed systems. Model-based testing is a transversal form of testing, which relates an implementation to more abstract models that were produced in earlier SDLC phases. Since these techniques relate to different SDLC phases and thus different abstraction levels, they are complementary. This means that the application of each of these techniques has the potential of strictly increasing the degree of security assurance obtained for a system under development.

Two further contributions are transversal methods. The first method is stepwise refinement, which spans several SDLC phases from requirements to implementation. Refinement integrates development and assurance into a single activity that produces a series of models resulting in a system that is correct by construction. Therefore, refinement may be chosen as an alternative to traditional development methods combined with the previously mentioned assurance techniques (in particular, verification and testing). The second method, security metrics, may be used to evaluate the security of artifacts from any SDLC phase.

1.3 Document structure and NESSoS tasks

The mapping of NESSoS tasks listed in the Description of Work to the Sections of this document is given in Table 1.1.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Section/deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Early assurance</td>
<td>2</td>
</tr>
<tr>
<td>9.1.1</td>
<td>Stepwise refinement</td>
<td>2.1</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Mapping security-design models to enable formal analysis</td>
<td>2.2</td>
</tr>
<tr>
<td>9.1.3</td>
<td>Extended model checking</td>
<td>2.3</td>
</tr>
<tr>
<td>9.2</td>
<td>Assurance in implementation</td>
<td>3</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Secure programming</td>
<td>not covered</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Testing and debugging</td>
<td>3.1</td>
</tr>
<tr>
<td>9.2.3</td>
<td>Run-time verification</td>
<td>3.2</td>
</tr>
<tr>
<td>9.3</td>
<td>Transversal methodology</td>
<td></td>
</tr>
<tr>
<td>9.3.1</td>
<td>Model-based testing</td>
<td>3.1 deliverable D7.2</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Bridging model-based and language-based security</td>
<td>3.2</td>
</tr>
<tr>
<td>9.4</td>
<td>Security metrics</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1.1: Mapping tasks to sections of this document
Note that we report on our work in “Model-based testing” (Task 9.3.1) in Section 3.1 on “Testing and Debugging” and Task 9.3.2 on “Bridging model-based and language-based security” is covered in deliverable D7.2 of WP 7 (Design and Architecture).

Task 9.2.1 on “Secure programming” is not covered in this first-year deliverable. We will report on this subtask in a later deliverable. However, deliverable D8.2 reports on work in secure programming involving assurance aspects (see also Section 5).
2 Early Assurance

An early detection of security failures in Future Internet applications reduces development costs and improves assurance in the final system. In this section, we report on our results for early assurance. Section 2.1 covers stepwise refinement, where the accent is set on constructing provably secure systems in a stepwise manner starting from requirements. In Section 2.2, we present formal mappings from security design models to other formalisms for which automated or semi-automated analysis tools are available. Section 2.3 is devoted to extended model checking, where algorithmic verification techniques are adapted to the increasingly expressive specification languages that are required to model secure Future Internet protocols and services.

2.1 Stepwise refinement

2.1.1 Developing security protocols using classical refinement

Designing security protocols is a non-trivial task and therefore an attractive target for formal methods. The vast majority of existing approaches, whether automated or interactive, symbolic or computational, are designed for post-hoc verification of completed protocol designs. This means that all decisions regarding the number of roles, the communication and message structure, and cryptographic primitives must be made before verification begins.

We advocate a development method based on stepwise refinement, where we start from an abstract model that we gradually concretize by incorporating additional design elements (superposition) and by transforming existing ones (data refinement). Organizing developments this way has numerous benefits compared to post-hoc verification approaches. First and foremost, refinement enables us to reason abstractly about the problem and extract its essential features. This abstract analysis leads to valuable insights into the problem. Second, refinement helps us to master the complexity of both models and proofs by focusing on individual design aspects at each step. Third, refinement naturally supports the hierarchical development of protocol families through branching points in the refinement design space. For example, we may prove the secrecy of a session key for key transport protocols in an abstract model that we later refine into more concrete protocols with different communication topologies, message structures, and cryptographic primitives, without repeating the secrecy proof.

Refinement strategy for security protocols  We specify each model by a transition system together with a set of invariants. The initial models constitute an abstract specification of the security properties required of our protocol. Each subsequent model transforms or extends its predecessor and may introduce additional security properties. The correctness of each refinement step is justified by a simulation proof, in the spirit of refinement in Event-B [34]. The final model is a full-fledged cryptographic protocol that is secure against a standard Dolev-Yao intruder [69]. We have formalized and machine-checked all the results described here, namely, our theory of refinement and all protocol developments, using the Isabelle/HOL theorem prover [107].

The following four-level refinement strategy guides our developments, where each level may itself consist of several refinement steps.

Level 0 Abstract, protocol-independent specifications of secrecy and authentication properties. The model's state contains just enough structure to formulate these properties as invariants and realize them abstractly by just a few events. There are no intruder actions.

Level 1 Abstract protocol without message passing. We introduce protocol roles, local states of agents, and basic protocol steps. Agents read data directly from other agents' local state. There are still no intruder actions.

Level 2 Protocol communicating over abstract channels with security properties, such as confidential and authentic channels. The intruder may eavesdrop messages on non-confidential channels and fake messages on non-authentic channels. No cryptography is used.
Level 3  Cryptographic protocol using an insecure channel. The messages on the abstract channels from Level 2 are now implemented using cryptographic messages sent over an insecure channel. A standard Dolev-Yao intruder completely controls the network.

Validation by case studies  In order to validate the effectiveness of our refinement strategy, we developed different authentication and key establishment protocols from abstract specifications. We derived the ISO/IEC 9798-3 protocol and the first two steps of the Needham-Schroeder-Lowe protocol, which are unilateral entity authentication protocols based on signatures and public-key encryption. Here, our initial model expresses authentication in terms of a minimal structure, namely, the view of each role at the end of a protocol run. We also developed various server-based key transport protocols using symmetric-key cryptography, including a variant of the Otway-Rees protocol, the Needham-Schroeder Shared-Key protocol, the Yahalom protocol, and the core of the Kerberos IV and V protocols. For each of these protocols, we refine two initial models to establish both secrecy and authentication properties: the first model is an abstract representation of secrecy that we instantiate with session keys and the second is the entity authentication model mentioned earlier. Interestingly, the final models in each class of protocols refine a common Level 1 ancestor, even though they use different cryptographic primitives and communication patterns.

Contributions  We see four main contributions in our work. The first contribution is methodological. Our initial models specify the security goals of the abstract protocol models at Level 1. These in turn determine the basic structure of entire families of protocols. Using the refinement strategy outlined above, we systematically refine these abstract models in a series of well-defined steps to protocols using cryptographic messages sent over an insecure channel. We illustrate this with known protocols, but we have also used this strategy to develop new variants of these protocols. Our refinement strategy aims at proving security properties at the highest possible abstraction level. General results guarantee that these properties are preserved by subsequent refinements. Our refinement strategy naturally gives rise to straightforward simulation relations for the refinement proofs. Moreover, the process of proving refinements helps us discover invariants, many of which are canonical. For example, the simulation relation linking Levels 2 and 3 usually expresses that the local states of the roles is untouched by the (superposition) refinement and maps the cryptographic messages at Level 3 to abstract channels at Level 2 (data refinement). A canonical invariant that appears in such refinement proofs states that the honest agents' long-term keys remain secret. This is the natural level of abstraction for this invariant. Typically, the other relevant security properties are already proved in earlier refinements.

Our second contribution is to show how to systematically model and use channels with security properties to construct and reason about security protocols. This fundamental abstraction allows us to reason about a protocol's security properties at a lower level of complexity than with the Dolev-Yao intruder. It also enables a range of different realizations. For example, we may implement an authentic channel using signatures or MACs. Moreover, the communication structure may change from Level 2 to 3. For instance, an abstract key server may independently distribute keys on secure channels to the initiator and responder, whereas in the concrete realization the distribution is sequential: one role receives two encrypted copies of the key and forwards one copy to the other role. The abstract view represents the essential structure of server-based key distribution. The forwarding is just an implementation decision.

Our third contribution is to show how refinement can be used to develop protocols that are secure under the standard Dolev-Yao intruder model (at Level 3). In contrast, in related work such as [49, 88, 51, 46], the authors do not continue the refinements down to the level of a standard Dolev-Yao model based on an algebra of cryptographic messages; some use ad-hoc, protocol-dependent intruder definitions. This makes it difficult to compare their models with existing work on protocol modeling and verification and to know for which adversaries their properties hold.

Our final contribution is a comprehensive definitional extension of Isabelle/HOL with a theory of refinement that is based on simulation and borrows elements from [34, 31]. We define an implementation relation on models including a notion of observation, derive proof rules for invariants and refinement, and show that refinement entails implementation.

For more details about this work, we refer the reader to [26] and [27].
2.1.2 Automated analysis of non-interference security by refinement

In [102], Morgan introduces the Shadow semantics for reasoning about non-interference properties for sequential programs. Program variables are partitioned into high- and low-security variables, with the assumption that an adversary may observe the low-security (or visible) variables but can only infer the value of the high-security (or hidden) variables. Shadow semantics ensures that refinements of specifications preserve functionality and, at the same time, keep secrets at least as well as the specification does.

We start by taking the Shadow Semantics for non-interference security [102] and show how its refinement proofs can be automatically generated and proved using the RODIN platform [33]. Our idea is to “encode” Shadow refinement condition using standard refinement in Event-B [32] and subsequently using its supporting RODIN platform for automation. The encoding is based on the operational semantics of Shadow refinement and can be summarized as follows.

- An additional variable (shadow variable) is attached to every program to capture the possible values of the hidden variables at every moment.
- Programs are altered to include the additional shadow variable.
- Shadow refinement are interpreted as a standard refinement preserving additional coupling invariants between shadow variables of a specification and an implementation.

We have implemented a front-end for inputting programs using a simple sequential language, including extra annotations to distinguish between hidden and visible variables. We use the RODIN platform as a “back-end” for generating and discharging (automatically/interactively) the resulting proof obligations.

An assumption of the Shadow refinement underlying the attack model is that the attacker can run the program precisely once, or he cannot retain any memory of the previous experiments if he runs the program several times. This weakness is common amongst the qualitative approaches. On the one hand, many security protocols are designed to work in more hostile and realistic environment with attackers who can track several runs and perform statistical analyses. On the other hand, it is more challenging to reason quantitatively than qualitatively. Subsequently, this endures a greater challenge for automation.

In order to overcome this difficulty, we investigate and identify a subset of the Shadow refinement theorems which might safely be promoted to the refinement in the full probabilistic non-interference model. We consider a probabilistic model that is based on a generalized version of Morgan’s Shadow semantics [96], where secure refinement is defined such that a statistical attack on the refinements reveal no more information than on the specifications. We have identified when possibilistic abstractions of probabilistic mechanisms are still sound, so that we can use the simple Shadow model for proofs, automate them using the RODIN platform as described earlier. Our main idea is to focus on the set of programs that always maintain total uniformity of the hidden distribution. With this restriction, reasoning about these programs probabilistically is the same as using the Shadow semantics.

We illustrate our approach using Chor’s Private Information Retrieval scheme [57]. In the future, we would like to expand the expressivity of the sequential program language and explore its applications to other protocols.

This work has been presented at the CryptoForma Workshop’11 (Limerick, Ireland). An extended version of the paper is going to be submitted to a special issue of Formal Aspects of Computing based on the CryptoForma Workshop’11 [18].

2.2 Mapping security-design models to enable formal analysis

Design models of secure systems should provide not only a classical representation of structure and behavior of the application but also address the modeling of security aspects. In fact, there are a set of approaches that address modeling of security aspects of service-oriented architectures (SOAs) and systems. Examples of the former are SECTET framework [76], the SENSORIA approach UML4SOA [72], and SecureSOA [97]. The first one proposes the use of sequence diagrams for the representation of a set of security patterns, in UML4SOA security features are modeled as non-functional properties using class diagrams, and the latter relies on Fundamental Modeling Concepts (FMC) block diagrams and the Business Process Modeling Notation (BPMN). UML-based Web Engineering (UWE) with security features
integrates security aspects in the web development process \([13, 12]\) allowing for a graphical representation of model authentication, access control and secure connections based on UML modeling techniques and security patterns.

Other approaches like UMLsec and SecureUML provide secure modeling elements for non-domain specific software. UMLsec \([82]\) is an extension of UML with emphasis on secure protocols. It is defined in form of a UML profile including stereotypes for concepts like authenticity, freshness, secrecy and integrity, role-based access control (RBAC), guarded access, fair exchange, and secure information flow. In particular, the use of constraints gives criteria to evaluate the security aspects of a system design, by referring to a formal semantics of a simplified fragment of UML. UMLsec models are more complex and more detailed than UWE models. The main difficulty is the missing support of UML2 since the provided UMLsec tools \([122]\) and the used CASE tool ArgoUML \([35]\) still only support UML 1.4.

SecureUML \([87]\) is a modeling language for the model-driven development of secure, distributed systems also based on UML. It provides modeling elements for the role-based access control and the specification of authorization constraints. A SecureUML dialect has to be defined in order to connect a system design modeling language as, e.g., ComponentUML to the SecureUML metamodel, which is needed for the specification of all possible actions on the predefined resources. In our UWE approach, we specify role-based execution rights to methods and access to data in a Basic Rights Model using dependencies instead of the SecureUML association classes, which avoids the use of method names with an access related return type.

The expressiveness and flexibility of the UML profiles for security and other security modeling techniques makes it desirable to obtain feedback on whether the model indeed satisfies the modeler’s security intentions.

The modeling techniques that build on standard UML and, in particular, UML state machines, by reducing many security modeling features to plain UML expansions, may apply formal techniques developed for standard UML, like model checking \([85, 118]\) or theorem proving \([40]\).

**Contributions** For our UWE approach we use the UML model checking tool Hugo/RT \([85]\) to check reachability within web applications. For example, it is used to check a state machine that is derived from the UML state machine in Figure 2.1. ‘Derived’ means that UWE stereotypes are transformed to plain UML without stereotypes, using transformation rules specified in the UWE profile. The figure depicts an excerpt of UWE’s navigation model of a Secure Address Book case study (the whole example can be found in \([30]\)). In the derived version, the prefix “Extended” is added to all transformed elements. Afterwards, it is possible to ensure that, e.g., no unauthorized user can enter the state ExtendedInternalAreaAdmin, i.e., whenever in this state the current user must play the role admins. In the temporal OCL extension supported by Hugo/RT this property reads:

```
G a.inState(AddressBookApplication.
    ExtendedInternalAreaAdmin) implies
a.currentUser.role == ADMINS;
```

where G is the linear-temporal logic operator “always” and object a represents the application.

In general, web applications have little control on what a user may try in order to reach some location, be it that the user just follows links or that the user browses to a location directly using link guessing or bookmarks. We thus build (currently manually) an attacking user who tries all possible interactions with the modeled web application in all possible ways; this user is again represented as a UML state machine. Hugo/RT translates the state machines for the web application and the user, as well as the assertion into the input language of a back-end model checker, in this case SPIN \([79]\). SPIN then verifies that the assertion indeed holds.

The model driven process then consists of (1) developing secure web applications using a UML case tool and the UWE profile, (2) mapping the UWE models to plain UML models, and (3) verifying security properties such as reachability of navigation nodes or restricted access to certain nodes. The tool support for this development process is provided any UML CASE tool, with the use of the UWE profile. Additional support is provided by the MagicUWE plugin \([23]\) for MagicDraw \([25]\), which eases the modeling task by a set of direct accessible stereotyped elements, shortcuts and patterns. Technically, UWE is a mainly UML class and statechart-based approach that uses these techniques for a model-driven development of secure applications. Class diagrams are used to specify the content as well as the rights model,
statecharts yield precise UML-based navigation models. Additionally, the statecharts can be subjected to model checking for verifying reachability of navigation nodes in general and of those that are restricted to authorized users. In fact, the security features of UWE could be used in combination with any UML-based web engineering approach; however, we combined them in our UWE method at the first step. Further information can be found in [13].

2.3 Algorithmic verification

We report on extending the scope of automatic protocol verification tools for the verification of Future Internet protocols and services and covers Task 9.1.3 on Extended Model Checking. These extensions introduce new attacker models such as multiple non-communicating intruders and coverage of XML rewriting attacks or new primitives such as combining security protocols with authorization policies for the specification and verification of service-oriented architectures.

2.3.1 Modeling multiple intruders and XML rewriting attacks with general deducibility constraints

A successful approach for detecting many logical flaws in security protocol specifications is to assume that the cryptography is perfect and use a Dolev-Yao intruder model. This approach has been extensively investigated in recent years [98, 100, 121, 62]. In particular symbolic constraint solving has proved to be a very successful approach in the area. It amounts to express the possibility of mounting an attack, e.g. the derivation of a secret, as a list of steps where for each step some message has to be derived from the current intruder knowledge. These steps correspond in general to the progression of the protocol execution, up to the last one which is the secret derivation.

One of the formalisms allowing to express a possibility of an attack on a protocol is a so-called systems of deducibility constraints. In this context, we have recently proposed a satisfiability algorithm for a large class of constraints systems. This class allows one to model multiple intruders and semi-structured data like XML. For a detailed account of this work, we refer the reader to [1].

Multiple intruders Many works in the literature have reduced the protocol insecurity problem to the satisfiability of so-called deducibility constraints (e.g. [115, 113, 53]). Up to one exception [94, 95], all
constraint satisfaction procedures rely on two strong assumptions about the constraints to be processed: knowledge monotonicity and variable origination. Constraints satisfying this hypothesis are called well-formed constraints in the literature and they are not restrictive as these conditions hold when handling standard security problems with a single Dolev-Yao intruder. However, in some situations it can be quite useful to relax these hypotheses and consider general constraints, that is constraints without the restrictions above. General constraints naturally occur when considering security problems involving several non-communicating Dolev-Yao intruders, each of them controls its own set of communication channels.

For instance, this model may be motivated by a case where a spy succeeded to install several small devices on parts of a deployed network (e.g. network cables) in some organization such that each device controls its own communication channels, but has no means to communicate directly with the other devices due to network configuration and physical conditions. After some time the spy can return to this organization and get back his devices. Then he is able to read and join the collected information to derive some secrets (note that the devices are not only passive eavesdroppers, but can affect the protocol execution).

Remark that if intruders can communicate during protocol execution, the model becomes attack-equivalent to one with a unique Dolev-Yao intruder [117].

XML rewriting attacks Enriching standard Dolev-Yao intruder model with different equational theories [43, 56] like exclusive OR, modular exponentiation, Abelian groups, etc. [60, 54, 63] helps to find flaws that could not be detected considering free symbols only. A particularly useful theory is the theory of an ACI operator (that is associative, commutative and idempotent) since it allows one to express sets in cryptographic protocols.

As remarked in [55], the XML format requires to consider set of terms to model message content rather than a term. Since security standards for Web Services like WS-SecurityPolicy are applied to XML messages, it is reasonable to consider properties of XML representation in combination with security elements like encryption and signature.

While protocols that use XML can be vulnerable to the same attacks as classical ones, the new type of attacks called XML rewriting attacks [48, 47] can be also applicable. Moreover, the parser of XML message can interfere the protocol execution since the protocol participants may use their own implementation which can select different nodes as an answer to the same request. Thus, one should consider a non-deterministic behavior for XML parsing. In our approach we take into account the mentioned points by considering ACI symbol as a set constructor, and since unification modulo ACI is in general not single-valued the non-determinism of XML parsing is achieved.

Contributions First, we show that it is still possible to derive an NP decision procedure for detecting attacks on a bounded number of protocol sessions without relying on the usual assumptions, namely monotonicity: the intruder never forget information and variable origination: if the protocol requires some participant to send a term involving a variable $x$, he must receive $x$ before.

Second, this result extends previous ones [94, 95] by allowing non-atomic keys and the usage of an associative, commutative and idempotent operator that can be used for instance to model sets of nodes in an XML document. Third, we will remark that the satisfiability procedure we obtain for general constraints is a non-trivial extension of the one for well-formed constraints by showing that this procedure cannot be extended to handle operators with subterm convergent theories since satisfiability gets undecidable in this case. On the other hand, it is known that satisfiability remains decidable for the standard case of well-formed constraints with the same operator properties [44]. Finally, we sketch some applications of our results: detecting attacks based on XML representation of messages and solving protocol insecurity problem with bounded number sessions with regard to multiple non-communicating intruders.

2.3.2 Integrated specification and verification of security protocols and policies

Security protocols and authorization logics are two major techniques used in securing software systems. A central role of any (security) protocol is to give meaning to the messages that are exchanged in the course of the protocol [110]. For example, a signed X.509 certificate sent by a certificate authority is in many security protocols meant to imply that the authority endorses the public key and its owner, mentioned in the certificate. There are several ways to make the meanings of messages, and in general actions,
a protocol explicit, e.g. by associating epistemic effects to the actions [68]. In [17], we propose a formal language for specifying service-oriented architectures, in which

- the messages received by a service are interpreted in terms of policy statements of the service, and
- the authorization policies of the service constrain the actions the service can perform.

The proposed language is well suited for integrated specification of security protocols and authorization policies in service-oriented architectures. We see a service-oriented architecture as a collection of finitely many services which communicate over insecure media. Each service consists of a number of processes that run in parallel and share a policy engine. Processes communicate by sending and receiving messages, as it is usual in asynchronous message passing environments. Each send event is constrained by a guard, and each receive event leads to an update. Guards and updates belong to the policy level, as opposed to send and receive events which constitute the communication level. In anthropomorphic terms, services “think” at the policy level, and “talk” at the communication level.

From an operational point of view, guards are predicates which, if derivable by the policy engine of a service, allow the service to perform a corresponding send action, cf. Dijkstra's guarded command language [66]. Updates are also predicates at the policy level. When a service receives a message in one of its processes, it adds the corresponding update predicates to its policy engine. Intuitively, updates associate meanings to the messages a service receives in terms of predicates in the policy level. The notion of updates is similar to the assumptions which are relied upon after receiving a message, in the trust management model of Guttman et al. [75].

**Motivations** The separation between the communication and policy levels is a useful abstraction for better understanding each of these levels. Indeed, distributed authorization logics, such as [65, 45, 74], typically abstract away the communication level events by assuming that all the policy statements exchanged among the participants are signed certificates. This frees the modelers from specifying the exact routes through which the statements travel, etc. The abstraction however obscures how each of the policy statements are represented (as messages) in a given application, how messages are interpreted as policy statements, whether there is a place for misinterpretation, etc. For instance, one would expect that the statement “Ann says employee(Piet)” is added to the policy engine of a service, only after the service receives a message which is meant to indicate that Piet is Ann's employee. However, the meanings of messages (determined by their format, who has signed them, etc.) is often not specified in the policy level. Therefore, in a concrete environment, it is unclear whether or not the attacker can fake a message which would mean that Piet is Ann's employee, even though he is not. Similarly, formal specifications of security protocols, e.g. as in [99, 112], fully detail the format of the exchanged messages, while the meanings of messages in terms of policy statements are left unspecified.

While maintaining the separation between the communication and policy levels, we believe that, for a thorough security analysis, the interaction between the two levels must also be defined precisely. A typical specification in our proposed language thus consists of three components: communication level events, policy level decisions, and the interface between the two. As the interface between the levels is explicitly present in the specifications, a more precise security analysis of service-oriented architectures becomes possible. This singles out our specification language from the formalisms which focus on either the communication or the policy level, and hence neglect their interactions.

We assume that the attacker is in direct control of the communication media, i.e. messages are passed through the attacker. The message composition capabilities of the attacker may, for example, reflect the Dolev-Yao threat model [69]. The attacker can indirectly affect the policies of the participating services, by sending tampered messages which in turn affect the update predicates.

**Contributions** We define a generic reachability problem for service-oriented architectures specified in the language. The reachability problem subsumes the secrecy problem for security protocols and the safety problem for authorization policies. The reachability problem turns out to be undecidable in general, even when assuming a finite bound on the number of participating services. We give a decision algorithm for the reachability problem under the following two conditions: (1) the message composition and decomposition capabilities of the attacker reflect the Dolev-Yao threat model, and (2) policy engines of (honest) services are centered around the trust application and trust delegation rules à la DKAL [74].
besides type-1 theories. Type-1 theories are sufficiently expressive for modeling, e.g., RBAC systems with role hierarchy. The trust application and trust delegation rules, which are the core of many distributed authorization logics [65, 45, 74], intuitively state that

- (Trust application) If Ann trusts Mike on statement $f$, and Mike says $f$, then Ann believes $f$ holds.

- (Trust delegation) If Ann trusts Mike on statement $f$, and Mike delegates the right to state $f$ to, e.g., Piet, then Ann trusts Piet on statement $f$.

Trust delegation often contributes to the resilience and flexibility of access control systems. In practice, however, for a given application, trust delegation may, or may not, be allowed. Our formalism and decision algorithm can be adapted to exclude (transitive) trust delegation, if desired.

The decidable fragment is of practical interest: several industrial service-oriented architectures studied in the context of AVANTSSAR [38] (The EU Project on Automated Validation of Trust and Security of Service-oriented Architectures) fall into this fragment. As a comprehensive example, a case study on specifying and verifying an on-line car registration service [38], stemming from the European initiative for points of single contact, is reported in [17].

To prove our decidability result, we encode the derivation of authorization predicates in the policy engine of a service into message inference trees induced by the Dolev-Yao deduction rules. The encoding benefits us in two ways: (1) it simplifies the decidability proof, and (2) it allows us to build upon existing tools which have been originally developed for verifying security protocols. In particular, we have extended the constraint solver of Millen and Shmatikov [99] in Prolog to validate service-oriented architectures.

Note that verification algorithms for correctness of security protocols and authorization logics have been mostly developed in isolation. For instance, it has been shown that the secrecy problem is decidable for security protocols with a bounded number of sessions [112, 99]. For these results the local computational power of the processes is limited to pattern matching, hence not fully accounting for authorization policies of the participants. Likewise, (un)decidability results for the safety problem in the HRU access control matrix model [77], and authorization logics such as [65, 45, 74] abstract away communication level events and their effects on policy level decisions. In contrast, our decision algorithm for reachability takes the communication and policy levels into account, and also covers the interface between them.

For more details about this work, we refer the reader to [17].

Related work Our proposed language can be used to specify security protocols as it is common in the literature, e.g. see [99]. Authorization policies are modeled as logic programs in the language. Logic programs have been extensively used for specifying and reasoning about policies, e.g. see BINDER [65], SECPAL [45], and DKAL [74].

Recent progress in analyzing business processes, augmented with authorization policies, is related to our work, e.g. see [37, 114]. These studies focus on using specific formalisms and techniques for selected case studies, and thus do not consider decidability issues in general. A notable exception to this is [41], where workflows, policy level predicates and their interfaces are all formalized in first-order logic. Reachability is not considered in [41].
3 Assurance in Implementation

Several assurance techniques are available to ensure the security at the implementation level. Internet application security can be validated through testing. Here the focus is on the automatic generation of effective test cases. Once a potential problem is found, we need efficient debugging techniques that help us to understand the cause of the fault or attack. Testing and debugging are covered in Section 3.1. Runtime monitoring and enforcement is a complementary technique to ensure that the running system satisfies the required security properties. This topic is discussed in Section 3.2.

3.1 Testing and debugging

This subtask deals with techniques and tools for automating the process of testing and debugging. Testing is the process of finding faults in an application after it has been implemented, but before it is deployed to the final users. The main goal of testing techniques is to produce inputs that manifest faults in the application. Debugging is the process of giving an input that manifests a fault, gathering enough information about the fault so that it can be fixed. This typically requires understanding the fault and its causes.

Testing and debugging complement techniques that happen earlier in the SDLC such early assurance techniques used during the design of the application and secure programming techniques used during its implementation. Even when early assurance and secure programming techniques have been used, testing is still needed to identify faults that may have been introduced during the implementation of the application or during the integration with external systems, while debugging is needed to fix those faults.

Traditionally, testing and debugging have been slow and costly processes requiring a large amount of manpower. In the highly dynamic environment of the modern and Future Internet, where applications need to be quickly updated and new applications are constantly being developed, it is of fundamental importance to develop automatic techniques and tools that reduce the time needed for testing and debugging, while assuring the safety and reliability of the deployed application.

This initial set of solutions covers three major angles that are characteristic of the testing and debugging of service-oriented applications in the Future Internet. First, in Section 3.1.1, we present techniques for penetration testing that leverage the availability of a high-level model of the application (generated in earlier stages of the SDLC) to automatically produce attack inputs. Those attacks inputs are replayed to the application under test to check whether they manifest a fault. Then, in Section 3.1.2, we present techniques for testing complex access control policies often found in Internet-based applications that require tight control over the access to private information. Finally, in Section 3.1.3, we present a debugging technique called differential slicing that can be applied, among other uses, for automatically analyzing a fault by comparing two executions of an application: one leading to a fault and one that does not.

3.1.1 Model-based testing

Model-based testing is a technique according to which the test cases are designed using the knowledge of a formal model of the system. The advantage of this technique is twofold. First, the formal model can be analyzed by a dedicated tool, aiming at automating the test generation phase. Second, the formal model provides the expected result of the test, namely the expected behaviour of the system under test when the test is executed.

The model can be a simple specification of the inputs of the system under test or a complete description of it (or a significant self-contained part) which allows to build extensive test cases involving long sequences of operations.

Model-based penetration testing

Penetration testing consists in testing the security of a system by playing the role of a malicious attacker. Coupling model-based testing and penetration testing thus consists in using a model to generate tests from an attacker’s point of view. We have studied in [15] a penetration testing approach dedicated to the validation of security protocols, that is now described.
Security protocols represent exchanges of messages between actors (a.k.a. agents) that aim at establishing a trustful communication link. The messages that are exchanged can be encrypted/decrypted, hashed, signed, etc. Agents communicate using Dolev-Yao channels that are unsafe (i.e. all exchanged messages can be intercepted by an intruder). Security protocols can be modeled in High-Level Protocol Specification Language (HLPSL) [36] a dedicated language designed during the AVISPA project. HLPSL models can be checked using dedicated verifiers, including the CL-Atse model checker [121] that we use in our work, to ensure that the protocol does not present any security flaws, even if cryptographic primitives are assumed to be perfect (e.g. man-in-the-middle attacks such as in the original NSPK protocol in which a participant can be impersonated). Once a model of the protocol is verified, it is mandatory to make sure that the protocol is correctly implemented on the system under test.

To achieve that, our approach relies on the mutation of the HLPSL model of the protocol. In this context, a security protocol is mutated (i.e. atomic modifications are made inside the protocol) so as to represent, at the model level, either a possible mistake in the implementation (e.g. changing the order of the message elements), or a possible choice of implementation (e.g. use of a XOR encryption scheme). Each mutant is then analyzed by the AVISPA tool-set so as to check if the mutation introduced a security flaw. If the mutant is still safe, then the mutation has no effect on the security properties requested for the protocol. If the mutation produced an incoherent protocol (i.e. a protocol that cannot be executed until the end for each participant), then it is discarded. Finally, if the mutation introduced a security flaw, the CL-Atse model-checker returns an attack trace that can be used as a test case, which consists of the description of messages sent and received by an attacker until a flaw is discovered. Replaying such a test case on a system under test may reveal a defect if the attack trace can be replayed entirely.

This approach relies on several mutation operators, which we describe in our paper [15]. We have evaluated our mutation operators on the set of 50 security protocols modeled during the AVISPA project. Our next step is then to increase the variety of mutation operators, and to concretize the model-based test cases we have produced to run them on a system under test.

Functional model-based testing for security

In the security context, there are two approaches to provide input for test generation. The first approach is to extend a language from the security domain to test generation. For example, specific security languages allow to define role as RBAC, access control aspects with XACML, or a more general language proposed to take into account (security) requirements/properties as in [70]. These approaches develop specific testing tools such as SecureUML [87] or for access control with the definition of policies in [111], a dedicated test generation process as in [93], or a full framework (specification to test generation) as proposed by [104]. These methods can be associated with the Model-Driven Testing (MDT) approach [39]. In MDT, the model is used to describe the test choreography as a sequence of possible or prohibited actions such as, for example, the script of an attack or of spoiling access control rules. The second approach extends an existing modeling language for testing in order to cover security aspects. These methods can be associated with the Model-Based Testing (MBT) approach [123]. In MBT, the model is used to describe the behaviors of the system with respect to the control and observation interfaces.

In current works, we extend the functional test generation method based on a formal model with a language to describe specific dynamic aspects of the system. This language, originally defined in [81], allows us to represent the states (defined by specific values of a subset of variables) and transitions (operations) of the system. We first give the definition of a Test scenario.

Definition 3.1 (Test Scenario)  A test scenario is a specification of operation sequences potentially reaching specific states.

The specific state can be defined intentionally by an operation or extensionally by specifying values for a subset of the variables. The definition states “potentially reaching specific states”, since we do not know whether a particular state specified by the scenario is reachable.

We define a language for specifying these scenarios. Its grammar, which is defined in Figures 3.1 and 3.2, is organized into three layers. We now describe each of these layers separately.

The sequence layer (Figure 3.1, left column) uses a syntax similar to regular expressions to define test scenarios as operation sequences (repeated or alternated) that may lead to specific states.
SEQ ::= OP1 | "(" SEQ ")"  
| SEQ "." SEQ  
| SEQ REPEAT (ALL | _ONE)  
| SEQ CHOICE SEQ  
| SEQ "⇝" (" SP ")  
OP ::= operation_name  
| $OP  
| OP1  
| SEQ REPEAT (ALL | _ONE)  
| $OP {OPLIST}"  
| SEQ CHOICE SEQ  
| SEQ "⇝" (SP) OPLIST ::= operation_name  
| operation_name , OPLIST  
| operation_name  
| operation_name , operation_name  
REPEAT ::= ?  
| "{n  
| "{\{n," m\}  
SP ::= state predicate  

Figure 3.1: Syntax of the sequence and model layers

CHOICE ::= "|"  
| "⊗"  
| "OP|w CPTLIST"  
| "OP|e CPTLIST"  
ALL_or_ONE ::= "_one"  
| ε  
| (" cpt_label") CPTLIST ::= cpt_label  
| (" cpt_label")  

Figure 3.2: Syntax of the test generation directive layer

The model layer (Figure 3.1, right column) describes the operation calls at the model level and constitutes the interface between the model and the scenario.

Rule SEQ describes a test scenario as a regular expression composed of a sequence of operation calls. A sequence is either a simple operation call, denoted by OP1, the concatenation of two sequences (SEQ "." SEQ), the repetition of a sequence (SEQ REPEAT (ALL | _ONE)), or a choice between two or more (recursive call of the rule (SEQ CHOICE SEQ) sequences. In practice, we use bounded repetition operators: 0 or 1, exactly \( n \) times, at most \( m \) times, between \( n \) and \( m \) times. The final branch of rule SEQ, that is SEQ "⇝" (SP), defines a sequence leading to a state satisfying a state predicate (SP). The state predicate provides a useful help for the design of the scenario since it enables the definition of the target (a specific state defined by predicate on state variables) of an operation sequence, without necessarily having to enumerate all the operations that compose this sequence. We can consider the following three types of operation calls in a sequence (rule OP):

1. an operation name,
2. the keyword $OP, meaning “any operation”, or
3. the expression $OP \{(OPLIST) meaning “any operation except those of OPLIST”.

The test generation directive layer (Figure 3.2) helps driving the test generation process to improve the time generation and/or to define more complex test sequences (with specific state(s) of system defined in sequence) in order to obtain better test quality and better model coverage, as well.

We propose three kinds of directives aiming to reduce the search space for the instantiation of a test scenario.

The rule CHOICE introduces two operators denoted | and ⊗, for covering, respectively, the exclusive and inclusive interpretation of “or”. For example, if \( S_1 \) and \( S_2 \) are two sequences, \( S_1 \mid S_2 \) specifies that the test generator has to produce tests that will cover \( S_1 \) and other tests that will cover \( S_2 \), whereas \( S_1 \otimes S_2 \) specifies that the test generator has to produce test cases covering either \( S_1 \) or \( S_2 \). Rule ALL_or_ONE makes it possible to specify if all the solutions of the iteration will be returned (ε – default option) or if only one will be selected (_one).

The first case of rule OP1 indicates to the test generator that it has to cover one of the behaviors of the OP operation. The test engineer may also require to cover all behaviors by surrounding the operation with brackets (second case). Two further variants make it possible to select the behaviors that will be applied. The first one (/w) specifies the authorized behaviors and the second (/e) the refused ones, by using labels that are associated with the behaviors of the model operations (CPTLIST).

Example 3.2 (Scenario and Test Generation Directives) Consider the following fragment of a scenario.

\[
\text{\texttt{Smartcard::block \{BLOCK_OK_SD_CONT_APP\}\{1,3\}; one \Rightarrow \{Bank.applicationState = BLOCKED\}}}
\]
This scenario tries to repeat the invocation of the block operation 1 to 3 times with a nominal behavior until the Bank application instance is in a blocked state. In practice, any sequence of length 1, 2 or 3, that ends with the blocking of the banking application will satisfy this scenario. However, the test generation directive (_one) will only keep the first one that succeeds (i.e. the shortest sequence).

The scenarios are currently designed manually. They address a specific test intention that originates from the know-how of the validation engineer who designs the scenario. Each test intention is generally associated with a given requirement.

We use this language in combination with several other specification languages. For example, we define an access control model for smart card applications using the B modeling language [24]. We also work on an UML model of GlobalPlatform (GP)\(^1\). GP is the standard for managing applications on secure chip technology for smart cards. Our work on this case study consists in defining security properties on the GP card life cycle, verifying them on the GP model and then generate security tests exercising these security properties. With this approach our goal is to guarantee the correctness of a given model and gain confidence in it, especially when it is a critical system.

The process is summarized in Figure 3.3. First, a validation engineer designs a test model (Step 1). He then extracts the security properties from the specification in Step 2. In Step 3, he writes the corresponding UMLSec stereotypes. In Step 4, he then uses the UMLsec approach to validate the model against the security properties, to make sure that the model respects them. Once the model is declared correct, a Hoare triple for each property is exported in Step 5. Then, in Step 6, we use transformation rules to automatize the scenario writing with respect to the property and the scenario syntax defined in Figure 3.1. The created scenario is used in Step 7 to produce test cases exercising the property on system.

Thus, in this case study, we combine the MBT approach with the UMLsec [82] security verification technique, by using UMLsec stereotypes to verify the model with respect to given security properties. We define a procedure to generate security tests from the model via UMLsec stereotypes. The first results of this work are published in [16]. Currently, we work on an extension of this approach in order to take into account more stereotypes and to have an automatic verification and test generation process.

---
\(^1\) [http://www.globalplatform.org/](http://www.globalplatform.org/)
3.1.2 Testing of access control policies

XACML [108] has become the de facto standard for specifying policies for access control decisions in many application domains such as Service Oriented Architectures (SOAs) and Peer-to-Peer (P2P) systems. Access control policies need to be carefully designed and tested to protect data from unauthorized access. A common approach for testing XACML policies is the derivation of test inputs (XACML requests) that are used for probing the PDP (XACML policy implementation engine called Policy Decision Point), and checking the PDP’s responses against the expected ones.

Background and related work

At the root of all XACML policies is a Policy or a PolicySet. A PolicySet can contain other Policies or PolicySets. A Policy consists of a Target, a set of Rules and a Rule combining algorithm. The Target specifies the Subjects, the Resources, the Actions and the Environments on which a policy can be applied. If a request satisfies the target of the policy, then the set of rules of the policy is checked, otherwise the policy is skipped without examining its rules. A Rule is the basic element of a policy. Each rule contains a Target that is similar to the policy target and specifies the requests to which the rule is applicable. The heart of most rules is a Condition, which is a boolean function evaluated when the rule is applicable to a request. The result of the condition evaluation is the rule effect (Permit or Deny) if the condition is evaluated to be true, NotApplicable otherwise. If an error occurs during the application of a policy to a request, Indeterminate is returned as decision. More than one rule in a policy may be applicable to a given request. The rule combining algorithm specifies the approach to be adopted to compute the decision result of a policy containing rules with conflicting effects. The access decision is given by considering all attribute values describing the subjects, the resources, the actions and the environments of an access request and comparing them with the attribute values of a policy.

Policy testing is a critical issue and the complexity of the XACML language specification prevents the manual specification of a set of test cases capable of covering all the possible interesting critical situations or faults. This implies the need of automated test cases generation.

Some existing approaches consider the policy values in the test cases derivation. In particular, [90] presents the Targen tool that derives the set of requests satisfying all the possible combinations of truth values of the attribute id-value pairs found in the subject, resource, and action sections of each target included in the policy under test.

A different approach is provided by Cirg [91] that is able to exploit change-impact analysis for test cases generation starting from policies specification. In particular, it integrates the Margrave tool [71] which performs change-impact analysis so to reach high policy structural coverage.

Other approaches for policy testing are based on the representation of policy-implied behavior by means of models [120, 111]. Usually these approaches provide methodologies or tools for automatically generating abstract test cases that have to be then refined into concrete requests for being executed.

XACML requests derivation testing strategies

We propose the following two methodologies for deriving XACML requests for policy testing:

- A XACML Context Schema-based testing strategy described in [11];
- A combinatorial testing strategy called Simple combination.

The XACML Context Schema-based testing strategy exploits the XACML Context Schema that describes the overall structure of the accepted input requests for the PDP and it is a model formally describing what constitutes an agreed valid input. The XACML instances, formatted according to the rules of the referred Context Schema, represent the conforming requests, i.e. allowed naming and structure of data for access requests.

The XACML Context Schema-based testing strategy applies a Category Partition [109] based approach, that provides a stepwise intuitive approach to functional testing, as follows: identify the relevant input parameters; define the environment conditions; combine their significant values into an effective test suite.

The Simple combination strategy applies a simple combinatorial strategy to the XACML policy values.
Both testing strategies have been implemented in a tool called X-CREATE, which automatically provides the test suites.

We performed a comparison of the proposed methodologies in terms of their fault detection effectiveness. In the following, we describe these strategies and present some results obtained by the application of mutation testing on a set of real policies.

**XACML Context Schema-based strategy** This approach generates conforming XML instances from the XACML Context Schema by applying a variant of the Category Partition (CP) method [109] and traditional boundary condition. In particular, the occurrences declared for each element in the schema are analyzed and, applying a boundary condition strategy, the border values (minOccurs and maxOccurs) to be considered for the instances generation are derived.

Combining the occurrence values assigned to each element, a set of intermediate instances are derived. The final XACML requests are derived from these instances intermediate by assigning values to the various elements.

Given the XACML 2.0 Context Schema, this strategy can derive \(3^Y \times 2^Z\) intermediate instances, where \(Y\) is the number of schema elements with unbounded cardinality, and \(Z\) is the number of elements having \([0,1]\) cardinality. Specifically, a maximum number (MAXREQ) of \(3^{15} \times 2^1 = 28697814\) structurally different intermediate requests can be derived. By applying the pair-wise approach [58], a test subset with good fault detection capability can be finally selected and successively used for final requests generation. In particular, the intermediate instances are filled with the values taken from the XACML policy for elements and attributes. A detailed description of the steps for the XACML policy analysis and assignment of the policy values to the intermediate instances for the final requests generation is provided in [11].

However, we proved in [11] a comparable or greater fault detection effectiveness of the XACML Context Schema-based strategy than the testing strategy implemented in the Targen tool [90] that represents the most similar existing approach to the XACML Context Schema-based strategy testing strategy.

**Simple combination strategy** In this strategy, we apply a combinatorial approach to the policy values. We define the SubjectSet, ResourceSet, ActionSet, and EnvironmentSet sets as described in [11], considering also random entities for robustness and negative testing purposes. In particular, we define a subject entity as a combination of the values of elements and attributes of the SubjectSet set, and similarly the resource entity, the action entity and the environment entity as a combination of the values of the elements and attributes of the ResourceSet, ActionSet, and EnvironmentSet respectively.

Then, we generate all combinations of values entities (subject entities, resource entities, action entities and environment entities). In particular, we consider a n-wise combining approach which is an effective test case generation technique based on the observation that most faults are caused by interactions of at most n factors. For example, pairwise (three and four) generated test suites cover all combinations of two (three and four) factors.

- First, we apply the pair-wise combination, i.e. each value entity is combined in a pair with every other value entity. We obtained a set called PW set;
- Then, we apply the three-wise, i.e. each value entity is combined in a pair with every couple of value entities and we obtain a set called TW set;
- Finally, we apply the four-wise, i.e. all possible combinations of the values contained in the entities sets, and we obtain a set called FW set.

These sets have the following inclusion propriety \(PW \subseteq TW \subseteq FW\). For eliminating duplicated combinations we consider the following set of combinations: PW called Pairwise, TW \(\setminus PW\) called Threewise and FW \(\setminus (TW \cup PW)\) called Fourwise. For each combination included in the above sets, we generate a simple request containing the entities of that combination. The derived requests are first those obtained using the combinations of the Pairwise set, then those ones using the combinations of the Threewise set and finally those using the combinations of the Fourwise set. In this way, we try to generate a test suite guaranteeing a coverage first of all pairs, then of all triples and finally of all quadruples of values entities derived by the policy. The maximum number of requests derived by this strategy is equal to the cardinality of the FW set. This number represents the
Table 3.1: Mutant-kill ratios achieved by test suites of Simple Combinatorial and XACML Context Schema-based

<table>
<thead>
<tr>
<th>policy</th>
<th># Mut</th>
<th># Req</th>
<th>Mut Kill %</th>
<th># Mut</th>
<th># Req</th>
<th>Mut Kill %</th>
<th># Mut</th>
<th># Req</th>
<th>Mut Kill %</th>
<th># Mut</th>
<th># Req</th>
<th>Mut Kill %</th>
</tr>
</thead>
<tbody>
<tr>
<td>demo-5</td>
<td>23</td>
<td>84</td>
<td>64.7 %</td>
<td>84</td>
<td>95.85%</td>
<td></td>
<td>35</td>
<td>95.85%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>demo-11</td>
<td>22</td>
<td>40</td>
<td>68.75%</td>
<td>-</td>
<td>-</td>
<td></td>
<td>40</td>
<td>95.45%</td>
<td></td>
<td>18</td>
<td>95.45%</td>
<td></td>
</tr>
<tr>
<td>demo-26</td>
<td>17</td>
<td>16</td>
<td>50%</td>
<td>-</td>
<td>-</td>
<td></td>
<td>16</td>
<td>94.11%</td>
<td></td>
<td>9</td>
<td>94.11%</td>
<td></td>
</tr>
<tr>
<td>Healthcare</td>
<td>14</td>
<td>48</td>
<td>78.57%</td>
<td>25</td>
<td>78.57%</td>
<td></td>
<td>48</td>
<td>78.57%</td>
<td></td>
<td>13</td>
<td>78.57%</td>
<td></td>
</tr>
</tbody>
</table>

Testing strategies comparison We compared the effectiveness of the Simple Combinatorial with that of the XACML Context Schema-based testing strategy by answering to the following research questions:

TSEff Adopting the proposed stopping criterion, is the fault detection of the Simple Combinatorial strategy similar to that of the XACML Context Schema-based strategy?

TSDecr Is it possible to reduce the test suites maintaining the same level of fault detection?

The test suites of the two test strategies were derived by the tool X-CREATE. For the comparison we used (see column 1 in Table 3.1), three policies presented in [90] (specifically demo-5, demo-11, demo-26) and a new real policy ruling a health care service, here called Healthcare.

We applied mutation analysis which is a standard technique to assess the quality of a test suite in terms of fault detection [64]. Mutation has been applied here to introduce faults into the policies. In particular, each mutant contains one fault, inserted considering the mutation operators for XACML policies indicated in [92]. Table 3.1 in the second column, shows the total amount of mutants obtained for the four policies: 23, 22, 17 and 14 mutants for demo-5, demo-11, demo-26 and Healthcare policies respectively. The sets of mutants obtained have been used for answering the two above Research Questions labeled TSEff and TSDecr.

RQ TSEff Applying the Simple Combinatorial methodology we obtain four test suites of cardinality 84, 40, 16 and 48 for demo-5, demo-11, demo-26 and Healthcare policies respectively (see Table 3.1 third column). In parallel, by using XACML Context Schema-based testing, we generated the same number of requests generated by the Simple Combinatorial strategy for each policy, so to get a fair comparison. Finally, the test suites obtained by the Simple Combinatorial method and those derived by the XACML Context Schema-based approach have been executed on the associated policies and on their mutants.

Each request in the requests set is executed on a policy and on each of its mutants, if the produced responses are different, then the mutant policy is killed by the request, i.e. the inserted fault is detected. For each policy under test, Table 3.1 reports the percentage of mutants killed using Simple Combinatorial strategy (4th column), and XACML Context Schema-based one (8th column).

Observe the obtained results we can deduce that the effectiveness of the XACML Context Schema-based derived test suites is generally higher than that provided by the test suites derived by Simple Combinatorial strategy. Note that the 8th column shows the maximum reachable percentage of mutants killed. The 100% was not feasible because in the set of mutants for demo-5, demo-11 and demo-26 there was an equivalent mutant\(^2\) and in that of Healthcare policy there were three. A deep analysis of the mutants killed by the test suites derived by the Simple Combinatorial strategy showed that these test suites were not able to detect situations where the access decision of the policy rules depends concurrently on the values of more than one subject or resource or action or environment entity. By construction the\(^2\)

\(^2\)A mutant is equivalent if it is not possible to find a test case that could kill it.

NESSoS - 256980 31
request derived by the Simple Combinatorial strategy contains at most a subject, a resource, an action and an environment entity. For demo-5, demo-11, demo-26 the structures of the requests generated by the XACML Context Schema-based testing strategy having more than one entity are responsible for the higher test effectiveness of this strategy. Considering instead the Healthcare policy where the satisfiability of the policy rules depends on combinations of a single subject, resource, action and environment entity, the two strategies are equivalent.

RQ TSDecr Since the test suites derived by the XACML Context Schema-based testing strategy were able to achieve the maximum reachable percentage of mutants killed for all the policies considered and the Simple combinatorial strategy achieved this for the for the Healthcare policy, a natural consequence was to evaluate whether it was possible to reduce these test suites maintaining the same level of fault detection. For this, we reduced the original set of requests while maintaining the maximum percentage of fault detection effectiveness. In Table 3.1, the columns 5 and 9 show the results obtained for the two test strategies respectively, i.e. the minimum number of requests needed for achieving the higher percentage of mutants killed for each policy.

Discussion and ongoing work The analysis of the experimental results evidenced a greater fault detection percentage of the XACML Context Schema-based testing strategy due to the variability of the structures of the generated requests, which could result in improved effectiveness of the derived test suite. The most important advantage of the XACML Context Schema-based testing strategy is the structure variability of the derived requests: i.e., a request may include more than one subject or resource or action, or environment entity. This feature is especially important for testing policies or rules in which the access decision involves simultaneously more than one subject or resource or action or environment.

However, the possibility of having a high variability in the structure of requests is also the main limitation of the XACML Context Schema-based testing. As described above, the total amount of different intermediate instances that can be generated by varying the structure only of the XACML 2.0 Context Schema is MAXREQ = 28697814, which is extremely high for any kind of policy specification. Even if in the XACML Context Schema-based testing strategy a pair-wise approach [58] is applied for ordering and selecting the instances that maximize the fault detection capability, a generic stopping criterion is missing. In particular, once the intermediate instances are generated, by construction they are filled with values taken from the tested policy, but no guarantee is provided that the values of the subjects, resources and actions of the policy are all covered if an arbitrary number of requests is derived.

A limitation of Simple Combinatorial strategy was that by construction the derived requests contain at most a subject, a resource, an action and an environment entity. Then this strategy is not able to detect situations where the satisfiability of the policy rules depends simultaneously on the values of more than one entity.

In future work, we plan to provide a new strategy in which by construction the derived requests contain all the possible combinations of more than one subject, resource, action and environment entity. In this way, the number of requests increases exponentially and could be soon comparable to the maximum number of requests obtained by the XACML Context Schema-based testing strategy, i.e MAXREQ. For avoiding this, some constraints for limiting the number of obtained requests could be applied. These constraints should take care of the cardinality of the subject, resource, action and environment entities that need to be simultaneously verified for giving an access decision. However, to assess the fault detection effectiveness of such a new test inputs derivation strategy a comparison with that of XACML Context Schema-based approach should be provided.

3.1.3 Debugging

While automatic testing techniques help locate inputs that demonstrate the existence of a bug or vulnerability in an application, they typically do not provide detailed information about the cause of such vulnerability. Such information is needed to quickly diagnose and fix the vulnerability, minimizing costly human involvement. Thus, in addition to automatic testing techniques, it is also critical to develop automatic debugging techniques that produce detailed information about the cause of a vulnerability. This information is meant to be consumed by a developer, who will use it to quickly understand and fix the bug. In this section, we introduce differential slicing, a novel debugging technique that identifies causal relationships between inputs and outputs.
execution differences [19]. Although differential slicing has many different applications, in this work we focus on two important ones: crash analysis and malware.

**Introduction.** Often, a security analyst needs to understand two runs of the same program that contain an execution difference of interest. For example, the security analyst may have a trace of an execution that led to a program crash and another trace of an execution of the same program with a similar input that did not produce a crash. Here, the analyst wants to understand the crash and why one program input triggered it but the other one did not, and use this knowledge to determine whether the bug causing the crash is exploitable, how to exploit it, and how to patch it.

For another example, a security analyst may use manual testing or previously proposed techniques to find trigger-based behaviors in malware [103, 59, 61, 50]. The analyst may obtain an execution trace of a piece of malware (e.g., a spam bot) in environment A, which does not exhibit malicious behavior (e.g., does not spam), and another trace of an execution of the same piece of malware in environment B, which does exhibit malicious behavior (e.g., does spam). However, knowing how to trigger the hidden behavior is not enough for many security applications. It is often important to know exactly why and how the trigger occurred, for example, in order to write a rule that bypasses the trigger [83]. Suppose there are many differences between environments A and B. The analyst needs to understand which subset of environment differences are truly relevant to the trigger, as well as locate the checks that the malware performs on those environment differences.

The two scenarios are similar in that one execution trace contains some unexpected behavior (e.g., the crash for the benign program and the non-malicious behavior for the malware) and the other trace contains some expected behavior. In both scenarios the analyst would like to understand why that execution difference, which we term the **target difference**, exists. This is a pre-requisite for the analyst to act, i.e., to write a patch or exploit for the vulnerability and to write a rule to bypass the trigger. In addition, the analyst needs to perform this analysis directly on binary programs because source code is often not available.

To automate the analysis in the above scenarios we propose a novel **differential slicing** approach. Given traces of two program runs and the target difference, our approach provides succinct information to the analyst about 1) the parts of the program input or environment that caused the target difference, and 2) the sequence of events that led to the target difference.

Automating these two tasks is important for the analyst because manually comparing and sieving through traces of two executions of the same program to answer these questions is a challenging, time-consuming task. This is because, in addition to the target difference, there are often many other execution differences due to loops that iterate a different number of times in each run, and differences in program input that are not relevant to the target difference (e.g., to the crash) but still introduce differences between the executions.

We implement our differential slicing approach and evaluate it for two different applications. First, we use it to analyze 11 real-world vulnerabilities. Our results show that the output graph often reduces the number of instructions that an analyst needs to examine for understanding the vulnerability from hundreds of thousands to a few dozen. We confirm this in a user study with two vulnerability analysts, which shows that our graphs significantly reduce the amount of time and effort required for understanding two vulnerabilities in Adobe Reader. Second, we evaluate differential slicing on 2 malware samples that check environment conditions before deciding whether to perform malicious actions. Our results show that differential slicing identifies the specific parts of the environment that the malware uses and that the output graphs succinctly capture the checks the malware performs on them. For additional details, we refer the reader to the published version of this work [19].

**Contributions.** This work [19] makes the following contributions:

- We propose differential slicing, a novel technique which, given traces of two executions of the same program containing a target difference, automatically finds the input and environment differences that caused the target difference, and outputs a causal difference graph that succinctly captures the sequence of events leading to the target difference.

- We propose an address normalization technique that enables identifying equivalent memory addresses across program executions. Such normalization enables pruning equivalent addresses from the causal difference graph and is important for scalability.
• We design an efficient offline trace alignment algorithm based on Execution Indexing [124] that aligns the execution traces for two runs of the same program in a single pass over both traces. It outputs the alignment regions that represent the similarities and differences between both executions.

• We implement differential slicing in a tool that works directly on binary programs. We evaluate it on 11 different vulnerabilities and 2 malware samples. Our evaluation includes an informal user study with 2 vulnerability analysts and demonstrates that the output of our tool can significantly reduce the amount of time and effort required for understanding a vulnerability.
3.2 Run-time verification: Monitoring usage control properties

Determining whether the usage of sensitive, digitally stored data complies with regulations and policies is a growing concern for companies, administrations, and end users alike. Classical examples of policies used for protecting and preventing the misuse of data are history-based access-control policies like the Chinese-wall policy and separation-of-duty constraints. Other policies from more specialized areas like banking involve retention, reporting, and transaction requirements. Simplified examples from this domain are that financial reports must be approved at most a week before they are published and that transactions over $10,000 must be reported within two days.

In the context of IT systems, compliance checking amounts to whether one can implement a process that monitors, either online or offline, other processes. Such a monitor needs to temporally relate actions performed by the other processes and the data involved in these actions. We analyzed, designed, and evaluated such monitors and monitoring architectures. In the following, we give an overview of our work and our contributions.

3.2.1 Interval-based and point-based properties

In [10], we analyze the impact of different time models on monitoring. We do this by presenting, analyzing, and comparing monitoring algorithms for real-time logics based on different time models. More concretely, we present monitoring algorithms for the past-only fragment of propositional metric temporal logics with a point-based and an interval-based semantics, also considering both dense and discrete time domains. We compare our algorithms on a class of formulas for which the point-based and the interval-based settings coincide. To define this class, we distinguish between event propositions and state propositions. The truth value of a state proposition always has a duration, whereas event proposition cannot be continuously true between two distinct time-points.

Our analysis explains the impact of different time models on monitoring. First, the impact of a dense versus a discrete time domain is small. The algorithms are essentially the same and have almost identical computational complexities. Second, monitoring in a point-based setting is simpler than in an interval-based setting: the monitoring algorithms for the point-based setting are conceptually simpler than the interval-based algorithms and our point-based monitoring algorithms perform better than our interval-based algorithms on the given class of formulas on which the two settings coincide.

Contributions Our contributions are as follows. First, our monitoring algorithms simplify and clarify key concepts of previously presented algorithms [89, 106, 105, 42]. In particular, we present the complete algorithms along with a detailed complexity analysis for monitoring properties specified in the past-only fragment of propositional metric temporal logic. Second, our monitoring algorithm for the dense, point-based time model has better complexity bounds than existing algorithms for the same time model [119]. Third, our comparison of the monitoring algorithms illustrates the similarities, differences, and trade-offs between the time models with respect to monitoring. Moreover, formulas in our fragment benefit from both settings: although they describe properties based on a more natural time model, they can be monitored with respect to a point-based time model, which is more efficient.

3.2.2 Monitoring tool

In [7], we present our monitoring tool MONPOLY for compliance checking, which implements the monitoring algorithm from [42]. The tool is publicly available from the web page http://projects.developer.nokia.com/MonPoly. Policies are given as formulas of an expressive safety fragment of metric first-order temporal logic (MFOTL). The first-order fragment is well suited for formalizing relations on data, while the metric temporal operators can be used to specify properties depending on the times associated with past, present, and even future system events. MONPOLY processes a stream of system events with identifiers representing the data involved and reports policy violations. We used and evaluated our tool in several case studies, see [9] and [8].
3.2.3 Monitoring of concurrent and distributed systems

In [8], we theoretically and practically tackle the problem of monitoring the usage of data in concurrent distributed systems. In particular, we explain the steps required to go from our original monitoring algorithm [7][42] to a working infrastructure capable of monitoring an existing distributed application producing millions of log entries per day.

In concurrent distributed systems, we have only a partial order on system actions since multiple actions can be performed simultaneously by different agents. The main theoretical challenge is to correctly and efficiently monitor the trace interleavings obtained by totally ordering the agents’ actions. In general, this is an intractable problem. We identify a subclass of formulas that describe properties that are insensitive to the ordering of actions labeled by the same timestamp and for which it suffices to monitor a particular merging of the logs, namely, the merging that assumes that actions with equal timestamps happen simultaneously. Furthermore, in case the given formula is outside this class we provide means to meaningfully monitor this merge by approximating the described property.

A practical challenge is to deploy adequate logging mechanisms in concurrent distributed systems. The mechanisms should be complete in that they log all occurrences of policy-relevant system actions. They should also be accurate in that if an action is logged then it has happened in the system and the corresponding log entry accurately describes the action, e.g., it describes the involved data and the associated timestamp is the actual time when the action happened. Incomplete or inaccurate logging may lead to false positives and false negatives when monitoring the system. We explain how we handle these practical challenges in a real-world application, Nokia’s Data-collection Campaign. See http://research.nokia.com/page/11367 for details on the campaign. Where possible, we use existing logging mechanisms and extract policy-relevant information from the produced log entries. For system components where no logging was available, we either added logging directly to the components or we extended the components with proxy mechanisms that logged actions. However, proxies have limitations: agents do not necessarily access data over a proxy and proxies see requested actions but not necessarily all the Effects on the involved data. In our case, the interactions could be accurately observed but not for all agents, which led to accurate but incomplete logs.

Furthermore, we evaluated the deployed monitoring architecture. In particular, we measured the performance of our monitoring tool MONPOLY [8] by checking compliance of different usage-control policies within Nokia’s Data-collection Campaign. Overall, our case study demonstrates the feasibility and benefits of monitoring the usage of sensitive data.

Contributions Our main contributions are as follows. We provide solutions for efficiently monitoring partially ordered logs, which is a central problem in monitoring real-time concurrent distributed systems. Moreover, we evaluate the performance of our monitoring approach and demonstrate its effectiveness on a real-world application.
4 Security Metrics

For more than two decades the security community has been looking for the metrics which can measure security correctly and unambiguously. A number of different metrics have been proposed. These metrics range from specific ones, which measure a specific part of a system (e.g., the time between antivirus updates), to general metrics, which assess security as a whole (e.g., attack surface) [80, 78]. The fact that these metrics are usually defined informally gives rise to ambiguities, which prevent a theoretical analysis. Therefore, recently, the need for a formal description of security metrics appeared. Since the experimental validation of correctness of security metrics faces various difficulties (e.g., it is almost impossible to get real data for statistical analysis because companies do not provide such sensitive information), we believe that formal validation could help to answer certain questions if a fine-grained model is proposed that is suitable for most metrics.

We present a formal model for security metrics. Section 4.1 proposes formal definitions of the security metrics and checks whether the metrics are able to detect changes in the security level of a system. Section 4.2 establishes formal relations between the security metrics and risk analysis. We propose a generic method for the assessment of the security of complex services under different metrics in Section 4.3.

4.1 Formal approach to security metrics. What does “more secure” mean for you?

Neither a single metric nor a closed set of metrics are widely accepted for correct measurement of security as a whole. Therefore, currently many metrics are used simultaneously. The reason for the large number and diversity of metrics is our inability to prove that a metric really measures security. One of the causes for this uncertainty is that there is no clear, unambiguous, and widely accepted definition of a “more secure” relation. Every inventor of security metrics defines what “more secure” is by means of metrics, but does not prove that the metric really indicates changes in security.

In our work [20], we consider the problem of defining what “more secure” means and propose a small step forward in this direction. We propose a formal model which is capable of describing many of the general security metrics and we define a simple and evident criterion which captures that one system is more secure than another one. The formal model was used to analyze whether security metrics are able to detect the changes in the security level and to study relations between different metrics. The target of our analysis is just a system which is applied out of a context (i.e., we do not consider preferences of attackers and possible impact). Thus, the metrics we considered tried to answer the question “how strong the security system is” instead of “how useful it is”.

Contributions. The first contribution of our work [20] is that we established the correspondence between the term “metric” in the theory of measurements and the one used in security analysis literature. In the theory of measurement the term “metric” is used as a distance between measurements, when in the security community this term is equal to the term “measurement” (according to the theory of measurement). We have formalized the following security metrics which can be found in the literature: number of attacks, minimal cost of attack, minimal cost for reduction of attacks, shortest length of attacks, maximal probability of attack, overall probability of success, attack surface metric, percentage of compliance. We evaluated them against a very simple empirical criterion saying that if a set of possible attacks on one system is broader than a set of attacks on another one then the latter system is more or (at least) equally secure than the former one. This criterion is not enough because many metrics satisfy it. Therefore, we either need to find more empirical criteria for a more fine-grained analysis or, ideally, define what we mean by “more secure”.

Second, we investigated dependencies among metrics and found that in a strict sense most metrics are independent (“X” in Table 4.1). However there are some correlations between metrics, i.e., the metric in a column can be derived using the metric from the row (“?X”) under some (often very strong) assumptions. For example, the maximal probability metric can be found from the shortest length of attack metric if every action has the same probability of success.
Finally, we found out that we do not have a strict empirical notion of “more secure” and, therefore, we cannot say which metric is good (or bad) for measuring security. Without this relation we can only say that security metrics should be used depending on the entity which requires evaluation of security, i.e., a stakeholder. In addition, we identified that a metric can be considered good or bad depending on the behavior of an attacker. We considered two models of the attacker. The omniscient attacker knows all possible attacks, the costs he has to pay for executing each attack, and also the probability that an attack will be successful. The attacker always selects the “easiest” way to attack the system (less costly or more probable). The blind attacker does not know anything about the system. The attacker finds the first possible attack and tries to execute it because there is no knowledge of how easy the attack is. By analyzing these models, we figured out that, even if we do not consider preferences of the attacker, its style of attacking makes certain sets of metrics more useful than others. Omniscient and blind attackers are simplistic models and for a more reliable analysis we need other models which better approximate behavior of attackers.

The general conclusion we made out of the work done is that in order to understand which metric is better we need to agree what “better” means. We found that, in general, metrics measure different aspects of security and different models of behavior of attackers may lead to selection of different metrics. Therefore, currently we are able to select the best metric only using our preferences (and our assumptions), when for a more objective selection more work is need.

### 4.2 Formal analysis of security metrics and risk

Risk analysis is the most widely used method for analyzing the complete picture of the security status of a system [116, 52, 73]. The main goal of this analysis is to compute the amount of possible losses which are caused by occurrences of various threats. Currently, security metrics and risk exist apart from each other and the relation between these indicators, although assumed, is not specified. On the other hand, risk is supposed to be one of the most general security indicators. Thus, risk already must incorporate some security metrics, but it is unclear how different metrics contribute to the overall risk value. Moreover, risk analysis is blamed for providing results with low precision and for consuming huge amounts of time [80]. In some situations, the exclusive contribution of security metrics to the overall risk value may facilitate the analysis and a preliminary assessment.

**Contributions.** The main goal of our work [21] was to establish the relation between security metrics and the most general and high-level way of security assessment – risk analysis. The formal model we propose explicitly connects various security metrics and indicates how they contribute to the overall assessment.
We can see that all metrics play only a small role when the overall risk is computed. Thus, we conclude that no single metric is enough to predict behavior of the risk value. The only metric which is as general as risk is the attack surface metric, but it relies on many strong assumptions.

We have considered very generic attacker models. In future work, we will consider the behavior of attackers and determine models for computation of probabilities of attack selection. Introducing the behavior of attackers (e.g., adapting Dolev-Yao model for assessment of systems) will enhance our attacker model and will allow us to analyse different strategies of attackers. The probability of attack selection is often left out of the scope of existing approaches. We plan to fill this gap.

4.3 A general method for assessment of security in complex services

A rapidly changing world requires rapidly changing solutions. This is one of the reasons why service-oriented technologies (Grid, Web Services, Cloud) become so popular. The idea behind such technologies is to be agile, easily reconfigurable, and provide different alternatives to fulfill the same goal. Thus, the service consumer is able to select the alternative she likes the most, i.e., the service which has the most suitable qualities, expressed as Service Level Agreement (SLA).

Security requirements also must be included in the agreement in order to protect valuable assets not only during data transmission, but also during data usage [84, 86]. Usually, security requirements or policies (we use the terms requirements and policies interchangeably) are precisely expressed with the help of metrics, which indicate the quantity of some parameter. For example, the number of successful virus attacks on the service could be less than a certain maximum.

The service consumer is able to select the service which has the best metric values. A problem arises when we have a complex service, a business process, which is composed of several simple services. In this case, we need a way to aggregate the values of simple services for the evaluation of the complex service. Moreover, existing alternatives of the implementation of a business process should be compared and the optimal alternative should be selected. Such an analysis is not only useful for service consumers, but also for the service orchestrator which provide the complex service hiding the implementation details. For example, instead of selecting the most secure implementation, the orchestrator may find the level of protection it is able to guarantee even if some atomic services become unavailable. Finally, the method for the analysis of the composed system should be able to assess the security of the system even if different metrics are used for the evaluation of the individual components. It is useful when services belong to third-parties and are evaluated using different security metrics.

Contributions. The main contribution of our work [22] is a general method for the aggregation of security metrics and for the selection the most secure implementation of a business process. This goal is achieved using algebraic structures called “semirings”. We consider a general business process (a complex service) composed of simple abstract services. An abstract service describes a single job that should be executed as part of the business process. Each abstract service is instantiated to several concrete services. We follow the Business Process Modelling Notation (BPMN) for the composition of the business processes from abstract services. BPMN is a high-level notation and, thus, is suitable for the analysis of high-level security properties. For the security analysis and evaluation of a business process, we transform the business process into a graph. Each abstract service from the business process is represented in the graph as a set of nodes modeling the corresponding concrete services. The edges of the graph represent message flows in the business process. We suppose that each node is weighted with the value of a security metric, which evaluates the security level of the associated concrete service. The shortest path in the graph corresponds to the optimal (the most secure) implementation of the business process.

An example of a transformation from a business process into a graph is depicted in Figure 4.1: the business process (an on-line shop) in the top half is transformed into the graph in the bottom half of the figure. Here, \( n_{ij} \) is a \( j^{th} \) concrete service corresponding to the \( i^{th} \) abstract service (e.g., we have two possible implementations of trading platform \( n_{11} \) and \( n_{12} \), \( n_0 \) and \( n_\infty \) correspond to the start and the end points of the business process respectively, \( w_{ij} \) is the weight of the concrete service \( n_{ij} \) (e.g., risk value). The weights of \( n_0 \) and \( n_\infty \) are both equal to 0.
We rely on a general semiring framework \cite{101} for the automated search of the shortest path in weighted graphs. Since the weights represent security metrics with different domains of values and different operations over the domains, classical shortest path search methods (like Dijkstra’s \cite{67}) based on real number operations cannot be applied here. Note that we assume that the security of all concrete services used in a particular business process is assessed using the same security metric. Our second contribution relaxes this assumption.

We have also shown how similar metrics can be combined to conduct a general analysis. This goal is achieved by considering relations between metrics using specific homomorphic mappings between semirings. These mappings allow us to find the set of the most secure implementations of business process or an approximate value for the security level of the business process.

Figure 4.1: Example of transformation of a business process into a graph
5 Interactions with other work packages

WP 9 is a transversal work package, which spans all phases of the SDLC. Accordingly, there are connections with the work packages related to the individual SDLC phases, namely, WP 6 (Security requirements for services), WP 7 (Secure service architectures and design), and WP 8 (Programming environments for secure and composable services) as well as with the second transversal work package, WP 10 (Risk and cost aware SDLC), and the case studies proposed in WP 11 (Future Internet application scenarios). In this section, we describe current interactions and we uncover potential synergies that may lead to additional interactions and collaborations.

WP 6 – Security requirements for services

Assurance techniques can be used during requirements analysis in order to validate the high-level models produced at this early stage of the SDLC (e.g., for checking the consistency of a set of requirements). Such analysis techniques are covered in Task 6.3.

A concrete application of assurance techniques related to requirements is reported in Deliverable D6.2, where a technique for the refinement of security requirements for services using security adaptation contracts is described. In this approach, a functional service adaptor is first synthesized based on the adaptation contract. Next, the adaptor’s security is verified using partial model checking techniques. Finally, any attacks found in this process are used to refine the functional adaptor into a secure one. This work is related to Extended Model Checking covered in Task 9.1.3.

WP 7 – Secure service architectures and design

There is a strong interaction between Task 7.2.3 and Task 9.3.2. In these tasks we study the transformation of high-level security policies and enforcement mechanisms considered in model-based security into lower level policies that are enforceable at the implementation level of executable code considered in language-based security.

There are also connections with Task 7.3 on the composition and adaptation of security concerns. First, runtime reconfiguration techniques to prevent or repair security flaws can leverage runtime verification techniques from WP 9. Second, contract-based service adaptation uses partial model checking, which is related to Extended Model Checking (Task 9.1.3).

WP 8 – Programming environments for secure and composable services

Runtime monitoring is studied in Tasks 8.2.3 and 8.3.2, in particular for usage control and information flow properties. Sections 3 and 4 of Deliverable D8.2 are devoted to this topic. Roughly speaking, WP 9 focuses primarily on logical and semantical aspects, while WP 8 puts the main emphasis on the required platform infrastructure for Java and BPEL (Task 8.2.3) and on fine-grained execution monitoring (Task 8.3.2). However, implementation aspects also play a role in WP 9, namely, in the study of the correct interleaving of distributed events for efficient monitoring.

Another point of connection is secure programming (Task 9.2.1) and Language support for the Future Internet (Task 8.4). Deliverable D8.2 reports on source code verification, more precisely, on Java and C code verification using the VeriFast tool. Moreover, as part of programming principles and best practices, Task 8.4.2 includes the formal verification of design patterns.

WP 10 – Risk and cost aware SDLC

Feedback from assurance methods serves as input for risk and cost analysis. Assurance methods can generally be expected to have a positive impact on risk and cost reduction. For example, from a qualitative perspective, formal verification and testing reduce the risk of vulnerabilities and efficient debugging techniques (such as differential slicing) reduces maintenance costs. For testing, one may be able to indicate a coverage estimate and use this as quantitative input in a subsequent risk analysis. Security metrics developed in WP 9 may also provide a quantitative input to risk and cost analysis processes. In particular, the work summarized in Section 4.2 studies the link between security metrics and risk analysis.
Vice versa, risk analysis in early stages of the SDLC can identify possible vulnerabilities and attacks that can be used to set the focus for the testing phase (e.g., for the selection of test cases).

**WP 11 – Future Internet application scenarios**

WP 11 provides two case studies, which involve a representative mix of Future Internet features. One case study is in the area of e-Health and the other on smart grids. Each case study is structured into several use cases.

The paper on modeling and verifying service-oriented architectures [17] contains an application case study from the e-Health domain, which is closely related to the “patient consent” use case proposed in WP 11 [28]. We plan further applications of the methods developed in WP 9 to the WP 11 case studies. Generally speaking, we expect the smart grid case study to be more difficult to deal with, since the associated systems are much more dynamic and involve more privacy issues.
6 Conclusions

In this deliverable, we have reported on a set of initial solutions for security assurance for services obtained during the first year of the NESSoS project. Let us therefore briefly reflect on what we have achieved so far and identify remaining gaps and possible foci for the second and third years.

The work done so far in WP 9 covers the large majority of the tasks and activities set out in the NESSoS Description of Work. We record good overall progress on all of the topics addressed. We even observe substantial advancements in a number of areas, including Extended Model Checking (Section 2.3), Debugging (Section 3.1.3), and Runtime Verification (Section 3.2). This progress is documented in a number of top-ranked publications.

As expected many research issues remain to be addressed or deepened in the years to come. In the following, we give a partial selection of important topics for future research (cf. [29]).

Extended Model Checking We need to find more tractable procedures than the currently available ones that are able to uncover attacks based on the flexibility of XML format, such as XML injection attacks. Moreover, decision procedures for equivalence checking should be investigated. These play a crucial role for modeling and verifying privacy properties, guessing attacks, and e-voting protocols.

Testing An important gap identified in the current state-of-the-art for model-based testing of Web applications is the focus on the server-side of the Web applications. In Web 2.0 applications, a large amount of the application functionality is located on the client-side, and browsers become highly complex application development platforms by themselves. Modeling the functionality of the Web browser is important for security analysis.

Moreover, the combination of models for the client-side and server-side of Web applications would enable reasoning about discrepancies that may exist between the interpretation of the same data or functionality on both sides of the application, which is the cause of important vulnerabilities such as cross-site scripting.

Runtime verification A conceptual challenge in monitoring security policies is to bridge the gap between security policies, which are typically formulated at high levels of abstraction, and monitors, which observe system events that are low-level and concrete.

Future research questions in the area of enforcement include the investigation of different enforcement mechanisms in a distributed setting, their automatic generation from declarative specification languages, and, as a long term goal, the development of scalable and flexible system architectures in which the usage of data can be enforced.

With respect to the challenges outlined in the introduction, the important but notoriously difficult modularity challenge $C4$ calls for more attention. The goal here is to decompose a verification or testing problem for a Future Internet system into separate (and hopefully more tractable) problems for the individual services. However, it is well-known that the possible interactions between the service components is a source of vulnerabilities. One way to approach this problem is to investigate sufficiently general conditions where the interactions are harmless and where security properties of a service can be deduced from security properties of its components.
NESSoS WP9 Bibliography


General Bibliography


