From Multiple Credentials to Browser-based Single Sign-On: Are We More Secure?

Alessandro Armando[1,2], Roberto Carbone[2], Luca Compagna[3], Jorge Cuellar[4], Giancarlo Pellegrino[3], and Alessandro Sorniotti[5]

[1] DIST, Università degli Studi di Genova, Italy
[2] Security & Trust Unit, FBK, Trento, Italy
[3] SAP Research, Mougins, France
[4] Siemens AG, Munich, Germany
[5] IBM Research Zurich, Rüschlikon, Switzerland

Abstract. Browser-based Single Sign-On (SSO) is replacing conventional solutions based on multiple, domain-specific credentials by offering an improved user experience: clients log on to their company system once and are then able to access all services offered by the company's partners. By focusing on the emerging SAML standard, in this paper we show that the prototypical browser-based SSO use case suffers from an authentication flaw that allows a malicious service provider to hijack a client authentication attempt and force the latter to access a resource without its consent or intention. This may have serious consequences, as evidenced by a Cross-Site Scripting attack that we have identified in the SAML-based SSO for Google Apps: the attack allowed a malicious web server to impersonate a user on any Google application. We also describe solutions that can be used to mitigate and even solve the problem.

1 Introduction

To provide access to restricted services, web applications assign digital credentials to registered users and require users to prove possessions of these credentials to receive access to protected resources. As web applications become more and more widespread, users must handle an increasing number of authentication credentials to establish security contexts with web applications. This is not only an annoying aspect of the current state of affairs, but has serious implications on the security of these systems as users tend to use weak passwords and/or to reuse the same password on different web applications.

Browser-based SSO solutions aim at improving this state of affairs by allowing users to log in once and by giving them subsequent access to multiple web applications. At the core of a browser-based SSO solution lies a browser-based

This work has partially been supported by the FP7-ICT Projects AVANTSSAR (no. 216471) and SPACIOS (no. 257876), and by the project SIAM funded in the context of the FP7 EU "Team 2009 - Incoming" COFUND action. Furthermore the authors would like to thank Brian Eaton, Scott Cantor, Matteo Grasso, and the SAP NetWeaver SIM team for the valuable discussions and feedback they provided.

authentication protocol. Three roles take part in the protocol: a client (C), an identity provider (IdP) and a service provider (SP). The objective of C, typically a web browser guided by a user, is to get 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 decide on C's entitlement to the requested resource.

A number of solutions for browser-based SSO have been put forward, e.g. Microsoft® Passport [11], the Liberty Alliance project [12], the Shibboleth Initiative [9], and OpenId [15]. The OASIS *Security Assertion Markup Language* (SAML) 2.0 Web Browser SSO Profile (SAML SSO, for short) [13] is an 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 must be exchanged in a variety of applications and/or deployment scenarios. Prominent software companies base their SSO implementations on SAML SSO. For example, Google has developed a SAML-based SSO service for its popular web applications (namely Gmail, Google Calendar, Talk, Docs and Sites), called the SAML-based SSO for Google Apps [5].

The security of SAML SSO critically relies on a number of assumptions on the trustworthiness of the principals involved as well as on the security of the transport protocols used to exchange messages. In this paper we argue that one of the assumptions on the security of the transport layer (i.e., that communication between the client and the service provider must be carried over a unilateral SSL 3.0 or TLS 1.0 connection) can only be met in practice in a way that leaves the protocol vulnerable to an authentication flaw. We discuss how this flaw can be exploited in general as well as on a number of prominent SAML-based SSO solutions, including the SAML-based SSO for Google Apps that is used by over one million business customers. Our findings show that the authentication flaw can be seriously exploited in actual deployments of SAML SSO. For instance, a severe attack could be mounted on the SAML-based SSO for Google Apps in which a malicious web server could impersonate the victim user on any Google application. In the paper we also provide solutions that allow the authentication flaw and its exploitations to be mitigated or even eliminated.

To the best of our knowledge neither the authentication flaw on the SAML SSO nor the vulnerability of the SAML-based SSO for Google Apps reported in this paper are publicly known. We are currently informing US-CERT and the relevant vendors about our findings. In response to our vulnerability report Google has already patched their implementation of their SAML SSO solution.

What about the original question in the title, are we more secure with SAML SSO than with multiple credentials? This question does not have a trivial answer and certainly a positive answer cannot be given as long as there are unaddressed issues such as the vulnerability we present in this paper. In addition, since the security considerations brought forward by this paper do not apply to SAML SSO only, we believe that other browser-based SSO protocols may suffer from similar vulnerabilities. We are currently extending our analysis to other SSO solutions to ascertain this.

Structure of the paper. In the next section we introduce the SAML SSO profile for web-based authentication. In Section 3 we present the authentication flaw on the SAML SSO, and in Section 4 describe how it can be exploited on actual implementations. In Section 5 we provide a number of solutions for the flaw. Last but not least, in Sections 6 and 7 we discuss some of the related work and present conclusions.

2 The SAML 2.0 Web Browser SSO Profile

SAML SSO provides a standardized, open, interoperable SSO solution applicable in a multitude of environments and situations, and can therefore be instantiated according to the specific requirements posed by the application scenario. In this paper we focus on one of its most widely used instantiations, the SP-Initiated SSO with Redirect/POST Bindings, whose typical use case is described in [14]. In the remainder of this paper we will refer to this use case as *the SAML SSO use case* and to the associated protocol as *the SAML SSO Protocol*.

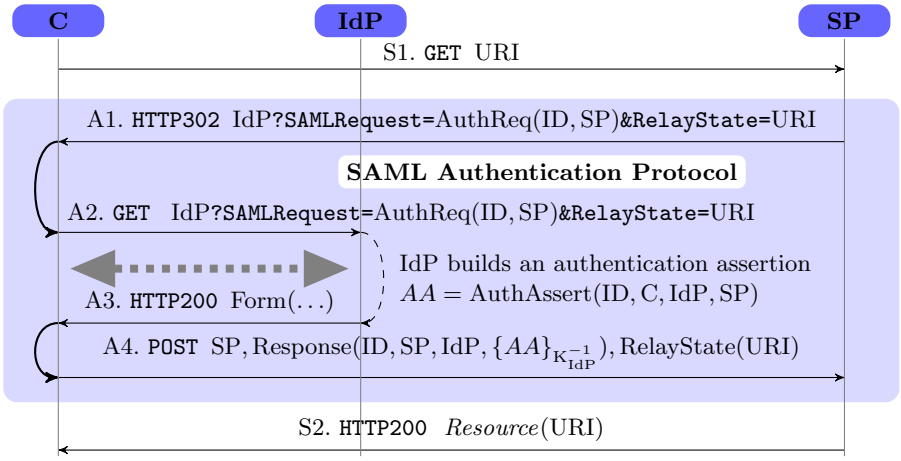

Fig. 1. SAML SSO Protocol: SP-Initiated SSO with Redirect/POST Bindings

In Figure 1 we capture the most important steps of the *SAML SSO Protocol*, abstracting away the steps that are irrelevant for our analysis, such as—among others—the IdP discovery phase. In step S1, C asks SP to provide the resource located at URI, say *Resource*(URI), without having a valid, active logon session (i.e. security context) with SP. SP then initiates the *SAML Authentication Protocol* by sending to C an HTTP redirect response (status code 302) to IdP, containing an authentication request AuthReq(ID,SP), where ID is a (pseudo-)randomly generated string uniquely identifying the request (steps A1 and A2). A frequent implementation choice is to use the `RelayState` field to carry the original URI that the client has requested (see [14]).

If C does not have an existing security context with the IdP, then IdP challenges C to provide valid credentials. If the authentication succeeds, the IdP creates the local security context, builds an authentication assertion as the tuple $AA = \text{AuthAssert}(\text{ID}, \text{C}, \text{IdP}, \text{SP})$, and places it in a response message $Resp = \text{Response}(\text{ID}, \text{SP}, \text{IdP}, \{AA\}_{\text{K}_{\text{IdP}}^{-1}})$, where $\{AA\}_{\text{K}_{\text{IdP}}^{-1}}$ is the assertion signed with $\text{K}_{\text{IdP}}^{-1}$, IdP's private key. IdP then places *Resp* and the value of `RelayState` received from the SP into an HTML form (indicated as Form(...) in Figure 1) and sends the result back to C in an HTTP response (step A3) together with some script that automatically posts the form to the SP (step A4). This completes the SAML Authentication Protocol. SP can then deliver the requested resource, *Resource*(URI), to C (step S2), and the SAML SSO Protocol completes as well.

Note that the steps at message S1 and S2 admittedly fall outside of the scope of the standard, and their implementation is left free. In this paper we capture steps S1 and S2 as described in *the SAML SSO use case*; a number of commercial SAML SSO solutions indeed adopt similar approaches to implement those steps.

As pointed out in [2] the security of the protocol critically relies on (unstated) assumptions about the trustworthiness of the participants involved and about the transport protocols used to exchange the protocol messages; we shall review these in the next Sections.

2.1 Trust and Transport Protocol Assumptions

The above protocols work under the assumption that *(i)* IdP is not compromised, i.e. it is not under the control of an intruder and it abides by the rules of the protocol and *(ii)* IdP is trusted by SP to generate authentication assertions about C. Even if they are not explicitly stated in the SAML 2.0 specifications, these are very reasonable assumptions to make and, in fact, both protocols are useless if the IdP is not trusted to generate authentication assertions about C or if there is the doubt that the IdP is compromised. However, *we do not assume that all SPs which C may play the protocol with are uncompromised.* In other words, unlike [8], we want to consider also those situations in which C runs the protocol with compromised SPs in order to determine whether they affect the security of sessions of the protocol played with other uncompromised SPs. This is very important as SPs are usually managed by different organizations that do not always share trust relationships.

The SAML 2.0 specifications repeatedly state the following assumptions of the transport protocols used to carry the protocol messages:

(TP1) Communication between C and SP is carried over a unilateral SSL 3.0 or TLS 1.0 channel (henceforth called SSL), established through the exchange of a valid certificate (from SP to C).

(TP2) 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).

2.2 Security Requirements

The SAML specifications do not explicitly state the security properties that the SAML SSO Protocol and the SAML Authentication Protocol are expected to achieve. By comparison with classic web authentication schemes, it is however natural to expect that at the end of the *SAML SSO Protocol*, the following security property is fulfilled:

(P1) SP and C mutually authenticate and agree on the value URI

As pointed out in [10], different definitions of authentication are possible. The notion of authentication we consider in this paper includes *recentness*, i.e. the fact that the principal being authenticated recently took part in the protocol run so as to exclude replay attacks.

We note that the SAML Authentication Protocol, the building block of the SAML SSO Protocol, is only able to guarantee the property

(P2) SP authenticates C

The converse is not true, i.e., the SAML Authentication Protocol does not provide to C any guarantee on SP's identity; indeed in message A1, SP may instruct IdP to force C to redirect message A4 to an arbitrary location. Even the use of SSL certificates only guarantees that there is no man-in-the-middle in the communications between C and the recipient of message A4.

In the remainder of this paper, we will investigate whether the SAML SSO Protocol, constructed with a building block that only guarantees **(P2)**, is able to fulfill the original property **(P1)**, and we will show that the fulfillment of this property is not automatically guaranteed; in particular depending on the implementation choices, a malicious SP may be able to hijack C's authentication attempt and force the latter to access a resource without its consent or intention.

3 An Authentication Flaw in the SAML SSO Protocol

An analysis of the SAML specifications reveals that the standard does not specify whether the messages exchanged at steps S1 and A4 must be transported over the same SSL channel or whether two different SSL channels can be used for this purpose. In other words, there is a certain degree of ambiguity on how assumption (TP1) of Section 2 can be interpreted.

The reuse of the SSL channel established at step S1 to also transport the message at step A4 is at first sight the most natural option. However this is difficult to achieve in practice for a number of reasons:

Resuming SSL sessions. The use of a single SSL session for the exchange of different messages cannot be guaranteed as, e.g., the underlying TCP connection might be terminated (e.g. timeout, explicitly by one of the end points), an SSL

server could not resume a previously established session, or a client might be using a browser that very frequently renegotiates its SSL session.

Software modularity. Nowadays, software is designed to be increasingly modularized, capitalizing on layering and separation of concerns. This may result in the fact that—within SP implementations—the software module that handles SAML messages has no access to the internal information of the transport module that handles SSL. Thus, the information on whether the client has used a single SSL session or two different ones may not be available.

Distributed SPs. The SAML SP may be distributed over multiple machines, for instance, for work-balancing reasons. This results in physically different SSL endpoints, with the inherent impossibility of enforcing a single session for all communications between SP and C.

We have extended the formal model discussed in [2] to faithfully capture the SAML SSO use case in which the messages of steps S1 and A4 can be transported over different SSL sessions and fed it to a state-of-the-art model-checker for security protocols [1]. (See Section 6 for more details.) The model checker detected the attack depicted in Figure 2, thereby witnessing a violation of property **(P1)** in the *SAML SSO Protocol*.

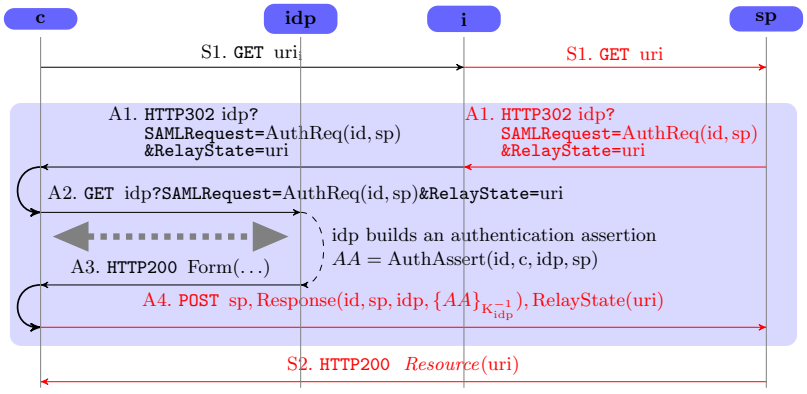

Fig. 2. Authentication Flaw of the SAML 2.0 Web Browser SSO Profile

The attack involves four principals: a client (c), an honest IdP (idp), an honest SP (sp) and a malicious service provider (i). The attack is carried out as follows: c initiates the protocol by requesting a resource uri_i at SP i. Now i, pretending to be c, requests a different resource uri at sp and sp reacts according to the standard by generating an Authentication Request, which is then returned to i. Now i maliciously replies to c by sending an HTTP redirect response to

See, for instance, `http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame2.doc_5.1/am51_webseal_guide54.htm`

idp containing AuthReq(id, sp) and uri (instead of AuthReq($\mathrm{id_i}$, i), and $\mathrm{uri_i}$ as the standard would mandate). The remaining steps proceed according to the standard. The attack makes c consume a resource from sp, while c originally asked for a resource from i.

Note that the attack is possible essentially because the client—usually a normal browser with no knowledge of the SAML protocol—has no means of verifying whether the authentication request and the authentication assertion are related to the initial request.

Interestingly enough, standard username/password authentication mechanisms do not suffer from this authentication flaw. To see this, let us assume that C has no active sessions with service providers SP1 and SP2; let us also assume that C's usernames and passwords are different for each SP. Then under no circumstance can SP1 hijack C's authentication attempt and unawarely and automatically force it to consume a protected resource at SP2. From this point of view, the advantage of domain-specific credentials in the control of the user is that the user knows exactly for whom the credentials are intended upon providing them. With SSO, "binding" the views of the user and of the service provider is not so easy.

Note that the attack would be prevented if sp could enforce that the initial request and authentication response are carried over the same secure channel, but we have previously explained why this requirement is very difficult to achieve in practice. Note also that requiring digitally signed authentication requests would not fix the vulnerability; indeed the authentication request is actually generated by the honest service provider, and only blindly forwarded to the client by the attacker; the signature is therefore valid and will be accepted.

Even more interestingly, the attack does not strictly require a malicious service provider in order to be successful. Any malicious web server i would be able, upon a request from c, to mount the attack provided that *(i)* c is a client of sp and *(ii)* c has an active authentication context with idp.
The attack in Figure 2 can be exploited in a number of ways:

Delivery of an unrequested resource. The most trivial exploitation of the flaw consists in the attacker forcing the client to receive a different protected resource from the initially requested one. The same exploitation may also be mounted if a malicious web server redirects the browser to a legitimate SP before the SAML SSO Protocol starts. However this attack can be prevented by using well-known browser-side plugins that restrict HTTP redirections (e.g., the NoRedirect addon for Firefox). By allowing only IdP-to-SP and SP-to-IdP redirections, the delivery of an unrequested resource upon redirection outside of the SAML SSO Protocol is prevented, but a malicious SP can still mount the one depicted in the Figure 2.

Launching pad for cross-Site Request Forgery (XSRF) attacks. This attack assumes that the URI that was initially requested did not point to a resource, but rather contained a URL-encoded command, such as a request for

If this assumption does not hold, C is vulnerable to a number of other trivial attacks anyway.

the change of some settings or user's preferences, for the deletion of some resource or for the annulment of/committing to an action, such as the purchase of a paid good. Depending on the output provided by the execution of the command, the client may or may not be able to detect the attack. This type of attack is even more pernicious than classic XSRF, because XSRF requires C to have an active session with SP, whereas in this case, the session is created automatically hijacking C's authentication attempt.

Launching pad for cross-Site Scripting (XSS) attacks. It is straightforward to see that this attack also constitutes a launching pad to reflected XSS attacks, i.e. XSS attacks that can be triggered by visiting a maliciously-crafted URL. In addition, a vanilla implementation of the SAML SSO protocol exposes the `RelayState` field to a possible injection of malicious code that may be executed at the honest SP side. Although the SAML standard recommends to protect the integrity of this field, our experience shows that this often is not the case (see Section 4). In addition, unlike normal XSS attacks, where the attacker has to rely on social engineering (phishing, spam and so forth) to lure a victim into clicking on a malicious link, an exploitation of the vulnerability paves the way for systematically luring victims into visiting URIs that may be vulnerable to cross-site scripting attacks. Note also that in this case, unlike in the previous exploitations, the client is not suspicious about receiving a different resource than the one requested. On the contrary, because arbitrary code can be embedded in uri, a redirection to uri_i, the page that c initially requested, can be eventually forced at the end of the attack. As an example, if uri is forged as `javascript:window.open('`uri_i`'+document.cookie)` the client would be victim of the theft of its cookies for the domain sp through a visit to the requested uri_i.

Although in this paper we focus on the SP-initiated SSO protocol, it is worth mentioning that IdP-Initiated flows may suffer from *login CSRF* attacks [3], whereby the attacker forges a cross-site request to the login form and, logs the victim into a honest web site *as the attacker*.

4 Exploitations in actual deployments

An interesting question that we also address in this paper is whether exploitations of the abstract weakness of the standard are possible in actual deployments of the SAML SSO Protocol. To this end, we have analyzed various SAML-based SSO solutions available on the market, including SAML-based SSO for Google Apps, SimpleSAMLphp as deployed for Foodle (see `https://foodl.org`), and a deployment of the Novell Access Manager 3.1 in a real industrial environment. All these deployments support the SAML SSO use case; not surprisingly, by inspecting SSL messages we verified that the SPs employed in these deployments accept and process a SAML response flowing on a different SSL channel than the one used to deliver the SAML request.

Our analysis of the SAML-based SSO for Google Apps shows that by exploiting the weakness of the standard, a malicious SP can force C to consume a

resource from Google, for instance, visiting any page of the gmail service. mailbox. This trivial attack is however easily detected by C, and does not bring any real advantage to the attacker. Definitely more serious for the over one million business customers of Google Apps was the XSS attack we were able to execute and that allowed the malicious SP to steal the C' cookies for the Google domain and thus to impersonate C on any Google application. The abstract flaw of Figure 2 served indeed as launching pad for this XSS: because of missing sanitization, an attacker could inject malicious code into the `RelayState` field and have it successfully executed on the client's browser as if coming from the Google domain (thus circumventing the same origin policy). In other words, the combination of the abstract flaw and the missing sanitization was the key to mount the XSS attack. The past tense is in order here since, as soon as we found this attack, we informed Google, who promptly patched the issue.

We have been able to mount a similar XSS attack on the SAML SSO solution of the Novell Access Manager 3.1 as deployed in a real industrial environment. In this deployment `RelayState` is not used to store the URI;instead, a URL-encoded parameter is used to this end, and this parameter is not sanitized.

The SimpleSAMLphp, as deployed in Foodle, stores the initially requested URI into the URL parameter `ReturnTo`. Although that field is not sanitized, we have not been able to mount any XSS. The reason is that SPs running SimpleSAMLphp additionally use cookies that block the abstract flaw we discovered. We will detail this solution in the next section.

The findings presented above show that the authentication flaw we discovered can be exploited on actual deployments of the SAML SSO Protocol, even leading to major security issues. We have informed Novell and UNINETT (the developer and maintainer of SimpleSAMLphp) about these findings as well as the US-CERT so that other vendors implementing and deploying SAML-based SSO solutions can get advantage of this information.

5 Fixing the vulnerability

The root of the problem of the authentication flaw presented earlier lies in the following two main factors:

1. Clients are not able to link the Authentication Request they receive from the SP in step A1 with their initial requests for a resource issued in step S1;
2. The SP is not able to enforce that the messages exchanged with C (cf. steps S1, S2, A1, and A4) are carried over the same channel.

We have verified that—could one of the two causes be removed—the vulnerability would no longer be exploitable. We emphasize nonetheless that the *SAML SSO Protocol* alone neither achieves property **(P1)** nor mandates the implementation of any of these solutions, thus leaving a vanilla implementation in principle flawed.

The challenge is to fix the vulnerability with minimal changes so that existing solutions can be secured without radical modifications to the software compo-

nents (e.g. SAML ECP profile) or to the standard. In what follows, we outline a number of possible solutions, highlighting their strengths and shortcomings.

Cookies. A standard way of enforcing bindings on sessions is implemented using session cookies. With reference to Figure 1, by setting a session cookie in step A1 and expecting to receive it back on message A4, SP could check that the communication has occurred with the same client. However, cookies only provide means to mitigate the problem—albeit sufficient in many scenarios—and do not represent a complete countermeasure. Indeed cookies are designed to be difficult to steal and it is not as hard to set them. For instance, cookies with the "Secure" flag on (which instructs the browser not to transmit them over unencrypted channels) can be set over unprotected connections. (The latest versions of IE and of Firefox allow this.) In practice an attacker could circumvent the protection offered by cookies by *(i)* setting a cookie for the victim SP through injected Javascript or HTML META tags; *(ii)* corrupting the proxy discovery phase setting up a rogue wpad or dhcp server, thus becoming the user's proxy; *(iii)* performing ARP poisoning thus becoming the victim's default gateway.

Feedback from the user. As seen in the preceding Sections, the user may initiate the SAML SSO profile, authenticating to an SP without actually having explicited requested anything from it. This can be avoided if the IdP informs the user about the attempt to access URI on the SP during the authentication and asks for an explicit consent before issuing the authentication assertion to SP. In this way, the user may realise that the authentication is going to be sent to a different SP than expected and may be given the possibility to stop the protocol. This solution has a number of drawbacks: first of all, it forces a security decision upon a (possibly technically unaware) user, who is asked to tell apart legitimate SP-to-SP redirections from malicious ones. In addition, it breaks the seamlessness of SSO, in which the authentication process is supposed to be carried out with minimal interactions with the user.

Self-signed client certificates. A simple, yet effective way to ensure SP that it is interacting with the same client is to provide the latter with a self-signed certificate. The solution goes as follows: during the first SSL session (cf. steps A1 and A2 in Figure 1) C is asked to present the certificate. SP will then generate an Authentication Request and its ID field is set to $n \,||\, \texttt{HMAC}_{K\,||\,n}(\texttt{RSA modulus})$, where n is a nonce, K is a secret known only to SP, `RSA modulus` is the RSA modulus of the public key contained in the client's certificate. HMAC is the well-known HMAC keyed hash function [4] and $||$ denotes the concatenation. After this, SP deletes all state information and sends the Authentication Request to C. During the second SSL session (cf. steps A3 and A4), C is again asked for the certificate and the same certificate will be delivered to SP. The standard requires the *InResponseTo* field of the Response message to contain the same value of the ID field of the Authentication Request message: therefore SP can parse such field as n' and H' and then check whether $H' = \texttt{HMAC}_{K\,||\,n'}(\texttt{RSA modulus})$.

Note that *(i)* the client can easily self-generate a certificate; alternatively, the SP can offer the client to forge one on his behalf; *(ii)* the certificate is *not*

expected to carry information about the identity of the user; in particular, it is not used to assess the user identity; and *(iii)* during the SSL handshake, the browser proves knowledge of the private key; the approach therefore guarantees with overwhelming probability that a malicious third party cannot forge a copy of the same certificate since – in case of certificates that use RSA encryption for instance – it would entail breaking the RSA hardness assumption.

6 Related Work

Pfitzmann et al. [16, 17, 7] lay the theoretical basis for a rigorous analysis of web-based federated identity-management protocols (e.g. the SSO protocol proposed by Liberty Alliance in 2002). They discuss some security vulnerabilities and possible preventive measures. Some of these results have been fed into the Liberty Alliance project and indirectly into the SAML 2.0 standard.

Security analyses of the SAML SSO v1.0 are presented in [6] and in [8]. The security analysis presented in our paper refers to SAML SSO v2.0, the latest version of the standard. Moreover, in our work we focus on scenarios that are most likely to occur in actual deployments. For instance, unlike [8] we do not assume that SPs are trustworthy and unlike [6] we assume that messages are exchanged over secure channels as recommended by the standard.

In [2] we provide a formal model of the SAML SSO protocol as well as of a variant implemented in the SAML-based SSO for Google Apps. By using a model checker, we discovered a subtle man-in-the-middle attack on the SAML-based SSO for Google Apps. In reaction to this discovery Google has modified the implementation of the protocol. The version of the protocol used by the SAML-based SSO for Google Apps we described in Section 4 is the one currently in use by Google and therefore does not suffer from the attack reported in [2]. Interestingly, in [2] we did not find any attack on the Web Browser SAML 2.0 SSO profile as in our analysis we assumed that communication between C and SP is carried over a single unilateral SSL channel. We have adapted that formal model so to allow the messages of steps S1 and A4 to be transported over different SSL sessions and used the SATMC model-checker to analyze this new specification. This has allowed us to discover the previously unknown attack described in Section 3.

7 Conclusions

Authentication protocols are notoriously difficult to get right, even more so for browser-based authentication protocols because "browsers, unlike normal protocol principals, cannot be assumed to do nothing but execute the given security protocol" [7]. In this paper we have showed that browser-based SSO protocols are no exception. We have presented an authentication flaw in the SAML SSO, discussed how this flaw can be generally exploited, and reported related security issues that we have detected in actual SAML-based SSO solutions developed by prominent software companies, including a severe attack on the SAML-based

SSO for Google Apps. We have finally presented a number of possible solutions that mitigate or even solve the problem. As a part of our future work we plan to extend our analysis to other SSO solutions.

References

1. A. Armando, R. Carbone, and L. Compagna. LTL Model Checking for Security Protocols. In *Journal of Applied Non-Classical Logics, special issue on Logic and Information Security*, pages 403–429. Hermes Lavoisier, 2009.
2. A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In *FMSE*. ACM, 2008.
3. A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In *15th ACM Conference on Computer and Communications Security (CCS 2008)*, 2008.
4. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *Advances in Cryptology CRYPTO 96*, volume 1109 of *LNCS*, pages 1–15. 1996.
5. Google. Web-based SAML-based SSO for Google Apps. `http://code.google.com/apis/apps/sso/saml_reference_implementation_web.html`, 2008.
6. T. Groß. Security analysis of the SAML Single Sign-on Browser/Artifact profile. In *Proc. 19th Annual Computer Security Applications Conference*. IEEE, Dec. 2003.
7. T. Groß, B. Pfitzmann, and A.-R. Sadeghi. Browser model for security analysis of browser-based protocols. In *ESORICS*, 2005.
8. S. M. Hansen, J. Skriver, and H. R. Nielson. Using static analysis to validate the SAML single sign-on protocol. In *WITS '05*, New York, NY, USA, 2005. ACM Press.
9. Internet2. Shibboleth Project. `http://shibboleth.internet2.edu/`, 2007.
10. G. Lowe. A hierarchy of authentication specifications. In *Proc. CSFW*. IEEE, 1997.
11. Microsoft. Windows Live ID. `https://www.passport.net/`.
12. OASIS. Identity Federation. Liberty Alliance Project. `http://www.projectliberty.org/resources/specifications.php`, 2004.
13. OASIS. SAML V2.0. `http://docs.oasis-open.org/security/saml/v2.0/`, April 2005.
14. OASIS. SAML V2.0 – Technical Overview. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`, March 2007.
15. OpenID Foundation. OpenID Specifications. `http://openid.net/developers/specs/`, 2007.
16. B. Pfitzmann and M. Waidner. Analysis of Liberty Single-Sign-on with Enabled Clients. *IEEE Internet Computing*, 7(6), 2003.
17. B. Pfitzmann and M. Waidner. Federated identity-management protocols. In *Security Protocols Workshop*, 2003.