The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures

Alessandro Armando[1], Wihem Arsac[2], Tigran Avanesov[3], Michele Barletta[4], Alberto Calvi[4], Alessandro Cappai[1], Roberto Carbone[1], Yannick Chevalier[5], Luca Compagna[2], Jorge Cuéllar[6], Gabriel Erzse[7], Simone Frau[8], Marius Minea[7], Sebastian Mödersheim[9], David von Oheimb[6], Giancarlo Pellegrino[2], Serena Elisa Ponta[1,2], Marco Rocchetto[4], Michael Rusinowitch[3], Mohammad Torabi Dashti[8], Mathieu Turuani[3], and Luca Viganò[4]

[1] AI-Lab, DIST, Università di Genova, Italy
[2] SAP Research, Mougins, France
[3] LORIA & INRIA Nancy Grand Est, France
[4] Department of Computer Science, University of Verona, Italy
[5] IRIT, Université Paul Sabatier, France
[6] Siemens AG, Corporate Technology, Munich, Germany
[7] Institute e-Austria and Politehnica University, Timişoara, Romania
[8] Institute of Information Security, ETH Zurich, Switzerland
[9] IBM Zurich Research Laboratory, Switzerland and DTU, Lyngby, Denmark
www.avantssar.eu

Abstract. The AVANTSSAR Platform is an integrated toolset for the formal specification and automated validation of trust and security of service-oriented architectures and other applications in the Internet of Services. The platform supports application-level specification languages (such as BPMN and our custom languages) and features three validation backends (CL-AtSe, OFMC, and SATMC), which provide a range of complementary automated reasoning techniques (including service orchestration, compositional reasoning, model checking, and abstract interpretation). We have applied the platform to a large number of industrial case studies, collected into the AVANTSSAR Library of validated problem cases. In doing so, we unveiled a number of problems and vulnerabilities in deployed services. These include, most notably, a serious flaw in the SAML-based Single Sign-On for Google Apps (now corrected by Google as a result of our findings). We also report on the migration of the platform to industry.

1 Introduction

Driven by rapidly changing requirements and business needs, IT systems and applications are undergoing a paradigm shift: components are replaced by services distributed over the network, and composed and reconfigured dynamically in a demand-driven way into *Service-Oriented Architectures (SOAs)*.

C. Flanagan and B. König (Eds.): TACAS 2012, LNCS 7214, pp. 267–282, 2012.

Deploying services in future network infrastructures such as SOAs or, even more generally, the Internet of Services (IoS), obviously entails a wide range of trust and security issues. Modeling and reasoning about these trust and security issues 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. 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 *trust and security properties* that SOAs should provide to the users are, moreover, very diverse in type and scope, ranging from basic properties like confidentiality and authentication to complex dynamic and domain-specific requirements (e.g., non-repudiation or separation and binding of duty).

In this paper, we present the AVANTSSAR Platform, an integrated toolset for the formal specification and automated validation of trust and security of SOAs and, in general, of applications in the IoS. It has been developed in the context of the FP7 project "AVANTSSAR: Automated Validation of Trust and Security in Service-Oriented Architectures".

To handle the complexity of trust and security in service orientation, the platform integrates different technologies into a single tool, so they can interact and benefit from each other. More specifically, the platform comprises three *back-ends* (CL-AtSe [6,36], OFMC [15,22,33], and SATMC [3,5]), which operate on the same input specification (written in the *AVANTSSAR Specification Language ASLan*) and provide complementary automated reasoning techniques (including service orchestration and compositional reasoning, model checking, and abstraction-based validation). A *connectors layer* provides an interface to application-level specification languages (such as the standard BPMN, and our custom languages ASLan++, AnB and HLPSL++), which can be translated into the core ASLan language and vice versa.

We have applied the platform to a large number of exemplary industrial case studies, which we have collected into the *AVANTSSAR Library* of validated problem cases. In doing so, we have been able to uncover a number of problems and vulnerabilities in deployed services including, most notably, the detection of a serious flaw in the SAML-based SSO solution for Google Apps. Finally, we also report on our successful activities in migrating the platform to industry. As we describe in more detail in the following, to the best of our knowledge, no other tool exhibits the same scope and expressiveness while achieving the same performance and scalability.

We have implemented the AVANTSSAR Platform as a SOA itself, where each component service is offered as a web service. The platform also has a web-based graphical interface that allows the user to choose between three interaction modes of increasing level of sophistication and to execute, monitor and inspect the results of the platform in a user-friendly way. The web services and the associated documentation (a tutorial, guidelines, the Library and other examples, scientific papers and deliverables, and a users mailing list) are available at `www.avantssar.eu`, where one can also download the binaries and/or source code of the validation back-ends and play online with the platform through a prototype, web-based graphical user interface.

The platform is a successor to the AVISPA Tool [2], a push-button tool for the formal analysis of security protocols. The AVANTSSAR Platform significantly extends its predecessor's scope, effectiveness, and performance by scaling up to the trust and security of SOAs and the IoS. We thus expect that the AVANTSSAR Platform will inherit and considerably widen the user basis of AVISPA, which already comprises not only the members of the AVANTSSAR consortium but also several dozens of other academic and industrial practitioners, who have published a large number of works in which AVISPA is used. Our first, and positive, experience with the integration of the AVANTSSAR Platform within industrial practice indicates a strong potential for its wide take up.

It is important to note that this is the first comprehensive description of the platform, including the results of the experiments that we carried out. Descriptions of some of the different platform components have already been given and we will often refer to the corresponding documents for additional information.

2 The AVANTSSAR Platform

2.1 Description and Architecture

Fig. 1 shows the main components of the AVANTSSAR Platform, where the arrows represent the most general information flow, from input specification to validated output. In this flow, the platform takes as input specifications of the available services (including their security-relevant behavior and possibly the local policies they satisfy) together with a policy stating the functional and security requirements of the target service. In the orchestration phase, the platform applies automated reasoning techniques to build a validated orchestration of the available services that meets the security requirements. More specifically, the *Orchestrator* (short for Trust and Security Orchestrator) looks for a composition of the available services in a way that is expected but not yet guaranteed to satisfy the input policy (it may additionally receive as input a counterexample found by the Validator, if any) and outputs a specification of the target service that is guaranteed to satisfy the functional goals. Then, the *Validator* (short for Trust and Security Validator), which comprises the three back-ends CL-AtSe, OFMC and SATMC, checks whether the orchestration satisfies the security goals. If so, the orchestrated service is returned as output, otherwise, a counterexample is returned to the Orchestrator to provide a different orchestration, until it succeeds,

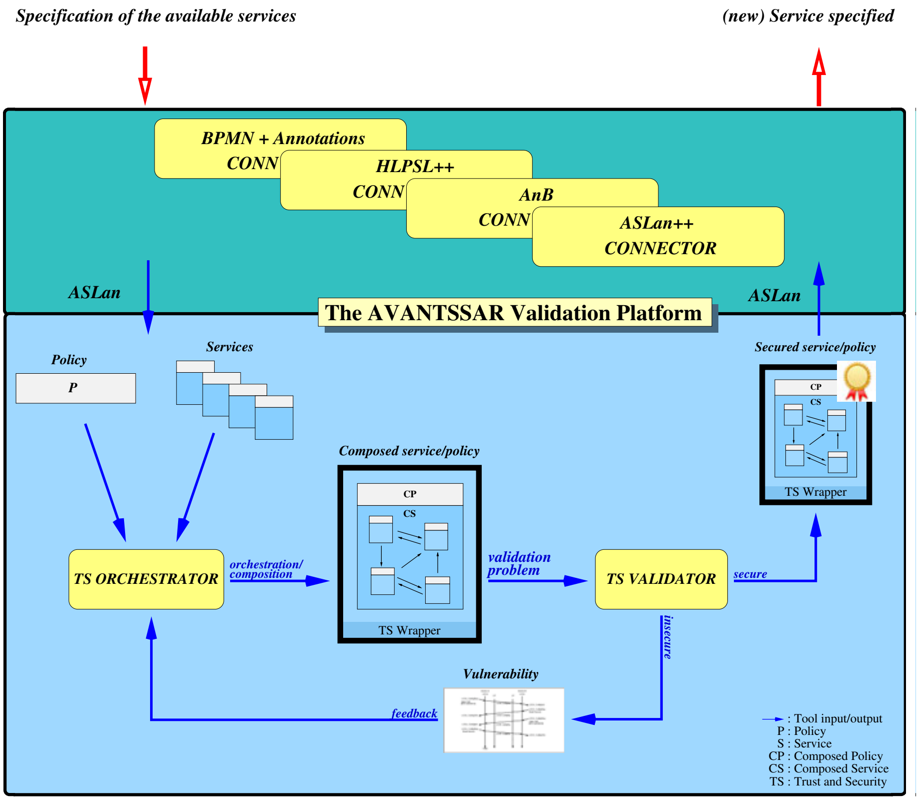

Fig. 1. The AVANTSSAR Validation Platform

or no suitable orchestration can be found. Instead of using the Orchestrator, a user may manually generate the target service and simply invoke the Validator, providing as input the service and its security goals. In this case, the platform outputs either validation success or the counterexample found.

To ease its usage and pave the way for its adoption by industry, the *connectors layer* of the platform provides a set of software modules that carry out both

(C1) the translation from application-level (e.g., our own ASLan++, AnB and HLPSL++) and industrially-suited specification languages (e.g., BPMN) into the low-level *AVANTSSAR Specification Language (ASLan)* [9], the common input language of formal analysis by the validator back-ends, and
(C2) the reverse translation from the common output format of the validator back-ends into a higher-level MSC-like output format to ease the interpretation of the results for the user.

Moreover, the connectors layer is open to the integration of other translations.

In the following subsections, we describe the different platform components in more detail, starting with the specification languages and the connectors layer, and then considering the Orchestrator and the Validator.

2.2 The Specification Languages ASLan and ASLan++

As observed in the introduction, modeling and reasoning about trust and security of SOAs is complex due to the fact that SOAs are heterogeneous, distributed and continuously evolving, and should guarantee security properties that are, typically, very diverse. Besides the classical data security requirements including confidentiality and authentication/integrity, more elaborate goals are 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., one may require that a web service for online shopping eventually processes every order 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 [34], π calculus [28], F# [17], 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.

We have designed ASLan so as to satisfy all these desiderata. At its core, ASLan describes a *transition system*, where states are sets of typed ground terms (facts), and transitions are specified as rewriting rules over sets of terms. A fact *iknows*, true of any message (term) known to the intruder, is used to model communication as we consider a general *Dolev-Yao intruder* [26] that is in complete control of the network and can compose, send, and intercept messages at will, yet cannot break cryptography (following the perfect cryptography assumption). A key feature of ASLan is the integration of this transition system that expresses the dynamics of the model with *Horn clauses*, which are used to describe policies in a clear, logical way. The execution model alternates transition steps with a transitive closure of Horn clause applications. This allows us to model the effects of policies in different states: for instance, agents can become members of a group or leave it, with immediate consequences for their access rights.

Moreover, to carry out the formal analysis of services, we need to model the *security goals*. 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. [27]), with backwards operators and ASLan facts as propositions. This logic gives us the desired flexibility for the specification of complex goals, as illustrated by the problem cases that are part of the AVANTSSAR Library.

ASLan is a low-level formal language and is thus easily usable only by experts, so we have developed the higher-level language ASLan++ to achieve three main design goals:

- the language should be expressive enough to model a wide range of SOAs while allowing for succinct specifications;
- it should facilitate the specification of services at a high level of abstraction in order to reduce model complexity as much as possible; and
- it should be close to specification languages for security protocols and web services, but also to procedural and object-oriented programming languages, so that it can be employed by users who are not formal specification experts.

For reasons of space, we refer to [13,37] for details on ASLan and ASLan++ including a tutorial with many modeling examples.

2.3 The Connectors Layer

As remarked above, writing formal specifications of complex systems at the low conceptual level of ASLan is not practically feasible and reasonable. The same applies to the activity of interpreting and understanding the raw output format returned by the validator back-ends. Industry, in particular, is used to higher-level modeling languages typically targeting very specific domain areas. That is why we have devised an open connectors layer, which currently comprises four connectors carrying out automatic translations.

The *ASLan++ connector* provides translations from ASLan++ specifications to ASLan and in the reverse direction for attack traces. Security protocol/service practitioners who are used to the more accessible but less expressive Alice-and-Bob notation or message sequence charts (MSCs) may prefer to use the *AnB connector*, which is based on an extended Alice-and-Bob notation [30,32,33], or the *HLPSL++ connector*, which is based on an extension of the High-Level Protocol Specification Language HLPSL [23], developed in the context of the AVISPA project [2,14].

Business process (BP) practitioners are used to standard languages such as the Business Process Modeling Notation (*BPMN*), the Business Process Execution Language (*BPEL*), etc. For them, even the usage of ASLan++ (or AnB or HLPSL++) may not be so easy, or they might already have specifications written in their favorite BP language that they do not wish to put aside to then repeat the modeling activity with another language. We have thus developed two connectors for BPMN (see [12]): a public connector that can be used in open-source environments such as Oryx to evaluate control flow properties of a BP modeled in BPMN, and a proprietary SAP NetWeaver BPM connector that is a plug-in of the SAP NetWeaver Development Studio that allows BP analysts to take advantage of the AVANTSSAR Platform via a security validation service. The business analyst provides the security requirements that are critical for the compliance of the BP (e.g., need-to-know in executing a task, data confidentiality with respect to certain users or roles) through easy-to-access UIs of the security validator that returns answers in a nice graphical BPMN-like format.

Connectors for other BP languages may be developed similarly. In fact, thanks to the openness of the connectors layer, new connectors for other application level and/or industrially-suited specification languages can be added by creating proper software modules implementing (C1) and (C2). To alleviate this task, we have devised, for both the common input language ASLan and the common output format of the validator back-ends, XML representations and software modules generating these XML representations [10].

2.4 The Orchestrator

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.

It can be very complex to discover or even to adequately describe composition scenarios respecting overall security constraints. This motivates introducing 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., [29]). However, in inter-organizational BPs 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 implements an idea presented in [24] 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 can 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.

We highlight here the most important distinguishing features of our approach. First, several tools have addressed the WS orchestration problem but, to our knowledge, previous works abstract away the security policies attached to the services, while we consider them as an additional constraint. Second, most automatic orchestration approaches work by computing products of (communicating) finite-state automata, where messages are restricted to a finite alphabet. However, by specifying web services in ASLan, we can express a richer set of messages using first-order terms (including symbols for cryptographic functions). Third, we have applied the AVANTSSAR Orchestrator to several industrial case studies (cf. Table 1) that cannot be handled by other tools because the messages exchanged by services are too complex (e.g., they are non-atomic and built with cryptographic primitives) and require some automatic adaptation. For example, the Orchestrator has automatically generated a Security Server in the Digital Contract Signing case study (which originated from a commercial product), while in the Car Registration Process case study, the Orchestrator has been able to cope with additional constraints imposed by the authorization policies of the available services, specified as a set of Horn clauses.

Finally, and most importantly, the orchestration output can be automatically checked for security by the Validator as described below. If the specification meets the validation goals, i.e., no attack is found, the orchestration solution is considered as the final, validated, result of orchestration. Otherwise the Validator returns a goal violation report including an attack trace, which may be fed back to the Orchestrator, requesting it to backtrack and try an alternative solution.

2.5 The Validator

A specification in ASLan may be the result of an orchestration or of the translation of a specification given in some higher-level language such as ASLan++. The Validator takes any ASLan model of a system and its security goals and automatically checks whether the system meets its goals under the assumption that the network is controlled by a Dolev-Yao intruder.

Currently, the functionality of the Validator is supported by the three different back-ends CL-AtSe, OFMC and SATMC, but, again, the platform is open to the integration of additional validation back-ends.

The user can select which back-end is used for the validation process. By default, all three are invoked in parallel on the same input specification, so that the user can compare the results of the validation carried out by the complementary automated reasoning techniques that the back-ends provide (including compositional reasoning, model checking, and abstract interpretation).

CL-AtSe. The *Constraint-Logic-based Attack Searcher* for security protocols and services takes as input a service specified as a set of rewriting rules, and applies rewriting and constraint solving techniques to model all states that are reachable by the participants and decides if an attack exists with respect to the Dolev-Yao intruder. The main idea in CL-AtSe consists in running the services in all possible ways by representing families of traces with positive or negative constraints on the intruder knowledge, variable values or sets, etc. Each service step execution adds new constraints on the current intruder and environment state. Constraints are kept reduced to a normal form for which satisfiability is easily checked. This allows one to decide whether some security property has been violated up to this point. CL-AtSe requires a bound on the number of service calls in case the specification allows for loops in system execution. It implements several preprocessing modules to simplify and optimize input specifications before starting a verification. If a security property is violated then CL-AtSe outputs a trace that gives a detailed account of the attack scenario.

OFMC. The *Open-source Fixedpoint Model Checker* (which extends the *On-the-fly model checker*, the previous OFMC) consists of two modules. The *classical module* performs verification for a bounded number of transitions of honest agents using a constraint-based representation of the intruder behavior. The *fixedpoint module* allows verification without restricting the number of steps by working on an over-approximation of the search space that is specified by a set of

Horn clauses using abstract interpretation techniques and counterexample-based refinement of abstractions. Running both modules in parallel, OFMC stops as soon as the classic module has found an attack or the fixedpoint module has verified the specification, so as soon as there is a definitive result. Otherwise, OFMC can just report the bounded verification results and the potential attacks that the fixedpoint module has found. In case of a positive result, we can use the computed fixedpoint to automatically generate a proof certificate for the Isabelle interactive theorem prover. The idea behind the automatic proof generator OFMC/Isabelle [22] is to gain a high reliability, since after this step the correctness of the verification result no longer depends on the correctness of OFMC and the correct use of abstractions. Rather, it only relies on: (i) the correctness of the small Isabelle core that checks the proof generated by OFMC/Isabelle, and (ii) that the original ASLan specification (without over-approximations) indeed faithfully models the system and properties that are to be verified.

SATMC. The *SAT-based Model Checker* is an open, flexible platform for SAT-based bounded model checking of security services. Under the standard assumption of strong typing, SATMC performs a bounded analysis of the problem by considering scenarios with a finite number of sessions. At the core of SATMC lies a procedure that, given a security problem, automatically generates a propositional formula whose satisfying assignments (if any) correspond to counterexamples on the security problem of length bounded by some integer k. Intuitively, the formula represents all the possible evolutions, up to depth k, of the transition system described by the security problem. Finding attacks (of length k) on the service therefore reduces to solving propositional satisfiability problems. For this task, SATMC relies on state-of-the-art SAT solvers, which can handle propositional satisfiability problems with hundreds of thousands of variables and clauses or more. SATMC can also be instructed to perform an iterative deepening on the number k of steps. As soon as a satisfiable formula is found, the corresponding model is translated back into a *partial-order plan* (i.e., a partially ordered set of rules whose applications lead the system from the initial state to a state witnessing the violation of the expected security property).

As we remarked above, to the best of our knowledge, no other tool exhibits the same scope and expressiveness while achieving the same performance and scalability of the AVANTSSAR Platform. We have already discussed the expressiveness of the AVANTSSAR languages and the possibility of carrying out automated orchestration under security constraints, so now we briefly describe related work on automated analysis (and then discuss industrial case studies and industry migration in the following sections).

Service analysis methods based on abstract interpretation have become increasingly popular, e.g., [16,18,19,20,25,38]. For instance, TulaFale [16], a tool by Microsoft Research based on ProVerif [18], 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. ProVerif implements a form of static analysis based on abstract interpretation that supports unbounded verification but does not support the modeling of many aspects that occur in problems of real-world complexity such as revocation of keys at a key-server. In contrast, the AVANTSSAR Platform supports the formal modeling and automatic analysis of a large class of systems and properties, albeit for a bounded number of sessions. Two recent tools, namely the AIF framework [31] and StatVerif [1], have overcome some of the limitations of ProVerif, but they do not (yet) cover the full scope of what is specifiable and analyzable with the AVANTSSAR Platform.

2.6 The AVANTSSAR Platform: Web Services and Web Interface

We have implemented the AVANTSSAR Platform as a SOA itself, where each component service is offered as a web service (the URLs where each service, and its WSDL interface, can be accessed are given at `www.avantssar.eu`; binaries of each platform component are also available there, together with the source codes of OFMC and SATMC). The platform service is implemented in PHP5, by using the WSO2 Web Services Framework for PHP (WSO2 WSF/PHP) [39], an open source, enterprise grade, PHP extension for providing and consuming Web Services in PHP. The framework provides base communication functionality in SOAP, XML, and other message formats carried over various transports including HTTP, SMTP, XMPP and TCP. SOAP and HTTP are the standards used for the current Web Services implementation.

The platform also comes with a web-based graphical user interface that allows the user to execute, monitor and inspect the results of the platform in a user-friendly way. Scalable vector graphics and AJAX are suitably coupled to provide the user with an enhanced user experience. Fig. 2 shows a screenshot of the interface. Since the number of functionalities offered by the platform can discourage newcomers, the web interface supports three interaction modes with increasing level of sophistication: demo mode, basic mode, and expert mode.

3 AVANTSSAR Library and Experimental Results

As proof of concept, we have applied the AVANTSSAR Platform to the case studies that are now part of the so-called *AVANTSSAR Library*. In this way, we have been able to detect a considerable number of goal violations 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 platform, both in terms of support for the new language and modeling features, as well as in efficiency improvements needed for the validation of the significantly more complex models. We expect that the library will provide a useful test suite for similar validation technologies.

As terminology, we say that an *application scenario* is composed of one or more *scenes* that focus on different use cases of the considered system, service, protocol, or the like. Each scene contains at least one goal formalizing a desired security property or security aspect, which we call a *problem case*.

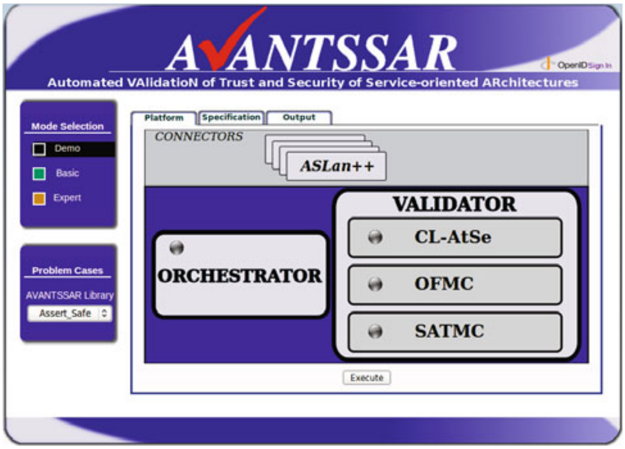

Fig. 2. The web interface of the AVANTSSAR Platform

The AVANTSSAR Library contains the formalization of 10 application scenarios of SOAs from the e-Business, e-Government and e-Health application areas. For these application scenarios we have written 26 specifications (in one of the application-level languages ASLan++, HLPSL++, annotated BPMN, or in the more low level specification language ASLan). Each of these specifications may address different security aspects, for a total of 94 problem cases. Among the 26 specifications, 4 involve orchestration, resulting in 13 problem cases that have to be orchestrated prior to validation.

Table 1 provides an overview of the problem cases formalized and validated by the AVANTSSAR Platform. It contains, for each application scenario, information about the connector used to translate high-level specifications into ASLan (for NW BPM see Section 4) and, if applicable, about the orchestration carried out (column "Orch."). For what concerns the families of problem cases, "f" indicates that a formalization of the problem case is present in the specification but was not validated, whereas "v" indicates its validation. Table 2 describes CPU times spent by each back-end on each application scenario. $S/NS/TOUT$ are abbreviations for *Supported*/*Not Supported*/*Timeout*; times are totals (in seconds) for successful runs. Moreover, for each scenario, the total number of Horn Clauses (HC) and transitions (i.e., ASLan steps) contained in the specifications are shown.

Since we lack space to describe all the application scenarios, problem cases and corresponding trust and security requirements in detail, we point the reader to [11] and here focus only on the SAML Single Sign-On scenario. It is representative for the effectiveness of the AVANTSSAR methods and tools, since we have succeeded in detecting vulnerabilities both in deployed SAML-based SSO solutions and in the use case described in the SAML Technical Overview [35]. Though well specified and thoroughly documented, the OASIS SAML security standard is written in natural language that is often subject to interpretation.

Table 1. The AVANTSSAR Library: formalization and validation status

Areas	Scenarios	Scene	Specification	Connector	Orch.	Problem Cases							
						Federation	Authorization Policies	Accountability	Trust Management	Workflow Security	Privacy	Application Data Protection	Communication Security
E-Business													
Banking Services	Loan Origination	1	lop-scene1.aslan	No	No					v			
		2	lop-scene2.aslan	NW BPM	No		v			v	v		
Electronic Commerce	Anonymous Shopping	1	IDMXScene1_Safe.aslan++	ASLan++	No			v			v		
		2	IDMXScene2_Safe.aslan++	ASLan++	No	v					v		
		3	IDMXScene3_Safe.aslan++	ASLan++	No		f				f		
E-Government													
Citizen and Service Portals	Visa Application	1	PTD_VisaBank.aslan++	ASLan++	No		v	v	v	v	v	v	v
	Car Registration	1	CRP.dyn.aslan++	ASLan++	Yes		v	v	v	v	v	v	v
Document Exchange Procedures	Public Bidding	1	pb_scene1.aslan++	ASLan++	No		f	f				f	f
		2	pb_scene2.aslan++	ASLan++	No		v	v				v	v
		3	pb-elig.aslan++	ASLan++	No		v						
		4	PB_alt.aslan	No	Yes		v	v				v	v
	Digital Contract Signing	1	dcs-scene1.aslan++	ASLan++	No		f	f		f		f	f
		2	dcs-scene2.aslan++	ASLan++	No		v	v		v		v	v
		3	dcs-scene3.aslan++	ASLan++	No		v	v		v		v	v
		4	DCS.ORCH.aslan	No	Yes		f						
		5	DCS-GoalStyleInput.ORCH.aslan	No	Yes		f						
E-Health													
Personal Health Information	Electronic Health Records	1	ECR.aslan++	ASLan++	No			v	v	v	v	v	v
	Process Task Delegation	1	PTD.aslan++	ASLan++	No	v	v			v	v	v	v
		2	PTD_PC.aslan++	ASLan++	No	v	v	v		v	v	v	v
	Access Control Management	1	eHRMS.txt	No	No		f				f	f	
	SAML Single Sign-On	1	SP_init-FC-one_channel.hlpsl++	HLPSL++	No	v					v		v
		2	SP_init-BC-two_channels.hlpsl++	HLPSL++	No	v					v		v
		3	IdP_init-FC.hlpsl++	HLPSL++	No	v					v		v
		4	IdP_init-BC.hlpsl++	HLPSL++	No	v					v		v
		5	SAML-based_SSO_for_GoogleApp.hlpsl++	HLPSL++	No	v					v		v
		6	SAML-based_SSO_for_GoogleApp.aslan++	ASLan++	No	v					v		v

Table 2. CPU analysis times for each back-end on the application scenarios

	Dimensions		SATMC		OFMC		CL-AtSe	
Application Scenario	**HC**	**Steps**	**Time**	**S/NS/TOUT**	**Time**	**S/NS/TOUT**	**Time**	**S/NS/TOUT**
Anonymous Shopping	180	94	0	0/6/0	57.83	2/0/4	5.91	4/0/2
Car Registration	349	258	60.54	7/1/4	1001.31	2/1/9	69.35	10/0/2
Digital Contract Signing	238	52	10504.87	9/5/1	0	0/13/2	906.77	9/0/6
Electronic Health Records	89	48	19.33	1/1/0	5.08	1/0/1	125.37	1/1/0
Loan Origination	303	418	767.80	9/0/0	0	0/9/0	7175.26	6/0/3
Process Task Delegation	90	39	1.68	0/0/2	0	0/2/0	1092.43	2/0/0
Public Bidding	117	631	6747.37	12/0/3	9781.38	8/2/5	9298.7	14/1/0
SAML Single Sign-On	21	215	1989.49	15/0/1	22.77	1/15/0	1.85	1/15/0
Visa Application	38	19	44.83	1/0/0	3.12	1/0/0	9.86	1/0/0
Total	1425	1774	20135.84	52/15/16	10871.49	11/42/30	18685.50	51/17/15

Since the many configuration options, profiles, protocols, bindings, exceptions, and recommendations are laid out in different, interconnected documents, it is not always easy to establish which message fields are mandatory in a given profile and which are not. Moreover, SAML-based solution providers may have internal requirements that may result in small deviations from the standard. For instance, internal requirements (or DoS considerations) may lead the service provider to avoid checking the match between the ID field in the AuthResp and in the previously sent AuthReq. The consequences of such a choice must be examined in detail.

The SAML-based SSO for Google Apps in operation until June 2008 deviated from the standard in a few, seemingly minor ways. By using the AVANTSSAR Platform, we discovered a serious authentication flaw in the service, which a dishonest service provider could use to impersonate the victim user on Google Apps, granting unauthorized access to private data and services (email, docs, etc.) [5]. The vulnerability was detected by SATMC and the attack was reproduced in an actual deployment of SAML-based SSO for Google Apps. We readily informed Google and the US Computer Emergency Readiness Team (US-CERT) of the problem. Google developed a new version of the authentication service and asked their customers to update their applications accordingly. The vulnerability report released by US-CERT is available at `http://www.kb.cert.org/vuls/id/612636`. The severity of the vulnerability has been rated High by the National Institute of Standard and Technology (`http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-3891`).

By using the AVANTSSAR Platform we also discovered an authentication flaw in the prototypical SAML SSO use case (as described in the SAML Technical Overview) [4]. This flaw allows a malicious service provider to hijack a client authentication attempt and force the latter to access a resource without its consent. It also allows an attacker to launch Cross-Site Scripting (XSS) and Cross-Site Request Forgery (XSRF) attacks. This last type of attack is even more pernicious than classic XSRF, 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 that we 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. The problem has been reported to OASIS, and a proposal for an *errata* to the SAML standard is currently being discussed within OASIS (`http://tools.oasis-open.org/issues/browse/SECURITY-12`).

4 Technology Migration

Formal validation of trust and security will become a reality in SOAs and the IoS only if and when the available technologies will have migrated to industry and to standardization bodies (which are mostly driven by industry and influence the future of industrial development). Such a migration has to face the gap between advanced formal methods and their real exploitation within industry and standardization bodies.

To ease the adoption of formal methods, several obstacles have to be overcome, in particular: (i) the lack of automated technology supporting formal methods, (ii) the gap between the problem case that needs to be solved in industry and the abstract specification provided by formal methods, and (iii) the differences between formal languages and models and the languages used in industrial design and development environments (e.g., BPMN, Java, ABAP).

AVANTSSAR has addressed these issues by devising industrially-suited specification languages (model-driven languages), equipped with easy-to-use GUIs and translators to and from the core formal models, and migrating them to the selected development environments. This enables designers and developers from industry and standardization bodies to check more rapidly the correctness of the proposed solutions without having a strong mathematical background.

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, the AVANTSSAR Platform has been exploited to formally validate relevant scenarios where the SAP NetWeaver SAML Next Generation Single Sign On services (NW NG SSO) are employed. More than 50 formal specifications capturing these scenarios, the variety of configuration options, and SAP internal design and implementation choices have been formalized. Unsafe service compositions and configurations have been detected, and safe compositions and configurations have been put forward for use by SAP in setting up the NW NG SSO services on customer production systems.

The AVANTSSAR technology has been also integrated via a plug-in into the SAP NetWeaver BPM (NW BPM) product [7,8] to formally validate if a business process together with its access control policy complies with security-critical requirements, e.g., separation and binding of duty, need-to-know principle, etc. The plug-in provides a push-button technology with accessible user interfaces, bridging the gap between business process modeling languages and formal specifications. Thus, a BP modeler can easily specify the security goals to validate against the business process and access control policy; any violation of the security properties is depicted in a graphical way, enabling the modeler to take countermeasures.

5 Concluding Remarks

As exemplified by the case studies and success stories mentioned above, formal validation technologies can have a decisive impact for the trust and security of SOAs and the IoS. The research innovation put forth by the AVANTSSAR Platform 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 (although none has yet the scale and depth of our platform, which is the reason why we could not compare scope and efficiency). Brought together, 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. For instance, part of the AVANTSSAR consortium is developing a security testing toolset in the context of the FP7 project "SPaCIoS: Secure Provision and Consumption in the Internet of Services" (`www.spacios.eu`). These advances will significantly improve the all-round security of SOAs and the IoS, and thus boost their trustworthy development and public acceptance.

References

1. Arapinis, M., Ritter, E., Ryan, M.D.: StatVerif: Verification of Stateful Processes. In: Proc. CSF 2011, pp. 33–47. IEEE CS Press (2011)
2. Armando, A., Basin, D.A., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P.H., Héam, P.-C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
3. Armando, A., Carbone, R., Compagna, L.: LTL Model Checking for Security Protocols. Journal of Applied Non-Classical Logics 19(4), 403–429 (2009)
4. 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: Camenisch, J., Fischer-Hübner, S., Murayama, Y., Portmann, A., Rieder, C. (eds.) SEC 2011. IFIP Advances in Information and Communication Technology, vol. 354, pp. 68–79. Springer, Heidelberg (2011)
5. Armando, A., Carbone, R., Compagna, L., Cuéllar, 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: Proc. FMSE 2008. ACM Press (2008)
6. Arora, C., Turuani, M.: Validating Integrity for the Ephemerizer's Protocol with CL-Atse. In: Cortier, V., Kirchner, C., Okada, M., Sakurada, H. (eds.) Formal to Practical Security. LNCS, vol. 5458, pp. 21–32. Springer, Heidelberg (2009)
7. Arsac, W., Compagna, L., Kaluvuri, S., Ponta, S.E.: Security Validation Tool for Business Processes. In: Proc. SACMAT 2011, pp. 143–144. ACM (2011)
8. Arsac, W., Compagna, L., Pellegrino, G., Ponta, S.E.: Security Validation of Business Processes via Model-Checking. In: Erlingsson, Ú., Wieringa, R., Zannone, N. (eds.) ESSoS 2011. LNCS, vol. 6542, pp. 29–42. Springer, Heidelberg (2011)
9. AVANTSSAR. Deliverable 2.1: Requirements for modelling and ASLan v.1 (2008)
10. AVANTSSAR. Deliverable 4.2: AVANTSSAR Validation Platform v.2 (2010)
11. AVANTSSAR. Deliverable 5.4: Assessment of the AVANTSSAR Validation Platform (2010)
12. AVANTSSAR. Deliverable 6.2.3: Migration to industrial development environments: lessons learned and best practices (2010)
13. AVANTSSAR. Deliverable 2.3: ASLan++ specification and tutorial (2011)
14. AVISPA: Automated Validation of Internet Security Protocols and Applications, `http://www.avispa-project.org`
15. Basin, D., Mödersheim, S., Viganò, L.: OFMC: A symbolic model checker for security protocols. IJIS 4(3), 181–208 (2005)
16. 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)
17. Bhargavan, K., Fournet, C., Gordon, A.: 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)
18. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: Proc. CSFW 2001, pp. 82–96. IEEE CS Press (2001)
19. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Riis Nielson, H.: Automatic validation of protocol narration. In: Proc. CSFW 2003, pp. 126–140. IEEE CS Press (2003)

20. Boichut, Y., Heam, P.-C., Kouchnarenko, O.: TA4SP: Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (2004)
21. Boichut, Y., Heam, P.-C., Kouchnarenko, O., Oehl, F.: Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols. In: Proc. AVIS 2004. ENTCS (2004)
22. Brucker, A., Mödersheim, S.: Integrating Automated and Interactive Protocol Verification. In: Degano, P., Guttman, J.D. (eds.) FAST 2009. LNCS, vol. 5983, pp. 248–262. Springer, Heidelberg (2010)
23. Chevalier, Y., Compagna, L., Cuéllar, J., Hankes Drielsma, P., Mantovani, J., Mödersheim, S., Vigneron, L.: A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In: Proc. SAPS 2004, pp. 193–205 (2004)
24. Chevalier, Y., Mekki, M.A., Rusinowitch, M.: Automatic Composition of Services with Security Policies. In: Proc. WSCA, pp. 529–537. IEEE CS Press (2008)
25. Comon-Lundh, H., Cortier, V.: New Decidability Results for Fragments of First-order Logic and Application to Cryptographic protocols. TR LSV-03-3, Laboratoire Specification and Verification, ENS de Cachan, France (2003)
26. Dolev, D., Yao, A.: On the Security of Public-Key Protocols. IEEE Transactions on Information Theory 2(29) (1983)
27. Hodkinson, I., Reynolds, M.: Temporal Logic. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic, pp. 655–720. Elsevier (2006)
28. Lucchi, R., Mazzara, M.: A pi-calculus based semantics for WS-BPEL. J. Log. Algebr. Program. 70(1), 96–118 (2007)
29. 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)
30. Mödersheim, S.: Algebraic Properties in Alice and Bob Notation. In: Proc. Ares 2009, pp. 433–440. IEEE CS Press (2009)
31. Mödersheim, S.: Abstraction by Set-Membership: Verifying Security Protocols and Web Services with Databases. In: Proc. CCS 17, pp. 351–360. ACM Press (2010)
32. Mödersheim, S., Viganò, L.: Secure Pseudonymous Channels. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 337–354. Springer, Heidelberg (2009)
33. Mödersheim, S., Viganò, L.: The Open-source Fixed-point Model Checker for Symbolic Analysis of Security Protocols. In: Aldini, A., Barth, G., Gorrieri, R. (eds.) FOSAD 2007. LNCS, vol. 5705, pp. 166–194. Springer, Heidelberg (2009)
34. OASIS. Web Services Business Process Execution Language Version 2.0. (April 11, 2007), http://docs.asis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf
35. OASIS. SAML v2.0 – Technical Overview (March 2007), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
36. Turuani, M.: The CL-Atse Protocol Analyser. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 277–286. Springer, Heidelberg (2006)
37. von Oheimb, D., Mödersheim, S.: ASLan++ — A Formal Security Specification Language for Distributed Systems. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 1–22. Springer, Heidelberg (2011)
38. Weidenbach, C.: Towards an Automatic Analysis of Security Protocols in First-Order Logic. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 314–328. Springer, Heidelberg (1999)
39. WSO2. Web Services Framework for PHP (2006), http://wso2.org/projects/wsf/php