Automatic Security Analysis of SAML-based Single Sign-On Protocols

Alessandro Armando
DIST, University of Genova
Viale Causa 13
16145, Genova, Italy
armando@dist.unige.it

Giancarlo Pellegrino
SAP Research, SAP Labs France SAS
805, avenue du Dr. Maurice Donat
06254 Mougins Cedex
giancarlo.pellegrino@sap.com

Roberto Carbone
DIST, University of Genova
Viale Causa 13
16145, Genova, Italy
carbone@dist.unige.it

Luca Compagna
SAP Research, SAP Labs France SAS
805, avenue du Dr. Maurice Donat
06254 Mougins Cedex
luca.compagna@sap.com

ABSTRACT

Single-Sign-On (SSO) protocols enable companies to establish a federated environment in which clients sign in the system once and yet are able to access to services offered by different companies. The OASIS Security Assertion Markup Language (SAML) 2.0 Web Browser SSO Profile is the emerging standard in this context. In previous work a severe security flaw in the SAML-based SSO for Google Apps was discovered. By leveraging this experience, this chapter will show that model checking techniques for security protocols can support the development and analysis of SSO solutions helping the designer not only to detect serious security flaws early in the development life-cycle but also to provide assurance on the security of the solutions identified.

1 – INTRODUCTION

Single Sign-On (SSO) protocols are the cornerstones of Identity and Access Management systems as they enable companies to establish a federated environment in which users sign in once and yet are able to access to services offered by different organizations.

The *Security Assertion Markup Language* (SAML) 2.0 Web Browser SSO Profile (SAML SSO, for short) (OASIS, 2005a) is the emerging standard in this context: it defines an XML-based format for encoding security assertions as well as a number of protocols and bindings that prescribe how assertions should be exchanged in a wide variety of applications and/or deployment scenarios. This is done to the minimum extent necessary to guarantee the interoperability among different implementations. As a consequence, SAML SSO features many configuration options ranging from optional fields in messages, usage of SSL 3.0 or TLS 1.0 channels (SSL channels from here on) at the transport layer, application of encryption and/or digital signature on specific sensitive message elements which need to be instantiated according to the requirements posed by the application scenario and the available security mechanisms.

The security recommendations that are available throughout the bulky SAML specifications (OASIS, 2005a, 2005b) are useful in avoiding the most common security pitfalls but are of little help in ensuring their absence in specific instances of the protocol. Indeed the designer of a SAML-based SSO solution, while striving to meet the requirements posed by the application scenario, may overlook the security implications associated with the choice of some optional elements or may even decide to deviate from the SAML standard. Needless to say, this may have dramatic consequences on the security of the SSO solution.
The situation is exemplified by the SAML-based SSO for Google Apps. The protocol is inspired by the SAML standard but the version in operation until June 2008 deviated from it in a few, seemingly minor aspects (see Section 3.1 for the details). In May 2008 a severe security flaw was discovered and reported to Google and US-CERT (Armando, Carbone, Compagna, Cuéllar, & Tobarra, 2008, US-CERT, 2008). The vulnerability allowed a dishonest service provider to impersonate a user at another service provider. In reaction to the finding, Google immediately asked their customers to implement counter-measures to mitigate potential exploits and then to migrate to a new, patched solution of their SSO solution. Interestingly, the vulnerability was found by using a model checker for security protocols.
By leveraging this experience, this chapter will show that model checking techniques for security protocols can support the development and analysis of SSO solutions helping designers not only to detect serious security flaws early in the development life-cycle but also to provide assurance on the security of the solutions identified. By using SAML-based SSO protocols as running examples it will be illustrated how the design space can be iteratively explored with the aid of a model checker leading to the identification of vulnerabilities and hence to more secure solutions.
This work is part of a wider project aimed at the development of automated verification technologies for security protocols and, more generally, for security-sensitive, distributed applications (Armando et al., 2005, The AVANTSSAR Team).

Structure of the chapter.

In the next section the scene is set by briefly describing the state-of-the-art in model checking of security protocols. In Section 3.1 a brief introduction of the SAML SSO is given. In Section 3.2 the specification formalism is given in order to model security protocols, the properties of the transport protocols as well as those that the protocols are expected to meet. Section 3.3 presents the results of the analysis of SAML-based SSO protocols. In Section 4 the future research direction is discussed and Section 5 concludes the chapter with some final remarks.

2 - BACKGROUND

SSO protocols belong to the wider family of security protocols, i.e. communication protocols that, by means of cryptographic primitives, aim to provide security guarantees such as (mutual) authentication of the agents taking part in the protocol or secrecy of some information (e.g. a session key). In spite of their apparent simplicity, security protocols are notoriously error-prone. Many published protocols have been implemented and deployed in real applications only to be found flawed years later. For instance, the Needham-Schroeder authentication protocol (Needham & Schroeder, 1978) was found vulnerable to a serious attack 17 years after its publication (Lowe, 1996). Quite interestingly, many attacks can be carried out without breaking cryptography. These attacks exploit weaknesses in the protocol that are due to the complex and unexpected interleavings of different protocol sessions as well as to the possible interference of malicious agents. (The attack on the SAML-based SSO for Google Apps reported in (Armando et al., 2008) is of this kind.)
This is even more so for browser-based security protocols like the SAML SSO since, as pointed out in (Groß, Pfitzmann, & Sadeghi, 2005), browsers—unlike normal protocol principals—cannot be assumed to do nothing but execute the given security protocol. Moreover, browser-based security protocols normally assume for their proper functioning that communication between the browser and the server is carried over a unilateral SSL channel established through the exchange of a valid certificate. SSL channels provide a powerful mechanism for secure exchange of data in web-based applications, but the security guarantees they offer must not be overestimated. As it

will be shown in the sequel, serious vulnerabilities are possible even when communication takes place of SSL channels. Indeed the security of a SAML SSO solution critically depends on several assumptions (e.g. the trust relationships among the involved parties) and security mechanisms (e.g. the secure transport protocols used to exchange messages) as well as on the options and decisions taken during the design, development and deployment of such a solution.

For this reason, SAML-based SSO protocols and, more in general, security protocols are considered a challenging and promising application domain for formal methods and model checking techniques in particular. Unlike traditional verification techniques (e.g. testing), model checking allows for the exhaustive exploration of the behaviors of the protocol when communication between honest principals is controlled by a Dolev-Yao intruder (Dolev & Yao, 1983), a malicious agent capable to overhear, divert, and fake messages. Notice that the problem of determining whether a security protocol meets the expected security properties is undecidable in the general case (Durgin, Lincoln, Mitchell, & Scedrov, 1999). However, a number of decidable subclasses of this general problem have been identified (Durgin et al., 1999, Rusinowitch & Turuani, 2001, Chevalier, Küsters, Rusinowitch, & Turuani, 2003). Moreover, even semi-decision techniques have proved very helpful in detecting flaws in security protocols. Thus, during the last decade it has been witnessed to the development of a new generation of model checking techniques specifically tailored for security protocols (Meadows, 1996, Basin, 1999, Jacquemard, Rusinowitch, & Vigneron, 2000, Millen & Shmatikov, 2001). As a result of this endeavor, a wide range of industrial strength security protocols can now be analyzed automatically by state-of-the-art tools (Armando et al., 2005, The AVANTSSAR Team).

The experiments reported in this chapter (see Section 3.3) have been conducted with the AVANTSSAR Validation Platform for security protocols (hereafter in the sequel referred to as the AVANTSSAR Platform). An architectural view of the platform is depicted in Figure 1.

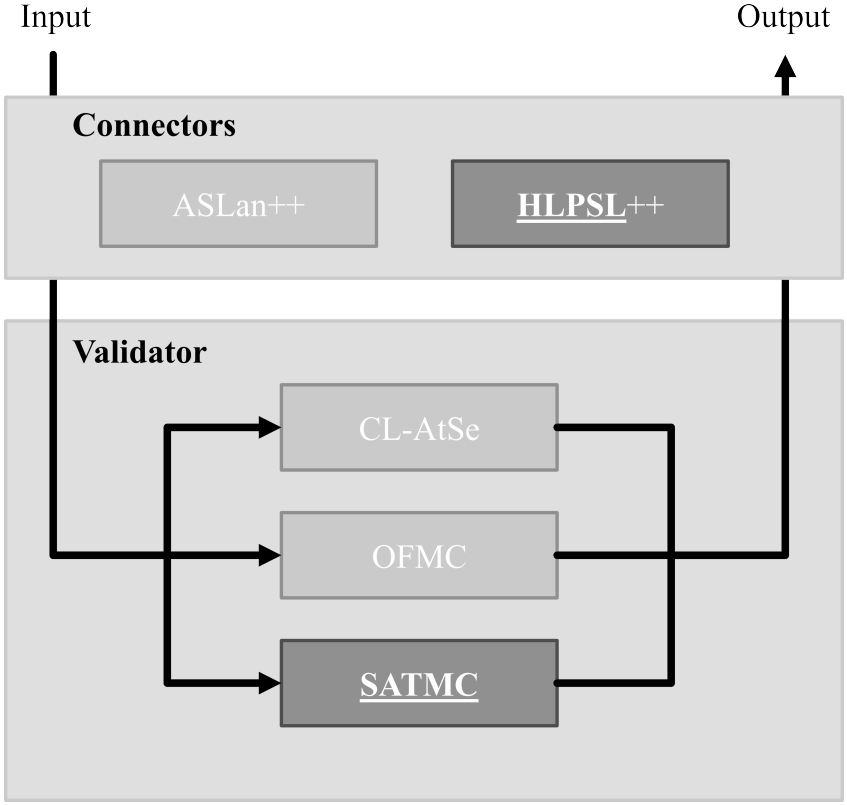


Figure 1: The AVANTSSAR Platform

The AVANTSSAR Platform takes as input a high-level specification of a security protocol, the expected security goals, as well as the scenario in which the protocol is employed and automatically evaluates its security. (A scenario is the parallel execution of a finite number of protocol sessions, where a protocol session is a run of the protocol played by a given set of agents.) A key feature of the AVANTSSAR Platform is the support of several types of communication channels. This is a fundamental feature in the case study addressed, as—as Section 1 has anticipated—SSO protocols heavily rely on the services offered by the transport protocols, namely SSL. The high-level specification is then translated by a connector into a security problem expressed in an intermediate specification language. The resulting specification, which is amenable to formal analysis, is fed to the validator which automatically checks—through the available back-ends—whether the protocol achieves its security goals. If this is not the case, then an attack trace is returned by the back-end and it is translated back into a user-friendly format. Currently the AVANTSSAR Platform supports three state-of-the-art model checker for security protocols as back-ends, namely CL-AtSe (Turuani, 2006), OFMC (Mödersheim & Viganò, 2009), and SATMC (Armando, Carbone, & Compagna, 2007). In the experiments reported in this chapter, the HLPSL++ connector and the SATMC back-end have been used.

The HLPSL++ connector takes as input a high-level specification written in HLPSL++ (see Section 3.2) and properly translates it so to instrument the formal analysis through the SATMC back-end. In addition, the HLPSL++ connector features a front-end that displays attacks (if any) as message sequence charts (MSC), a standardized notation commonly used in the protocol domain for the description of message exchanges between entities.

SATMC takes as input a formal specification of the protocol, a scenario to be considered for the analysis, a specification of the expected security property, and an integer *max* and tries to determine whether the protocol enjoys the expected security property in the scenario by considering up to *max* execution steps. At the core of SATMC lies a procedure that automatically generates a propositional formula whose satisfying assignments (if any) correspond to attack of length bounded by some integer $k \leq max$. Finding attacks (of length k) on the protocol therefore boils down to solving propositional satisfiability problems. SATMC relies on state-of-the-art SAT solvers for this task which can handle propositional satisfiability problems with hundreds of thousands variables and clauses and even more. SATMC can also be instructed to perform iterative deepening on k. By setting max to infinite ($max=-1$), SATMC is a semi-decision procedure that it is guarantee to terminate if there is an attack, but may not terminate if the protocol is secure. SATMC is a decision procedure for protocols without loops, i.e. it is guaranteed to terminate with a definitive sound answer. The security protocols considered in this chapter do not have loops and thus fall in this decidable class. When run against them, SATMC is thus guaranteed to either report an attack (if any) or to reach a termination condition that ensures that enough execution steps, say k_{safe}, have been explored proving the safety of the protocol (i.e., absence of attacks). More details on SATMC can be found in (Armando & Compagna, 2008, Armando et al., 2007).

3 - AUTOMATED SECURITY ANALYSIS OF SAML-BASED SSO PROTOCOLS

3.1 - THE SAML WEB BROWSER SSO PROFILE

The SAML specifications are based on the notions of assertions, protocols, bindings and profiles: A SAML *assertion* is an XML expression encoding a statement about a principal (also called subject). This chapter will consider a special type of assertions, called authentication assertions. Authentication assertion states, among other things, that a given subject was authenticated by a particular means at a particular time for the purpose to get access to the service provider. *Protocols* prescribe how assertions should be exchanged and *bindings* detail how assertions can be mapped into transport protocols, e.g. SOAP or HTTP. Finally, *profiles* define the use of SAML assertions, protocols and bindings so to meet some specific use case requirements (e.g. SSO).

The purpose of the Web Browser SAML 2.0 SSO profile is to enable a client to obtain authenticated access to a restricted resource on a service provider. Three roles take part in the protocol: a client (C), an identity provider (IdP) and a service provider (SP). C, typically a web browser guided by a user, aims at getting access to a service or a resource provided by SP. IdP authenticates C and issues corresponding authentication assertions. Finally, SP uses the assertions generated by IdP to give C access to the requested service. The standard allows the protocol to be initiated

either by the SP or by the IdP. In the *SP-initiated SSO*, any non-authenticated client trying to access a resource on SP is automatically redirected to the IdP for authentication. In the IdP-initiated SSO, C starts the protocol by directly contacting to and authenticating at the IdP. This chapter focuses on the SP-initiated SSO using a Redirect Binding for the SP-to-IdP `<AuthnRequest>` message and a POST Binding for the IdP-to-SP `<Response>`. The protocol is depicted in Figure 2.

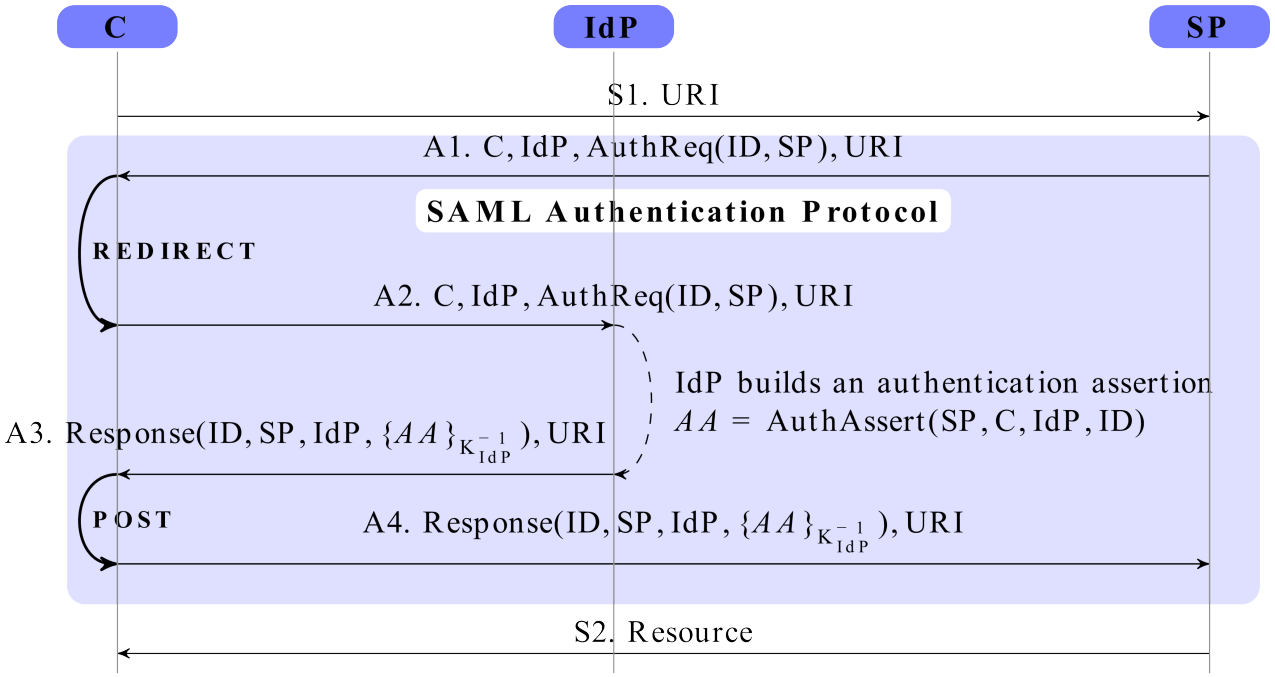


Figure 2: SP-Initiated SSO with Redirect/POST Bindings

In step S1, C asks SP to provide the resource located at the address URI. SP then initiates the *SAML Authentication Protocol* by sending C a redirect response directed to IdP containing an authentication request of the form AuthReq(ID,SP)—where ID is a string uniquely identifying the request—and the address URI of the resource. IdP then challenges C to provide valid credentials and—if the authentication succeeds—IdP builds an authentication assertion *AA*=AuthAssert(SP,C,IdP,ID) and places it into a response message *Resp*=Response(ID,SP,IdP,$\{AA\}_{K^{-1},IdP}$), where $\{AA\}_{K^{-1},IdP}$ is the assertion digitally signed with $K^{-1,IdP}$, the private key of IdP. (As unnecessary for the analysis, it will not be modeled how the authentication between C and IdP is performed, but it is assumed it is successful.) IdP then places *Resp* into an HTML form as a hidden form control and sends it back to C. For ease of use purposes, the HTML form is typically accompanied by script code that automatically posts the form to SP. This completes the protocol and SP can deliver the requested resource to C.

The security of the SAML SSO protocol relies on a number of assumptions about the trustworthiness of the principals involved as well as the security of the transport protocols employed. Below they are summarized.

Trust Assumptions. The protocol assumes that IdP is trustworthy for both C and SP, but SP is not assumed to be trustworthy. Indeed service providers do not necessarily trust each other e.g., a SP could have been compromised or simply act maliciously driven by own interests. In accordance with this the analysis will consider protocol sessions in which C and SP might be played by the intruder, while it is assumed that IdP is played by an honest agent. It is worth noticing that in the analysis of the SAML SSO v1.0 reported in (Hansen, Skriver, & Nielson, 2005) it is assumed that

the SP is trustworthy. This assumption is not realistic as service providers do not necessarily trust each other and it would prevent the detection of the attack on the SAML-based SSO for Google Apps reported in (Armando et al., 2008).

Assumptions on the communication channels. Communications between the parties are subject to the following assumptions:

(A1) the communication between C and SP is carried over a unilateral SSL channel, established through the exchange of a valid certificate (from SP to C); and

(A2) the communication between C and IdP is carried over a unilateral SSL channel that becomes bilateral once C authenticates itself on IdP. This is established through the exchange of a valid certificate (from IdP to C) and of valid credentials (from C to IdP).

Under the above assumptions the protocol is expected to meet the following security properties:

(P1) SP authenticates C, i.e., at the end of its protocol run SP believes it has been talking with C; and

(P2) Resource must be kept secret between C and SP.

The SAML-based Single Sign-On for Google Apps used by Google until June 2008 shared the above assumptions and expected properties, but deviated from the SAML SSO protocol for a few, seemingly minor simplifications in the messages exchanged:

(G1) ID and SP are not included in the authentication assertion, i.e. *AA* is AuthAssert(C,IdP) instead of AuthAssert(SP,C,IdP,ID); and

(G2) ID, SP and IdP are not included in the response, i.e. *Resp* is Response($\{AA\}_{K^{-1},IdP}$) instead of *Resp*=Response(ID,SP,IdP,$\{AA\}_{K^{-1},IdP}$).

3.2 - FORMAL MODELING OF THE SAML WEB BROWSER SSO PROFILE

To formally specify SAML-based SSO protocols it has been used HLPSL++, a role-based specification language for security protocols developed in the context of the AVANTSSAR Project (The AVANTSSAR Team). Unlike its predecessor HLPSL (High-Level Protocol Specification Language) (Chevalier et al., 2004), HLPSL++ supports the specification of communication channels beyond the Dolev-Yao model. This includes confidential and authentic channels that are building blocks for capturing real world transport-level communications such as SSL. In HLPSL++, actions carried out by a protocol participant are grouped and specified in a *basic role*. Basic roles are then instantiated and glued together into *composed roles*. This section provides a brief introduction to the language through its application to the SAML SSO.

```
1   role client(
2     C, IdP, SP : agent, KIdP : public_key,
3     C2SP, SP2C, C2IdP, IdP2C : channel,
4     URI : text
5   ) played_by C def=
6   local
7     State : nat,
8     ID, Resource : text
9   init
10    State:=0
11  transition
12  %% C asks for a resource to SP
13  S1.    State = 0 =|>
14         State':=2 /\ snd(C2SP, SP, URI) /\ witness(C, SP, sp_c_uri,URI)
15  %% C receives an AuthnReq(ID, SP) to be forwarded to IdP
16  A1_A2. State = 2 /\ rcv(SP2C, SP, C.IdP.(ID'.SP).URI) =|>
```

```
17        State’:=4 /\ snd(C2IdP, IdP, C.IdP.(ID’.SP).URI)
18  %% C receives a Response(SP, C, IdP, ID) for SP
19  A3_A4. State = 4 /\ rcv(IdP2C, IdP, SP.{SP.C.IdP.ID’}_inv(KIdP).URI) =|>
20        State’:=6 /\ snd(C2SP, SP, SP.{SP.C.IdP.ID’}_inv(KIdP).URI)
21  %% C receives a resource from SP
22  A4_S2. State = 6 /\ rcv(SP2C, SP, Resource’) =|>
23        State’:=8
24  end role
```

Figure 3: HLPSL++ specification of the SAML SSO protocol: role C

```
1   role serviceProvider (
2     C, IdP, SP : agent, KIdP : public_key,
3     SP2C, C2SP : channel, URI : text
4   ) played_by SP def=
5   local
6     State : nat,
7     ID, Resource : text
8   init
9     State:=1
10  transition
11  %% SP receives a request for a resource and issues an
12  %% AuthReq(ID, SP)
13  S1_A1. State = 1 /\ rcv(C2SP, C, URI) =|>
14         State’:=3 /\ ID’ := new()
15                   /\ snd(SP2C, C, C.IdP.(ID’.SP).URI)
16  %% SP receives a Response(SP, C, IdP, ID) and serves the
17  %% resource to C
18  A4_S2. State = 3 /\ rcv(C2SP, C, SP.{SP.C.IdP.ID}_inv(KIdP).URI)
    =|>
19         State’:=5 /\ Resource’ := new()
20                   /\ snd(SP2C, C, Resource’)
21                   /\ request(SP, C, sp_c_uri, URI)
```

Figure 4: HLPSL++ specification of the SAML SSO protocol: role SP

```
1   role identityProvider (
2     C, IdP, SP : agent, KIdP : public_key,
3     IdP2C, C2IdP : channel
4     ) played_by IdP def=
5   Local
6     ID, URI : text,
7     State : nat
8   init
9     State:=7
10  transition
11  %% IdP receives an AuthReq(ID, SP) from C and issues a
12  %% Response(SP, C, IdP, ID)
13  A2_A3. State = 7 /\ rcv(C2IdP, C, C.IdP.(ID’.SP).URI’) =|>
14         State’:=9          /\          snd(IdP2C,          C,
    SP.{SP.C.IdP.ID’}_inv(KIdP).URI’)
15  end role
```

Figure 5: HLPSL++ specification of the SAML SSO protocol: role IdP

The definitions of the three basic roles participating in the SAML SSO, namely `client`, `serviceProvider`, and `identityProvider`, are given in the figures 3, 4, and 5 respectively. The specification of a role includes a list of parameters and the agent playing that role. A basic role with parameters therefore corresponds to many possible instances of same role obtained by instantiating the parameters with terms of the appropriate type. For instance , the role `serviceProvider`—cf. lines 2-3 of Figure 4—has the following parameters: `C`, `SP`, and `IdP` of type `agent`, `KIdP` of type `public_key`, `SP2C` and `C2SP` of type `channel` (used by SP to communicate with the client), and `URI` of type `text`. In addition, this role will be played by agent `SP` (cf. line 4). Roles (both basic and composed) can also contain the definition of local variables and constants within the section `local`. For instance, `serviceProvider` uses the local variable `State` of type `nat` and the local variables `ID` and `Resource` of type `text` (cf. lines 5-7). Local variables can be assigned to initial values in the section `init`. For instance, `State` is initially assigned to the value `1` in the `serviceProvider` role (cf. line 9).

The behavior of roles is specified in the `transition` section. This comprises a collection of transitions of the form *Pre*`=|>`*Post*, where *Pre* is the conjunction (denoted by the `/\` symbol) of preconditions for the applicability of the transition and *Post* is the conjunction of effects that result by the execution of the transition. For example, the first transition in Figure 4 (cf. lines 13-15) occurs only if `State=1` and a message `URI` is received on channel `C2SP` from agent `C`—this is represented by `rcv(C2SP, C, URI)`—and its execution sets the value of `State` to `3`, generates a fresh value for `ID` (cf. second conjunct in line 14), and makes `SP` (the player of the role) sending the message `C.IdP.(ID'.SP).URI` to `C` over channel `SP2C`. The primed variable `ID'` denotes the new value of `ID`. In general when a primed variable occurs in a received message, it means that the variable will be assigned to a new value, the one specified in the corresponding part of the message received. If the primed variable occurs in an outgoing message, then the value just assigned to that variable will be included in the message. In summary, unprimed variables denote message elements that the role is checking while receiving that message (as they are already stored somewhere in its state), while primed variables model those elements that are unknown to the receiver. For instance, the transition of `identityProvider` (see Figure 5, lines 13-14) states that `IdP` checks whether the first two and

```
1  role session (
2    C, IdP, SP : agent, KIdP : public_key,
3    C2SP, SP2C, C2IdP, IdP2C : channel,
4    URI : text
5  ) def=
6  init
7  %% Assumption (A1): C <-> SP
8       confidential(SP, C2SP)
9    /\ weakly_authentic(C2SP)
10   /\ weakly_confidential(SP2C)
11   /\ authentic(SP, SP2C)
12   /\ link(C2SP, SP2C)
13 %% Assumption (A2): C <-> IdP
14   /\ confidential(IdP, C2IdP)
15   /\ weakly_authentic(C2IdP)
16   /\ confidential(C, IdP2C)
17   /\ authentic(IdP, IdP2C)
18   /\ link(C2IdP, IdP2C)
19 composition
20      client(C, IdP, SP, KIdP, C2SP, SP2C, C2IdP, IdP2C, URI)
21   /\ serviceProvider(C, IdP, SP, KIdP, SP2C, C2SP, URI)
22   /\ identityProvider(C, IdP, SP, KIdP, IdP2C, C2IdP)
23 end role
```

Figure 6: HLPSL++ specification of the SAML SSO protocol: session

the fourth element of the message received correspond to the (known) values of C, IdP and SP respectively, while it will not check the last and the third elements, namely ID' and URI'.

Once the basic roles are defined, their parallel composition is defined by means of the composed role session as shown in Figure 6. The parameters of the composed role can be related to those of the component roles so that when the composed role is instantiated its component roles are properly instantiated too. Parameters include the channels used by component basic roles to exchange messages. Each SSL channel (cf. assumptions (A1) and (A2)) is modeled by two unidirectional channels (see part (a) of Figure 7). C2SP and SP2C model the SSL channel between C and SP: the former carries messages from C to SP, the latter carries messages flowing in the opposite direction. They are assumed to enjoy the following properties (cf. lines 8-12):

- confidential(SP, C2SP), i.e. the output of C2SP is accessible to SP only, and weakly_authentic(C2SP), i.e. the input of C2SP is exclusively accessible to a single, yet unknown, sender;
- weakly_confidential(SP2C), i.e. the output of SP2C is exclusively accessible to a single, yet unknown, receiver and authentic(SP, SP2C), i.e. the input of SP2C is accessible to SP only; and
- link(C2SP, SP2C), i.e. the principal sending messages on C2SP is the same principal that receives messages from SP2C.

Channel C2IdP and IdP2C model the SSL channel between C and IdP. The former carries messages from C to IdP, the latter carries the messages flowing in the opposite direction. The properties enjoyed by C2IdP (IdP2C) are similar to those of C2SP (resp. SP2C), the only difference being that IdP2C is confidential to C and not simply weakly confidential thanks to the authentication of C on IdP (cf. lines 14-18). A precise definition of the properties of channels supported by HLPSL++ can be found in (Armando et al., 2008).

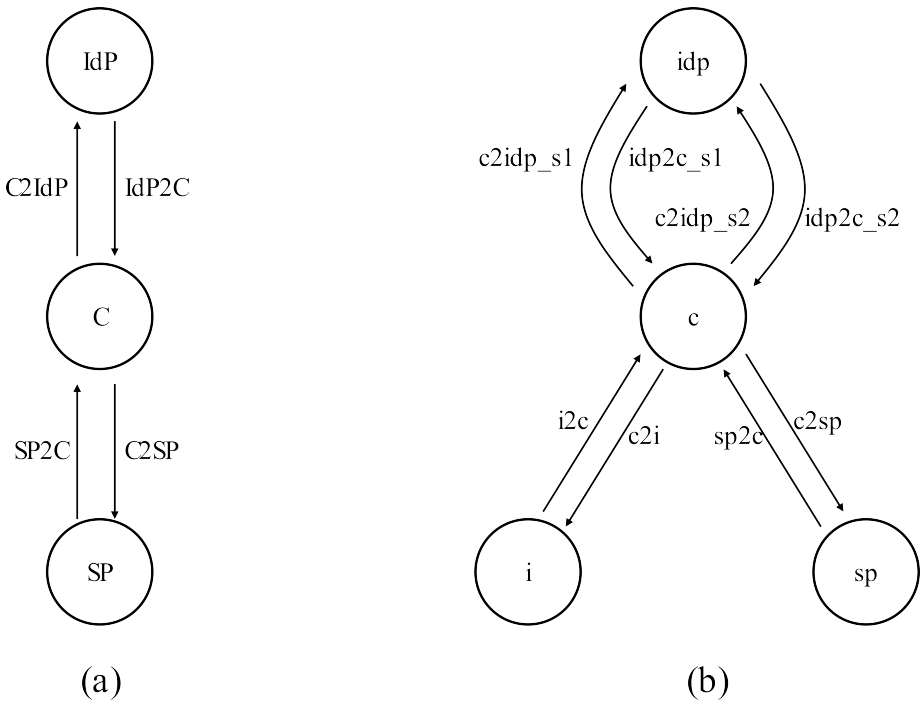


Figure 7: (a) Communication channels among basic roles; (b) Communication channels instantiated within the environment role

The HLPSL++ specification is concluded by the top-level role `environment` (see Figure 8). This role includes a `const` section (cf. lines 4-8) containing the declaration of global constants, the definition of the knowledge initially possessed by the intruder (cf. lines 9-11), and a `composition` section (cf. lines 13-14) that defines the parallel composition of the sessions. This specification contains a session in which `c` and `idp` play the protocol with `i` (playing the role of SP) and a session in which `c` and `idp` play the protocol with `sp` by using proper communication channels to communicate (see part (b) of Figure 7). Notice that a constant `i` of type `agent` is implicitly declared and associated with the intruder.

```
1  role environment()
2  def=
3  Const
4    sp_c_uri, c_sp_resource, uri_i, uri_sp : protocol_id,
5    c, idp, sp : agent,
6    c2idp_s1,idp2c_s1,c2idp_s2,idp2c_s2,c2i,i2c,c2sp,sp2c: channel,
7    n : text,
8    kidp, ki : public_key,
9  intruder_knowledge={c, sp, idp, kidp, ki, inv(ki), uri_i, uri_sp, n,
10                     c2i, i2c, c2sp, sp2c, c2idp_s1, idp2c_s1,
11                     c2idp_s2, idp2c_s2}
12 Composition
13       session(c, idp, i, kidp, c2i, i2c, c2idp_s1, idp2c_s1, uri_i)
14   /\ session(c, idp, sp, kidp, c2sp, sp2c, c2idp_s2, idp2c_s2,uri_sp)
15 end role
16
17 Goal
18 %% property (P1)
19   authentication_on sp_c_uri
20 %% property (P2)
21   secrecy_of c_sp_resource
22 end goal
23
24 environment()
```

Figure 8: HLPSL++ specification of the SAML SSO protocol: environment and goals

The security properties that the protocol is expected to meet are stated in the `goal` section (cf. lines 17-22). These are the authentication property (P1) and the secrecy property (P2) described in Section 3.1. Hereafter more details will be provided about how these properties are modeled.

Authentication.

The authentication property supported by the HLPSL++ language corresponds to the notion of agreement as defined in (Lowe, 1997). Thus, *`SP` authenticates `C` on `URI`* amounts to saying that whenever `SP` completes a run of the protocol apparently with `C`, then *(i)* `C` has previously been running the protocol apparently with `SP`, and *(ii)* the two agents agree on the value of `URI` (Lowe, 1997). To specify this property in HLPSL++ it suffices for `SP` to assert the fact `request(SP, C, sp_c_uri, URI)` in its last transition (cf. line 21 of Figure 4), and for `C` to assert the fact `witness(C, SP, sp_c_uri, URI)` as soon as it sends `URI` (cf. line 14 of Figure 3). The statement `authentication_on sp_c_uri` in the `goal` section (cf. line 19) states that if `request(SP, C, sp_c_uri, URI)` is asserted, then sometime in the past a corresponding `witness(C, SP, sp_c_uri,`

URI) must have been asserted for the authentication property to hold. In fact, if a request is not matched by a corresponding witness, then the intruder must have been playing the protocol run pretending to be the one that otherwise would have asserted the witness expression.

Secrecy.

The secrecy property (P2) holds if and only if the resource at URI provided by SP to C is not learned by the intruder when the intruder is neither playing SP nor C in that protocol session. To specify this property in HLPSL++ it suffices for SP to assert the fact secret(Resource',c_sp_resource,{C,SP}) when sending the last message (cf. line 22 of Figure 4). (The identifier c_sp_resource serves as a label for this fact.) The statement secrecy_of c_sp_resource in the goal section (cf. line 21 in Figure 8) states that if secret(Resource',c_sp_resource,{C,SP}) is asserted, then i should not learn the secret value stored in Resource' unless i is the value of either C or SP.

3.3 - SECURITY ANALYSIS OF SAML-BASED SSO PROTOCOLS

This section illustrates how the AVANTSSAR Platform can be used by software vendors and adopters of SAML SSO to evaluate the impact of their decisions (e.g., selected SAML SSO options and/or deviations from the SAML SSO standard) on the security of their SSO solutions. In doing so it focuses on a set of SAML-based SSO protocols ranging from that proposed by the OASIS standardization body (hereafter referred to as SAMLSSO) to the one currently in use by Google (GoogleSSO-Post) and considering in-between variants of the two, including the flawed version used by Google until June 2008 (GoogleSSO-Pre).
Table 1 presents an excerpt of the results obtained by running the AVANTSSAR Platform against all these SAML-based SSO variants that differ from each other for the decisions taken in their design, development, and/or deployment:

- Sign(AuthReq): it indicates whether the authentication request AuthReq is signed or not by the SP. (The SAML standard leaves this as optional.)
- SP∈*AA*: it indicates that the SP field is within the authentication assertion *AA* as required by the SAML standard.
- ID∈*AA*: it indicates that the ID field is within the authentication assertion *AA* as required by the SAML standard.
- ID checked: it indicates that SP checks that the ID value received in the authentication assertion is identical to the one it generated for the authentication request. (Stateless SPs may not be able to perform this check despite this is required by the SAML standard.)

For each variant, the table indicates the results of the analysis for the authentication property (P1): if there is an attack (Attack column), the time spent by the tool (Time column), and the number of steps required by SATMC to perform the analysis i.e., the number of steps necessary to either find the attack or prove the safety of the protocol (Steps column). (Experiments have been carried out on a laptop PC with an Intel Core 2 Duo 2.53 GHz CPU and 4 GB of RAM.) The results for the secrecy property (P2) are similar: when no attack is found on (P1), no attack is discovered on (P2) and whenever an attack is found on (P1), an attack on (P2) is found which contains an additional step in which the intruder receives the resource.

	Decisions				Property (P1)		
Variant	**Sign(AuthReq)**	**SP□*AA***	**ID□*AA***	**ID checked**	**Attack**	**Time (sec)**	**Steps**
SAMLSSO	n	y	Y	y	n	15,2	21
GoogleSSO-Pre	n	n	N	-	y	3,7	13
SSOVariant1	y	n	N	-	y	3,3	13
SSOVariant2	n	n	Y	n	y	4,9	13
SSOVariant3	n	n	Y	y	y	11,3	17
SSOVariant4	n	y	N	-	n	3,7	15
GoogleSSO-Post	n	y	Y	n	n	8,4	18

Table 1: Results of the Experiments

As shown in Table 1 the AVANTSSAR Platform does not find any attack on `SAMLSSO`. This means that the protocol is safe in the scenario considered.

`GoogleSSO-Pre` differs from `SAMLSSO` for the missing ID and SP fields in the authentication assertion. The reasons for not having used those fields are unknown. Maybe they were the consequence of internal requirements. For instance, a vendor may decide to not use ID as the advantages of handling it—from its generation in the authentication request to its check in the authentication response—are not evident when compared to potential drawbacks (e.g., additional communication between the server at the SP that generates the authentication request and the server at the SP that provides the final resources). To assess the implications of these decisions on the security of the protocol, the HLPSL++ specification for `SAMLSSO` has been modified by removing ID and SP from the authentication assertion. This amounts to carrying out the following changes in specification of the basic roles:

- change for IdP: Figure 5, line 14 becomes:

```
14        State':=9 /\ snd(IdP2C, C, SP.{C.IdP}_inv(KIdP).URI')
```

- change for C: Figure 3, lines 19-20 become:

```
19  A3_A4. State = 4 /\ rcv(IdP2C, IdP, SP.{C.IdP}_inv(KIdP).URI) =|>
20        State':=6 /\ snd(C2SP, SP, SP.{C.IdP}_inv(KIdP).URI)
```

- change for SP: Figure 4, lines 18 becomes:

```
18  A4_S2. State = 3 /\ rcv(C2SP, C, SP.{C.IdP}_inv(KIdP).URI) =|>
```

By analyzing the resulting specification, the AVANTSSAR Platform reports violations on both authentication and secrecy in accordance with the discoveries outlined in (Armando et al., 2008). The attack shows that a compromised SP is able to impersonate a client at another SP. Figure 9 depicts the MSC returned by the AVANTSSAR Platform for the attack on the authentication property. The attack relies on the execution of two parallel protocol sessions where the intruder plays the man-in-the-middle by exploiting a weakness in the authentication assertion. In fact the authentication assertion no longer states that *C is authenticated at IdP for the request ID made at SP* as prescribed by the standard, but it simply states that *C is authenticated to IdP*. This allows the intruder to misuse the assertion to get access to a different SP. More in detail:

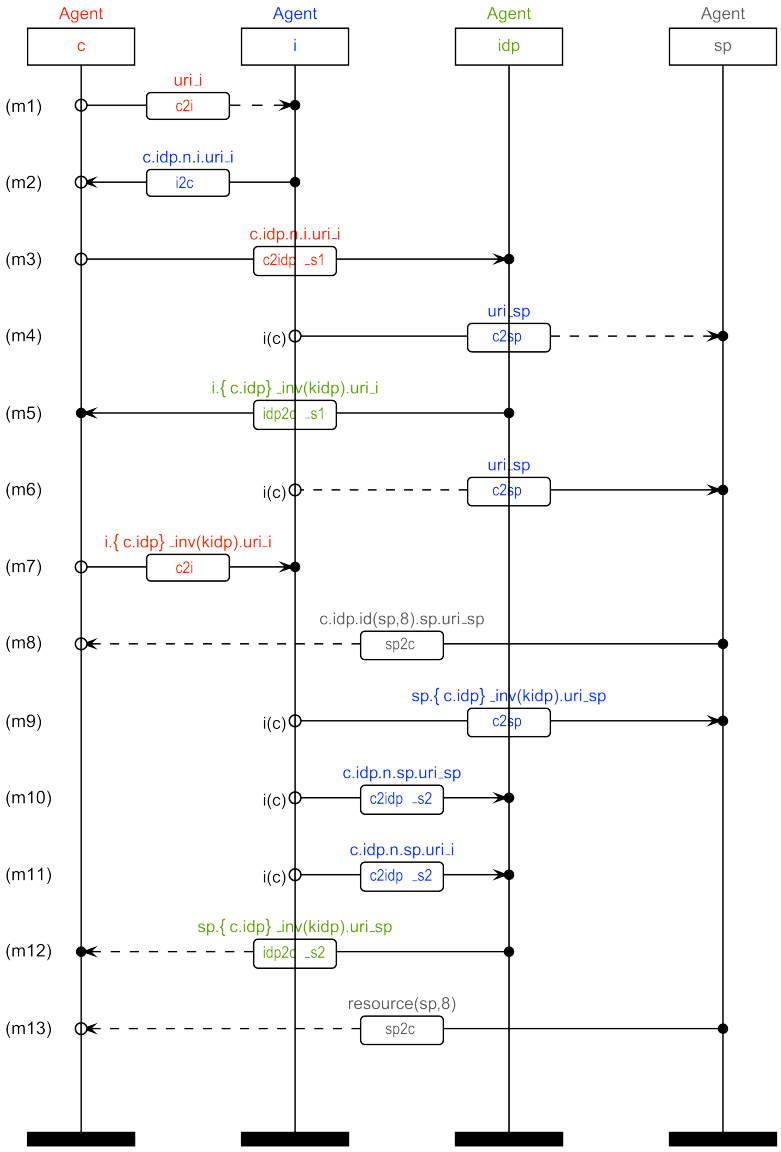
- in message (`m1`) `c` initiates a session of the protocol to access a resource provided by the (malicious or compromised) service provider `i`. Notice that it is not necessary that the message is actually received by `i` because `i` already knows the content of the first message (`uri_i`).
- `i` sends to `c` a redirect response directed to `idp` containing an authentication request and the address `uri_i` of the resource (see messages (`m2`) and (`m3`)).
- The `idp` generates the authentication assertion which is posted by `c` to `i` (see messages (`m5`) and (`m7`)).

- In parallel `i` starts a new session of the protocol with `sp` pretending to be `c` (thus indicated as `i(c)`). This session is interleaved with the previous one. In message (`m4`) `i` sends the first message to `sp`, and in message (`m6`) this message is received by `sp`.
- The authentication request sent by `sp` to the official recipient `c` (see message (`m8`)) is neglected by `i`.
- The intruder mischievously reuses the assertion received by `c` (see message (`m7`)) to trick `sp` into believing he is `c` (see message (`m9`)).

In the MSC, different arrows are used to indicate if a particular message has been sent or received. The notation (*i*) $A \xrightarrow{M}_{ch} B$ indicates that the message *M* has been sent by *A* over the channel *ch* to the agent *B* that has not received yet (pointed by the dotted arrow). With reference to the specification in Section 3.2, this corresponds to the execution of a transition of the basic role *A*, having as effect `snd(`*ch*`,`*B*`,`*M*`)`. Notice that *M* will not be necessarily received by *B*. If this is the case, the notation (*ii*) $A \xrightarrow{M}_{ch} B$ indicates that the message *M* sent by the sender *A* has been received by *B* (pointed by the solid arrow). With reference to the specification in Section 3.2, this corresponds to the execution of a transition of the basic role *B*, having as precondition `rcv(`*ch*`,`*B*`,`*M*`)`. In order to increase the readability of the MSC, when the same message is sent and received by the same couple of agents in two contiguous steps, then (*i*) and (*ii*) are replaced by a single solid arrow, thus the resulting notation is as follows: $A \xrightarrow{M}_{ch} B$.
In the MSC, a bullet notation is used to display the properties of the channels defined in the HLPSL++ specification. For instance, the notation $A \circ\!\!\xrightarrow{M}_{ch}\!\!\bullet\ B$ indicates that the channel *ch* is weakly authentic and confidential to *B*, while $A \bullet\!\!\xrightarrow{M}_{ch}\!\!\bullet\ B$ indicates that *ch* is authentic for *A* and confidential to *B*. A complete definition of these symbols is detailed in the legend of Figure 9.
Note that though SATMC is able to discover the shortest attack (if any), still this attack can comprise some spurious messages. This is the case for messages (`m10`), (`m11`), and (`m12`) that are not relevant for the attack.

As a possible countermeasure against this man-in-the-middle attack one may think to ask the SP to sign the authentication request. The corresponding HLPSL++ specification (`SSOVariant1`) is obtained by carrying out a few minor changes similarly to what it has been done above. By running the AVANTSSAR Platform against `SSOVariant1`, it has been discovered that the signature on the authentication assertion does not prevent the man-in-the-middle attack. The MSC of Figure 10 depicts the first four messages of this attack as the rest of the attack proceeds as in Figure 9, the only difference being the presence of the digitally signed authentication assertion in (`m3`) and (`m4`).

As shown above the lack of the SP field in the authentication assertion seems an insurmountable cause of the attack, but one may wonder whether the presence of the ID field in the authentication assertion could make the protocol secure. Two variants of the `SAMLSSO` protocol have been taken into account. Both of them do not contain the SP field in the authentication assertion but contain the ID field: `SSOVariant2` and `SSOVariant3`. In the latter SP checks that the ID in the authentication assertion is identical to the one included in the authentication request, while in the former this check is omitted. Both variants are still vulnerable to the attack. Notice that the attack on `SSOVariant3` requires a few more steps (cf. Table 1). In fact, differently from the previous attacks, the intruder cannot start the two sessions in parallel, because he needs to know the ID generated by `sp` before starting the session with `c`. The MSC of Figure 11 depicts the first messages of the attack, in which `i` starts a session of the protocol with `sp` in order to obtain the ID (cf. messages (`m1`) and (`m2`)). When `i` receives the request by `c` (cf. message (`m3`)), she sends to `c` message



Legend:

$A \ast\!\!-\!\!\overset{M}{\boxed{ch}}\!\!\rightarrow\!\star\ B$: A sends M to B over the channel ch

$A \ast\!\!-\!\!-\!\!\overset{M}{\boxed{ch}}\!\!\rightarrow\!\star\ B$: B receives M from the real sender A over the channel ch

$A \ast\!\!-\!\!\overset{M}{\boxed{ch}}\!\!\rightarrow\!\star\ B$: A sends M to B over the channel ch, and B receives it

Channel properties:
if $\ast$ = ○ then ch is weakly authentic
if $\ast$ = ● then ch is authentic for A
if $\star$ = ○ then ch is weakly confidential
if $\star$ = ● then ch is confidential to B

Figure 9: Attack on the SAML-based SSO for Google Apps

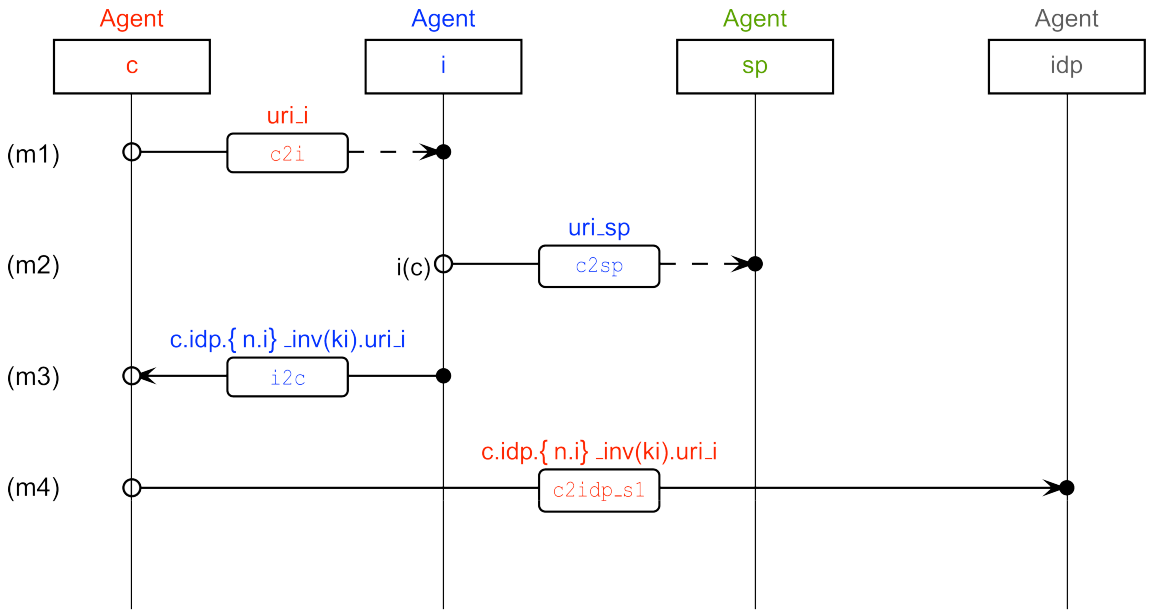


Figure 10: Attack on the SAML-based SSO for Google Apps despite signature on the authentication request

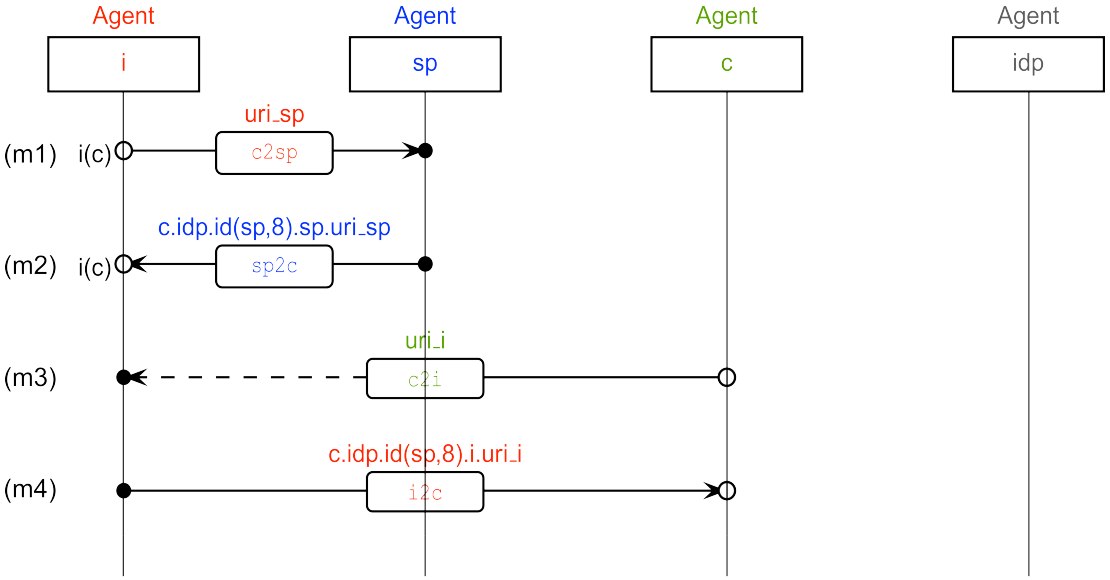


Figure 11: Excerpt of the attack on `SSOVariant3`

(m4) which contains the ID obtained in (m2). Thus, the authentication assertion produced by idp for i contains the ID expected by sp. The analysis of SSOVariant2 and SSOVariant3 thus confirms that the SP field is indeed necessary to prevent the man-in-the-middle attack of Figure 9.

One may now wonder whether the ID field, which is mandated by the standard, is necessary to achieve the expected security property. The specification SSOVariant4 models the variant of the protocol where the ID field is dropped from both the authentication request and the authentication assertion. No attack is found by the AVANTSSAR Platform in this case. This result questions the importance and need of the ID field: though this field is useful to link the authentication response with the corresponding authentication request (which seems a good practice to proceed with), it seems irrelevant for the authentication and secrecy properties that have been considered.
Finally it has been built a HLPSL++ specification of the current version of the SAML-based SSO for Google Apps which is in operation since August 2008. This specification (GoogleSSO-Post) comprises the ID field in both the authentication request and the authentication assertion but the correspondence of these values is not checked by the Google SP. As expected, by running the AVANTSSAR Platform on GoogleSSO-Post no attack is found.

4 - FUTURE RESEARCH DIRECTIONS

As shown in the previous section, model checking is effective to automatically detect subtle flaws in the logic of distributed applications. Yet, it must be noted that the analysis is carried out on a formal model of the system (as opposed to the actual system) and therefore the applicability of model checking techniques is usually confined to the design phase.
Security testing, unlike model checking, can be used to check the behavior of the actual system. It has been successfully applied to authorization and similar application-level policies. A special form of security testing, namely penetration testing, is effective in finding low-level vulnerabilities in on-line applications (e.g., cross-site scripting), but heavily relies on the guidance and expertise of the user. Security testing is normally applied in later stages of the service life-cycle, i.e., during the deployment or even the consumption phase.
Both model checking and security testing are already routinely used to unveil serious vulnerabilities and are therefore going to play a central role in improving the security of SSO solutions, and more generally of web-based applications. However, there is an enormous potential in using these technologies in combination rather than in isolation. In fact, state-of-the-art security verification technologies, if used in isolation, do not provide automated support to the discovery of important vulnerabilities and of the associated exploits that are already plaguing complex, web-based, security-sensitive applications. On the one hand, while model checking is key to the discovery of subtle vulnerabilities due to unexpected interleavings of service executions, it provides no support to testing the actual services. On the other hand, penetration testing tools—by supporting the analysis of a single service at a time—lack the global view and the automated reasoning capabilities necessary to discover the kind of vulnerabilities found by model checkers, but provide both infrastructure and repertoires of testing techniques that are very useful to find exploits related to the high-level vulnerabilities found by model checkers.
The integration of model checking for security protocols and security testing can be achieved by automatically deriving test cases from counter-examples found through model checking. Preliminary work in this direction is proposed in (Armando, Carbone, Compagna, Li, & Pellegrino, 2010), where the feasibility of the approach is illustrated via its application to the SAML SSO.

5 - CONCLUSION

Model checking is a verification technique that helps finding flaws in security-sensitive, distributed applications at the different phases of the service life-cycle. This chapter shows that model checking can be used by software vendors and adopters of SAML SSO to automatically assess the impact of their decisions (e.g., choice of options and/or deviations from the standard) on the security of their SSO solutions. The practical viability of the approach has been demonstrated through the application of a state-of-the-art model checker to support the security analysis of several

variants of the SAML SSO. The analysis confirms the attack on the SAML-based SSO for Google Apps in operation until June 2008 as well as on variants thereof. The analysis also confirms the security of the SAML SSO 2.0 as well as of the SAML-based SSO for Google Apps in operation since August 2008, since no attack has been detected despite the several protocol scenarios considered. The analysis seems also to indicate that the presence of the ID field in authentication assertion—which is mandated by the standard—does not affect the security of the protocol.

REFERENCES

Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., et al. (2005). The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*. Springer-Verlag. (Available at `www.avispa-project.org`)

Armando, A., Carbone, R., & Compagna, L. (2007, July). LTL model checking for security protocols. In *20th IEEE Computer Security Foundations Symposium (CSF20)*. Venice (Italy).

Armando, A., Carbone, R., Compagna, L., Cuéllar, J., & Tobarra, M. L. (2008, October). Formal analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In V. Shmatikov (Ed.), *Proceedings of the 6th ACM workshop on formal methods in security engineering, FMSE 2008* (pp. 1–10). ACM.

Armando, A., Carbone, R., Compagna, L., Li, K., & Pellegrino, G. (2010, 6-10). Model-checking driven security testing of web-based applications. In *2010 third international conference on Software testing, verification, and validation workshops (ICSTW)*. (p. 361 -370).

Armando, A., & Compagna, L. (2008, January). SAT-based Model-Checking for Security Protocols Analysis. *International Journal of Information Security*, *7*(1), 3-32.

Basin, D. (1999). Lazy infinite-state analysis of security protocols. In R. Baumgart (Ed.), *Secure networking — CQRE (secure)'99* (pp. 30–42). Springer-Verlag.

Chevalier, Y., Compagna, L., Cuellar, J., Hankes Drielsma, P., Mantovani, J., Mödersheim, S., et al. (2004). A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In *Proc. SAPS'04*. Austrian Computer Society.

Chevalier, Y., Küsters, R., Rusinowitch, M., & Turuani, M. (2003). An NP Decision Procedure for Protocol Insecurity with XOR. In P. Kolaitis (Ed.), *Proceedings of LICS'2003*. IEEE.

Dolev, D., & Yao, A. (1983). On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, *2* (29).

Durgin, N., Lincoln, P. D., Mitchell, J. C., & Scedrov, A. (1999). Undecidability of Bounded Security Protocols. In *Proceedings of the FLOC'99 workshop on formal methods and security protocols (FMSP'99)*.

Groß, T., Pfitzmann, B., & Sadeghi, A.-R. (2005). Browser model for security analysis of browser-based protocols. In S. D. C. di Vimercati, P. F. Syverson, & D. Gollmann (Eds.), *ESORICS* (Vol. 3679, p. 489-508). Springer.

Hansen, S. M., Skriver, J., & Nielson, H. R. (2005). Using static analysis to validate the SAML single sign-on protocol. In *WITS '05: Proceedings of the 2005 workshop on issues in the theory of security* (pp. 27–40). New York, NY, USA: ACM Press.

Jacquemard, F., Rusinowitch, M., & Vigneron, L. (2000). Compiling and Verifying Security Protocols. In M. Parigot & A. Voronkov (Eds.), *Proceedings of LPAR 2000* (pp. 131–160). Springer-Verlag.

Lowe, G. (1996). Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In T. Margaria & B. Steffen (Eds.), *Proceedings of TACAS'96* (pp. 147–166). Springer-Verlag.

Lowe, G. (1997). A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)* (pp. 31–43). IEEE Computer Society Press.

Meadows, C. (1996). The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, *26*(2), 113–131. (See `http://chacs.nrl.navy.mil/projects/crypto.html`)

Millen, J. K., & Shmatikov, V. (2001). Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the ACM conference on computer and communications security CCS'01* (p. 166-175).

Mödersheim, S., & Viganò, L. (2009). The Open-source Fixed-point Model Checker for Symbolic Analysis of Security Protocols. In *Fosad 2007-2008-2009* (Vol. 5705, pp. 166–194). Springer-Verlag.

Needham, R. M., & Schroeder, M. D. (1978). *Using Encryption for Authentication in Large Networks of Computers* (Tech. Rep. No. CSL-78-4). Palo Alto, CA, USA: Xerox Palo Alto Research Center. (Reprinted June 1982)

OASIS. (2005a, April). *Security Assertion Markup Language (SAML) v2.0.* (Available at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`)

OASIS. (2005b, July). *SSTC response to "Security Analysis of the SAML Single Sign-on Browser/Artifact Profile".* (Available at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`)

Rusinowitch, M., & Turuani, M. (2001). Protocol Insecurity with Finite Number of Sessions is NP-complete. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop.* IEEE Computer Society Press.

The AVANTSSAR Team. (s. d.). *The AVANTSSAR Project.* (`http://www.avantssar.eu/`)

Turuani, M. (2006). The CL-Atse Protocol Analyser. In *Term rewriting and applications (proceedings of RTA'06)* (p. 277-286).

US-CERT. (2008, Sep 2). *Vulnerability Note VU#612636 - Google SAML Single Sign on Vulnerability.* (`http://www.kb.cert.org/vuls/id/612636`)