Towards Formal Validation of Trust and Security in the Internet of Services

Roberto Carbone[1], Marius Minea[2], Sebastian Alexander Mödersheim[3], Serena Elisa Ponta[4,5], Mathieu Turuani[6], and Luca Viganò[7]

[1] Security & Trust Unit, FBK, Trento, Italy
[2] Institute e-Austria, Timişoara, Romania
[3] DTU, Lyngby, Denmark
[4] SAP Research, Mougins, France
[5] DIST, Università di Genova, Italy
[6] LORIA & INRIA Nancy Grand Est, France
[7] Dipartimento di Informatica, Università di Verona, Italy

Abstract. Service designers and developers, while striving to meet the requirements posed by application scenarios, have a hard time to assess the trust and security impact of an option, a minor change, a combination of functionalities, etc., due to the subtle and unforeseeable situations and behaviors that can arise from this panoply of choices. This often results in the release of flawed products to end-users. This issue can be significantly mitigated by empowering designers and developers with tools that offer easy to use graphical interfaces and notations, while employing established verification techniques to efficiently tackle industrial-size problems. The formal verification of trust and security of the Internet of Services will significantly boost its development and public acceptance.

1 Introduction

The vision of the *Internet of Services (IoS)* entails a major paradigm shift in the way ICT systems and applications are designed, implemented, deployed and consumed: they are no longer the result of programming components in the traditional meaning but are built by composing *services* that are distributed over the network and aggregated and consumed at run-time in a demand-driven, flexible way. In the IoS, services are business functionalities that are designed and implemented by producers, deployed by providers, aggregated by intermediaries and used by consumers. However, the new opportunities opened by the IoS will only materialize if concepts, techniques and tools are provided to ensure security. Deploying services in future network infrastructures entails a wide range of trust and security issues, but solving them is extremely hard since making the service components trustworthy is not sufficient: composing services leads to new, subtle and dangerous, vulnerabilities due to interference between component services and policies, the shared communication layer, and application functionality. Thus, one needs validation of both the service components and their composition into secure service architectures.

J. Domingue et al. (Eds.): Future Internet Assembly, LNCS 6656, pp. 193–207, 2011.

Standard validation technologies, however, do not provide automated support for the discovery of important vulnerabilities and associated exploits that are already plaguing complex web-based security-sensitive applications, and thus severely affect the development of the future internet. Moreover, security validation should be carried out at all phases of the service development process, in particular during the design phase by the service designers themselves or by security analysts that support them in their complex tasks, so as to prevent the production and consumption of already flawed services.

Fortunately, a new generation of analyzers for automated security validation at design time has been recently put forth; this is important not just for the results these analyzers provide, but also because they represent a stepping stone for the development of similar tools for validation at service provision and consumption time, thereby significantly improving the all-round security of the IoS. In this chapter, we give a brief overview of the main scientific and industrial challenges for such verification tools, and the solutions they provide; we also discuss some concrete case studies and success stories, which provide proof of concept. As an actual example, we discuss the main ideas and results of one such rigorous technology: the *AVANTSSAR Validation Platform* (or *AVANTSSAR Platform* for short) is an integrated toolset that has been developed in the context of the AVANTSSAR project (`www.avantssar.eu`, [4]) for the formal specification and automated validation of trust and security of *service-oriented architectures (SOAs)*. This technology, which involves the design of a suitable specification language and is based on a variety of complementary techniques[8], has been tuned and proven on a number of relevant industrial case studies. We also report on our activities in migrating project results to industry and disseminating them to standardization bodies, which will ultimately speed up the development of new network and service infrastructures, enhance their security and robustness, and thus increase the development and public acceptance of the IoS.

We proceed as follows. In Sections 2 and 3, we discuss, respectively, some of the main features of specification languages and automated validation techniques that have been developed for the verification of trust and security of services. In Section 4, we present the AVANTSSAR Platform and the AVANTSSAR Library, and then, in Section 5, we present some case studies and validation success stories, and the migration of results into industrial practice. Section 6 concludes the chapter.

2 Specification Languages

Modeling and reasoning about trust and security of SOAs is complex due to three main characteristics of service orientation.

First, SOAs are *heterogeneous*: their components are built using different technology and run in different environments, yet interact and may interfere with each other.

[8] Such as model checking with constraints, approaches based on SAT (i.e., satisfiability) or SMT (i.e., satisfiability modulo testing), or abstract interpretation.

Second, SOAs are also *distributed* systems, with functionality and resources distributed over several machines or processes. The resulting exponential state-space complexity makes their design and efficient validation difficult, even more so in hostile situations perhaps unforeseen at design time.

Third, SOAs and their security requirements are *continuously evolving*: services may be composed at runtime, agents may join or leave, and client credentials are affected by dynamic changes in security policies (e.g., for incidents or emergencies). Hence, security policies must be regarded as part of the service specification and as first-class objects exchanged and processed by services.

The security properties of SOAs are, moreover, very diverse. The classical data security requirements include confidentiality and authentication/integrity of the communicated data. More elaborate goals are structural properties (which can sometimes be reduced to confidentiality and authentication goals) such as authorization (with respect to a policy), separation or binding of duty, and accountability or non-repudiation. Some applications may also have domain-specific goals (e.g., correct processing of orders). Finally, one may consider liveness properties (under certain fairness conditions), e.g., for a given web service for online shopping one may require that every order will eventually be processed if the intruder cannot block the communication indefinitely. This diversity of goals cannot be formulated with a fixed repertoire of generic properties (like authentication); instead, it suggests the need for specification of properties in an expressive logic.

Various languages have been proposed to model trust and security of SOAs, e.g., BPEL [24], π calculus [19], F# [5], to name a few. Each of them, however, focuses only on some aspects of SOAs, and cannot cover all previously described features, except perhaps in an artificial way. One needs a language fully dedicated to specifying trust and security aspects of services, their composition, the properties that they should satisfy and the policies they manipulate and abide by. Moreover, the language must go beyond static service structure: a key challenge is to integrate policies that are dynamic (e.g., changing with the workflow context) with services that can be added and composed dynamically themselves.

As a concrete solution, in the AVANTSSAR project, we have defined a language, the *AVANTSSAR Specification Language ASLan*, that is both expressive enough that many high-level languages, such as BPEL, can be translated to it, and amenable to formal analysis.[9] A key feature of ASLan is the integration of *Horn clauses* that are used to describe policies in a clear, logical way, with a *transition system* that expresses the dynamics of the system, e.g., agents can become members of a group or leave it, with immediate consequences for their access rights.

[9] The AVANTSSAR Platform allows users also to input their services by specifying them using the high-level formal specification language ASLan++, which we have defined to be close to specification languages for security protocols/services and to procedural and object-oriented programming languages. The semantics of ASLan++ is formally defined by translation to ASLan.

As a simple, general (i.e., not AVANTSSAR/ASLan specific) example, consider the policies that a user U has access to a file F if U belongs to a group G that is the owner of F, or U is the deputy of a user that has access to F:

$$access(U, F) \leftarrow member(U, G) \wedge owner(G, F)$$
$$access(U, F) \leftarrow deputy(U, U') \wedge access(U', F)$$

Policies are dynamic, since facts like *member*, *owner*, and *deputy* can change over time, which in turn affects *access* rights. For instance, if user *Alice* changes to another group within the organization, she will immediately obtain all access rights to files of the new group, but lose access rights to files of her old group, except for those that she maintains due to her being a deputy for other users.

We consider transition systems in which a state is a set of facts like *member*, *owner*, etc.; they can be used to describe service workflows and steps in security protocols. For instance, an employee (*Alice*) changing group membership at the command of her manager (*Peter*) can be formalized as:

$$member(Alice, g_1) \wedge isManager(Peter, Alice) \wedge canAssign(Peter, g_3) \Rightarrow$$
$$member(Alice, g_3) \wedge isManager(Peter, Alice) \wedge canAssign(Peter, g_3)$$

The above transition is applicable in a state that includes all facts on the left hand side. When the transition is applied, *Alice*'s membership to g_1 is removed, she is added as member to g_3, and other facts are preserved.

We can now integrate policies with the dynamic aspects of transition systems by defining: the set of facts that hold true in a state is the least closure of the state under all Horn clauses. For example, if a state contains the facts

$$member(a, g_1), member(b, g_2), owner(g_1, f_1), owner(g_2, f_2), deputy(a, b)$$

then the policy implies the following access rights:

$$access(a, f_1), access(b, f_2), access(a, f_2)$$

The least closure represents a "default deny": if the policies do not imply the access right, it is false. The main difference between policies expressed via Horn clauses and transitions expressed via rewrite rules is that the effects of the policy are inferred in the same state (repeatedly, after each transition), while the effects of a transition are inferred in the next state. Thus, the use of Horn clauses enables a deduction chain which is performed for every state of the transition system. Integrating Horn clauses with transition systems is, of course, a broader concept: next to policies, we can model other consequences of facts that become true.

Finally, we need to model the *security properties*. While this can be done by using different languages, in ASLan we have chosen to employ a variant of linear temporal logic (LTL, e.g. [18,25]), with backwards operators and ASLan facts as propositions. This logic gives us the desired flexibility for the specification of complex goals. As an example, in a system employing the policy described above, we may require a separation of duty property, namely that for privacy

purposes, no agent can access both files f_1 and f_2. This can be expressed in LTL as follows:

$$G(\ (access(U, f_1) \rightarrow \neg F(access(U, f_2))) \wedge (access(U, f_2) \rightarrow \neg F(access(U, f_1)))\ )$$

3 Automated Validation Techniques

Due to the inherent complexity (heterogeneity, distribution and dynamicity) of the Internet of Services, the challenge of validating services and service-oriented applications cannot be addressed simply by scaling up the current generation of formal analysis approaches and tools. Rather, novel and different validation techniques are required to automatically reason about services, their composition, their required security properties and associated policies. In particular, one has to consider the various ways in which component services can be coordinated, and develop new techniques, such as model checking, that allow for compositional validation reflecting this modularity, as well as cope with the complexity problem. Moreover, for the practical use and take-up by industry and standardisation organisations, it is essential that any such verification technique provides a high degree of automation.

An important solution to overcome the complexity of SOAs and the heterogeneous security contexts is to integrate different technologies into a single analysis tool, in such way that they can interact and benefit from each other's features. For instance, the AVANTSSAR Platform comprises three validation backends (CL-AtSe [27], OFMC [23], and SATMC [1]), which are based on different automated deduction techniques operating on the same ASLan input, and which thus provide complementary strengths.

In the following subsections, we discuss these four points — orchestration, model checking of SOAs, compositional reasoning, and abstraction-based validation techniques — in more detail and describe how they have been implemented in the AVANTSSAR Platform.

3.1 Orchestration

Composability, one of the basic principles and design-objectives of SOAs, expresses the need for providing simple scenarios where already available services can be reused to derive new added-value services. In their SOAP incarnation, based on XML messaging and relying on a rich stack of related standards, SOAs provide a flexible yet highly inter-operable solution to describe and implement a variety of e-business scenarios possibly bound to complex security policies.

When security constraints are to be respected, it can be very complex to discover or even to describe composition scenarios. This motivates the introduction of automated solutions to scalable services composition. Two key approaches for composing web services have been considered, which differ by their architecture: *orchestration* is centralized and all traffic is routed through a *mediator*, whereas *choreography* is distributed and all web services can communicate directly.

Several "orchestration" notions have been advocated (see, e.g., [20]). However, in inter-organizational business processes it is crucial to protect sensitive data of each organization; and our main motivation is to take into account the security policies while computing an orchestration. The AVANTSSAR Platform, for example, implements an idea presented in [11] to automatically generate a *mediator*. We specify a web service profile from its *XML Schema* and *WS-SecurityPolicy* using first-order terms (including cryptographic functions). The *mediator* is able to use cryptography to produce new messages, and is constructed with respect to security goals using the techniques we developed for the verification of security protocols.

3.2 Model Checking of SOAs

Model checking [13] is a powerful and automatic technique for verifying concurrent systems. It has been applied widely and successfully in practice to verify digital sequential circuit designs, and, more recently, important results have been obtained for the analysis of security protocols. In the context of SOAs, a model-checking problem is the problem of determining whether a given model — representing the execution of the service under scrutiny in a hostile environment — enjoys the security properties specified by a given formula. As mentioned in Section 2, these security properties can be complex, requiring an expressive logic.

Most model-checking techniques in this context make a number of simplifying assumptions on the service and/or on its execution environment that prevent their applicability in some important cases. For instance, most techniques assume that communication between honest principals is controlled by a Dolev-Yao intruder [17], i.e. a malicious agent capable to overhear, divert, and fake messages. Yet we might be interested in establishing the security of a service that relies on a less insecure channel. In fact, services often rely on transport protocols enjoying some given security properties (e.g. TLS is often used as a unilateral or a bilateral communication authentic and/or confidential channel), and it is thus important to develop model-checking techniques that support reasoning about communication channels enjoying security-relevant properties, such as authenticity, confidentiality, and resilience.

Among general model-checking techniques, *bounded model checking*, by supporting reasoning about LTL formulae, allows one to reason about complex trace-based security properties. In particular, the AVANTSSAR Platform integrates a bounded model-checking technique for SOAs [1] that allows one to express complex security goals that services are expected to meet as well as assumptions on the security offered by the communication channels.

3.3 Channels and Compositional Reasoning

A common feature of SOAs is an organization in *layers*: we may have a layer that provides a secure communication infrastructure between participants, e.g. a virtual private network or a TLS [26] channel, and run applications on top of it, as if the participants were directly connected via tamper-proof lines. It is,

of course, undesirable to verify the entire system as a whole: this can easily be too complex for automated methods, and lacks generality and reuse. In fact, an application that requires a secure connection should not depend on the details of the realization of the secure connection and, vice-versa, a system that establishes a secure channel should be able to run arbitrary protocols over it. Thus, a compositionality result is desired: if components are safe in isolation and satisfy certain properties, then they can be composed into a larger system that is also safe.

Progress has been made in this direction for the parallel (and sequential) composition of protocols [12,15,16], i.e., independently using several protocols over the same communication medium. Moreover, there are first results for the layered compositional reasoning needed for SOAs, namely running an application over a channel [22]. This means verifying that *(i)* a protocol such as TLS indeed provides an authentic and confidential channel, *(ii)* an application system is safe if its communication is routed over a secure channel, and *(iii)* both satisfy certain sufficient conditions (their message formats do not interfere). Here, *(i)* and *(ii)* are verification tasks that the tools can check in isolation, and *(iii)* is a format property that can be checked statically. If *(i)*, *(ii)*, and *(iii)* hold, then we can conclude that running the application over the established channel is also safe.

3.4 Abstract Interpretation

The complexity of SOAs is a major challenge for classical model-checking methods. To cope with this, formal validation approaches often strictly bound all aspects of a system, e.g., the number of service runs that honest agents (and a dishonest agent) can perform. However, one would rather verify a system without such limitations, e.g., no matter how many agents use it in parallel. Hence, methods based on abstract interpretation have recently become increasingly popular [6,7,8,9,14,28]. For instance, TulaFale [6], a tool by Microsoft Research based on ProVerif [7], exploits abstract interpretation for verification of web services that use SOAP messaging, using logical predicates to relate the concrete SOAP messages to a less technical representation that is easier to reason about.

There are two basic ideas here: *(i)* partition the infinite set of constants (representing, for instance, agents, request numbers, or cryptographic keys) into finitely many equivalence classes and to compute on those equivalence classes instead; *(ii)* avoid reasoning about a transition system and rather compute an over-approximation of everything that will ever hold true. This allows for the use of classical automated first-order reasoning techniques, in particular resolution or fixed-point computations of static analysis. Thanks to the over-approximation, these systems completely avoid the state-explosion problems of model checking and can analyze systems without bounds on the number of runs that agents participate in. On the downside, the over-approximation can introduce false positives, i.e., attacks introduced by the over-approximation while the actual system is safe.

The ability of abstraction methods to avoid exploration of concrete transition systems has been the reason for their success, but on the other hand also implies

a serious limitation for the verification of complex SOAs. Since there is no notion of time, we cannot model that at some time-point, a key, certificate, access right, or membership is revoked. The *set-based abstraction method* [21] can overcome this limitation while preserving the benefits of abstract interpretation. The idea is to organize data by means of sets and to abstract data by set membership. In the example of Section 2, we may consider the set U_g of users that are currently members of a group g. We can then identify all users that belong to the same set of groups as one abstract equivalence class. The difficult part is how to handle the change of the set memberships, e.g., if a user changes from one group to another. Here, the set-abstraction method defines a mechanism to reason about how facts about one class imply facts about a new class, e.g., roughly, a dishonest user would not delete any information that he has learned in the old group, but he cannot read any new information that the old group produces. Thus, revocation of facts can be modeled without the need to directly express sequencing in time.

4 The AVANTSSAR Platform and Library

We have implemented the AVANTSSAR Platform as a service-oriented architecture. As shown in Fig. 1, the platform includes a *connectors layer*, i.e., a layer of software modules that carry out the translation from application-level specification languages (such as BPMN and BPEL, as well as our own AnB and ASLan++) into ASLan, and vice versa for the platform output. The platform then takes as concrete input a policy stating the functional and security requirements of a goal service and a description of the available services (including a specification of their security-relevant behavior, possibly including the local policies they satisfy) and applies automated reasoning techniques in order to build an orchestration of the available services that meets the security requirements stated in the policy. More specifically, the platform comprises of two main components:

- The *Orchestrator* tries to build an orchestration, i.e., a composition, of the available services in a way that is expected (but not yet guaranteed) to satisfy the input policy. It takes as input an ASLan file with a specification of the available services and either a specification of the client or a partial specification of the goal, and it produces as output an ASLan file with the specification of the available services, a full specification of the goal, and a specification of the client (a putative one, if it was not given as input).
- The *Validator* takes as input an orchestration and a security goal formally specified in ASLan, and automatically checks whether the orchestration meets the security goal. If this is the case, then the ASLan specification of the validated orchestration is given as output, otherwise a counterexample is sent back to the Orchestrator (where a failed validation means the existence of vulnerabilities that need to be fixed).

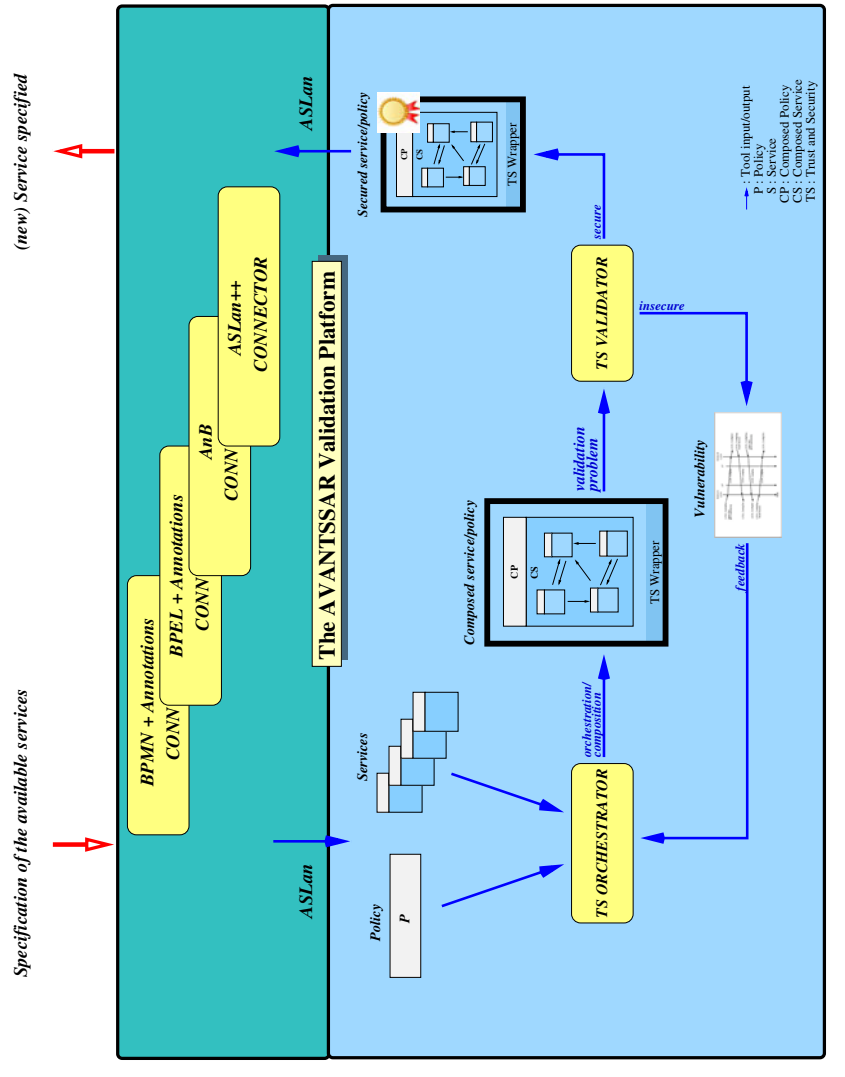

Fig. 1. The AVANTSSAR Validation Platform and its usage towards Enterprise SOA.

As proof of concept, we have applied the AVANTSSAR Platform to the case studies in the *AVANTSSAR Library*, which comprises of the formalization of 10 application scenarios and 94 problem cases of service-oriented architectures from the e-Business, e-Government and e-Health application areas. In this way, we have been able to detect a considerable number of attacks in the considered services and provide the required corrections. Moreover, the formal modeling of case studies has allowed us to consolidate our specification languages and has driven the evolution of the validation platform, both in terms of support for the new language and modeling features, as well as in efficiency improvements needed for the validation of significantly more complex models. We expect that the library will provide a useful test-suite for similar validation technologies.

5 Case Studies, Success Stories, and Industry Migration

The landscape of services that require validation of their security is very broad. The validation is made more difficult by the tension between the need for flexibility, adaptability, and reconfigurability, and the need for simple, understandable, coherent, declarative policies. These must contain all relevant information required to determine the access to private data and to the meta-policies that control them. For example, e-Health practitioners are under pressure to reduce redundant and inefficient behaviors in daily workflows and methods. They must establish repeatable, standards-based solutions that promote a "plug-and-play" approach to context-based information access, making clinical and non-clinical data available anywhere and anytime in a health care organization, while lowering infrastructure costs. Clearly, privacy requirements will be much more difficult to implement and assess in such environments. To ease the analysis, it is necessary to factor out the access control policies and meta-policies from the possible workflow, and to understand and validate the authorization conditions and the security mechanisms that implement them independently of their use in particular workflows. There is thus a clear advantage in having a language allowing the specification of policies via clauses (e.g., Horn clauses) next to the transition system defining the workflow as put forward in Section 2.

Within the AVANTSSAR project, services from a wide variety of application areas have been modeled: banking (loan origination), electronic commerce (anonymous shopping), e-Government (citizen and service portals, public bidding, digital contract signing), and e-Health (electronic health records). Classes of properties that have been verified include authorization policies, accountability, trust management, workflow security, federation and privacy.

A highlight of the effectiveness of the AVANTSSAR methods and tools is the detection of a serious flaw in the SAML-based SSO solution for Google Apps [3]. Though well specified and thoroughly documented, the OASIS SAML security standard is written in natural language that is often subject to interpretation. Since the many configuration options, profiles, protocols, bindings, exceptions, and recommendations are laid out in different, interconnected documents, it is hard to establish which message fields are mandatory in a given

profile and which are not. Moreover, SAML-based solution providers have internal requirements that may result in small deviations from the standard. For instance, internal requirements (or denial-of-service considerations) may lead the service provider to avoid checking the match between the ID field in the AuthResp and in the previously sent AuthReq. What are the consequences of such a choice? The technical overview document provided by OASIS SAML as a non-official addendum increases the clarity in this respect. Still, when Google developed their SAML-based SSO solution for Google Apps they released a flawed product, which allowed a dishonest service provider to impersonate the victim user on Google Apps, granting unauthorized access to private data and services (email, docs, etc.). The vulnerability was detected by the SATMC backend of the AVANTSSAR Platform and the attack was reproduced in an actual deployment of SAML-based SSO for Google Apps. Google and the US Computer Emergency Readiness Team (US-CERT) were informed and the vulnerability was kept confidential until Google developed a new version of the authentication service and Google's customers updated their applications accordingly. The severity of the vulnerability has been rated High in a note issued by the National Institute of Standard and Technology (NIST).

Moreover, as shown in [2], the SATMC backend of the AVANTSSAR Platform also allowed us to detect that the prototypical SAML SSO use case (as described in the SAML technical overview) suffers from an authentication flaw that, under some conditions, allows a malicious service provider to hijack a client authentication attempt and force the latter to access a resource without its consent or intention. It also allows an attacker to launch Cross-Site Scripting (XSS) and Cross-Site Request Forgery attacks (XSRF). This last type of attack is even more pernicious than classic XSS, because XSRF requires the client to have an active session with the service provider, whereas in this case, the session is created automatically hijacking the client's authentication attempt. This may have serious consequences, as witnessed by the new XSS attack identified in the SAML-based SSO for Google Apps and that could have allowed a malicious web server to impersonate a user on any Google application. In [2], solutions that can be used to mitigate and even solve the problem are described. These possible solutions are being discussed with OASIS.

In another notable validation success story, the tool Tookan [10], which is based on SATMC, has automatically found vulnerabilities in PKCS#11-based products by Aladdin, Bull, Gemalto, RSA, and Siemens among others. PKCS# 11 specifies an API for performing cryptographic operations such as encryption and signature using cryptographic tokens (e.g., USB tokens or smart cards). Sensitive cryptographic keys, stored inside the token, should not be revealed to the outside and it should be impossible for an attacker to change those keys. The attacks found show that in many implementations this is not the case: the compromise of a key allows an attacker to clone the token and, more generally, to perform the same security-critical operations as the legitimate token user.

Formal validation of trust and security will become a reality in the Internet of Services only if and when the available technologies will have migrated to industry, as well as to standardization bodies (which are mostly driven by industry

and influence the future of industrial development). Such an industry migration has to face the gap between advanced *formal methods (FM)* techniques and their real exploitation within industry and standardisation bodies. Though the use of FM would promote a more secure development environment, a variety of practical and cultural reasons lead the industrial world to perceive FM approaches as being expensive in terms of time and effort in comparison to the benefits they provide, and difficult to be integrated within industrial processes. In order to ease their adoption, several obstacles have to be overcome, such as: *(i)* the lack of automated FM technology, *(ii)* the gap between the problem case that needs to be solved in industry and the abstract specification provided by FM, and *(iii)* the differences between formal languages and models and those used in industrial design and development environments (e.g., BPMN, Java, ABAP).

The problem is how to make new, efficient methodologies and technologies accessible and readily exploitable, benefitting industry designers and developers. This amounts to migrating the research outcomes of the logical level into the application level by providing a push-button technology so that industry and standardization bodies could check more rapidly the correctness of the proposed solutions without having a strong mathematical background. In particular, industrially suited specification languages (model-driven languages), equipped with easy-to-use GUIs and translators to and from the core formal models should be devised and migrated to the selected development environments.

A concrete example is the industry migration of the AVANTSSAR Platform to the SAP environment. Two valuable migration activities have been carried out by building contacts with core business units. First, in the trail of the successful analysis of Google's SAML-based SSO, an internal project has been run to migrate AVANTSSAR results within SAP NetWeaver Security and Identity Management (SAP NW SIM) with the objective of exploiting the AVANTSSAR technology to initiate a deep formal analysis of the SAP NetWeaver SAML Next Generation Single Sign-On (NW-NGSSO) to formally establish its soundness, i.e., to have formal evidence that the employed service providers and identity provider services fulfill the expected security desiderata in the considered SAP relevant scenarios. This has included the evaluation of those configurations of the highly configurable SAML SSO standard that are relevant for SAP as well as design and development decisions SAP could have taken to fulfill internal customer requirements. More than 50 formal specifications capturing these scenarios, the variety of configuration options, and SAP internal design and implementation choices have been specified. By means of the AVANTSSAR Platform, safe and unsafe configurations for NW-NGSSO for several SAML profiles relevant for SAP have been identified. All discovered risks and flaws in the SAML protocol have been addressed in NW-NGSSO implementation and counter-measures have been taken. The results have been collected in tables that can be used by SAP in setting-up the NW-NGSSO services on customer production systems.

Besides this, the results triggered very valuable discussions in the steering committee that was supervising this internal project. For instance, the authentication flaw in the SAML standard helped SAP business units to get major insights in the SAML standard than the security considerations described in

there and helped SAP Research to better understand the vulnerability itself and to consolidate the results.

The AVANTSSAR technology has been also integrated into the SAP NetWeaver Business Process Management (NW BPM) product to formally validate security-critical aspects of business processes. An eclipse plug-in extension for NW BPM was proposed through the design and development of a security validation plug-in that enables a business process modeler to easily specify the security goals one wishes to validate such as least privilege which can be accomplished by means of the Need-to-Know principle (giving to the users enough rights to perform their job, but no more than that). It also proposes to control the access over automated tasks through the restriction on the invocation and consumption of remote services. A scalability study has also been conducted on a loan origination process case study with a few security goals and on a more complex aviation maintenance process (designed with 70 human activities). The performance analysis helped us to devise a number of optimizations (up to three orders of magnitude).

These results show that the AVANTSSAR technology can provide a high level of assurance within industrial BPM systems, as it allows for validating all the potential execution paths of the BP under-design against the expected security desiderata. In particular, the migration activity succeeded in overcoming obstacles for the adoption of model-checking techniques to validate security desiderata in industry systems by providing an automatic generation of the formal model on which to run the analysis, as well as highlighting the model-checking results as a comprehensive feedback to a business analyst who is neither a model-checking practitioner nor a security expert. As a successful result, the security validation plug-in is currently listed in the productization road-map of SAP products for business process management.

6 Conclusions and Outlook

As exemplified by these case studies and success stories, formal validation technologies can have a decisive impact for the trust and security of the IoS. The research innovation put forth by AVANTSSAR aims at ensuring global security of dynamically composed services and their integration into complex SOAs by developing an integrated platform of automated reasoning techniques and tools. Similar technologies are being developed by other research teams. Together, all these research efforts will result in a new generation of tools for automated security validation at design time, which is a stepping stone for the development of similar tools for validation at service provision and consumption time. These advances will significantly improve the all-round security of the IoS, and thus boost its development and public acceptance.

References

1. Armando, A., Carbone, R., Compagna, L.: LTL Model Checking for Security Protocols. Journal of Applied Non-Classical Logics, special issue on Logic and Information Security, 403–429 (2009)
2. Armando, A., Carbone, R., Compagna, L., Cuéllar, J., Pellegrino, G., Sorniotti, A.: From Multiple Credentials to Browser-based Single Sign-On: Are We More Secure? In: Proceedings of IFIP SEC 2011 (to appear)
3. Armando, A., Carbone, R., Compagna, L., Cuellar, J., Tobarra Abad, L.: Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In: Proceedings of the 6^{th} ACM Workshop on Formal Methods in Security Engineering (FMSE 2008), pp. 1–10. ACM Press, New York (2008)
4. AVANTSSAR: Automated Validation of Trust and Security of Service-Oriented Architectures. FP7-ICT-2007-1, Project No. 216471, `http://www.avantssar.eu`, 01.01.2008–31.12.2010
5. Bhargavan, K., Fournet, C., Gordon, A.D.: Verified Reference Implementations of WS-Security Protocols. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 88–106. Springer, Heidelberg (2006)
6. Bhargavan, K., Fournet, C., Gordon, A.D., Pucella, R.: Tulafale: A security tool for web services. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2003. LNCS, vol. 3188, pp. 197–222. Springer, Heidelberg (2004)
7. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: Proceedings of the 14^{th} IEEE Computer Security Foundations Workshop, pp. 82–96. IEEE Computer Society Press, Los Alamitos (2001)
8. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. Journal of Computer Security 13(3), 347–390 (2005)
9. Boichut, Y., Héam, P.-C., Kouchnarenko, O.: TA4SP (2004), `http://www.univ-orleans.fr/lifo/Members/Yohan.Boichut/ta4sp.html`
10. Bortolozzo, M., Centenaro, M., Focardi, R., Steel, G.: Attacking and Fixing PKCS#11 Security Tokens. In: Proceedings of the 17^{th} ACM conference on Computer and Communications Security (CCS 2010), pp. 260–269. ACM Press, New York (2010)
11. Chevalier, Y., Mekki, M.A., Rusinowitch, M.: Automatic Composition of Services with Security Policies. In: Proceedings of Web Service Composition and Adaptation Workshop (held in conjunction with SCC/SERVICES-2008), pp. 529–537. IEEE Computer Society Press, Los Alamitos (2008)
12. Ciobâca, S., Cortier, V.: Protocol composition for arbitrary primitives. In: Proceedings of 23^{rd} IEEE Computer Security Foundations Symposium, pp. 322–336. IEEE Computer Society Press, Los Alamitos (2010)
13. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
14. Comon-Lundh, H., Cortier, V.: New decidability results for fragments of first-order logic and application to cryptographic protocols. Technical Report LSV-03-3, Laboratoire Specification and Verification, ENS de Cachan, France (2003)
15. Cortier, V., Delaune, S.: Safely composing security protocols. Formal Methods in System Design 34(1), 1–36 (2009)
16. Datta, A., Derek, A., Mitchell, J., Pavlovic, D.: Secure protocol composition. In: Proceedings of the 19^{th} MFPS, ENTCS 83, Elsevier, Amsterdam (2004)

17. Dolev, D., Yao, A.: On the Security of Public-Key Protocols. IEEE Transactions on Information Theory 2(29) (1983)
18. Hodkinson, I., Reynolds, M.: Temporal Logic. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic, pp. 655–720. Elsevier, Amsterdam (2006)
19. Lucchi, R., Mazzara, M.: A pi-calculus based semantics for WS-BPEL. Journal of Logic and Algebraic Programming 70(1), 96–118 (2007)
20. Marconi, A., Pistore, M.: Synthesis and Composition of Web Services. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 89–157. Springer, Heidelberg (2009)
21. Mödersheim, S.: Abstraction by Set-Membership — Verifying Security Protocols and Web Services with Databases. In: Proceedings of 17^{th} ACM conference on Computer and Communications Security (CCS 2010), pp. 351–360. ACM Press, New York (2010)
22. Mödersheim, S., Viganò, L.: Secure Pseudonymous Channels. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 337–354. Springer, Heidelberg (2009)
23. Mödersheim, S., Viganò, L.: The Open-Source Fixed-Point Model Checker for Symbolic Analysis of Security Protocols. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD 2007/2008/2009. LNCS, vol. 5705, pp. 166–194. Springer, Heidelberg (2009)
24. Oasis Consortium. Web Services Business Process Execution Language vers. 2.0 (2007), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`
25. Pnueli, A.: The Temporal Logic of Programs. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pp. 46–57. IEEE Computer Society Press, Los Alamitos (1977)
26. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol, Version 1.2. IETF RFC 5246 (Aug. 2008)
27. Turuani, M.: The CL-Atse Protocol Analyser. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 277–286. Springer, Heidelberg (2006)
28. Weidenbach, C., Afshordel, B., Brahm, U., Cohrs, C., Engel, T., Keen, E., Theobalt, C., Topic, D.: System Description: Version 1.0.0. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 378–382. Springer, Heidelberg (1999)