RZ 3643 (# 99653) 01/03/06
Computer Science 17 pages

Research Report

SAML Artifact Information Flow Revisited

Thomas Groß and Birgit Pfitzmann

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland
{trg, bpf}@zurich.ibm.com

IBM Research
Almaden · Austin · Beijing · Delhi · Haifa · T.J. Watson · Tokyo · Zurich

SAML Artifact Information Flow Revisited

Thomas Groß
IBM Zurich Research Lab
Rüschlikon, Switzerland
tgr@zurich.ibm.com

Birgit Pfitzmann
IBM Zurich Research Lab
Rüschlikon, Switzerland
bpf@zurich.ibm.com

Abstract

The standardized OASIS Security Assertion Markup Language (SAML) has become one of the most deployed frameworks in federated identity management even though it focuses only on single sign-on. Answering industry's pursuit of the reduction of user-management costs and enabling cost-efficient deployment because of its browser-based profiles, SAML is believed to become widely used soon. With the revision to Version 2.0, especially SAML's browser/artifact profile has gained new security measures defeating old vulnerabilities. We analyze this profile and focus on the problem of artifact information flow. We devise a concrete exploit to demonstrate the impact of this problem. We address this problem by a new browser/artifact profile called Janus. The innovation is to split the artifact into two independent shares that have different information flow in a standard web browser. This new method defeats artifact information flow efficiently without relying on assumptions on the artifact lifetime.

1 Introduction

One of the recent advances of access control and user management products was the introduction of federated identity management proposals such as the Security Assertion Markup Language (SAML) [16]. Industry expects a dramatic reduction of user management costs from federated identity management by savings in password helpdesks, user management, and user deletion.

SAML features browser-based profiles that only rely on a standard web browser to carry out identity federation, e.g., by means of single sign-on. These protocols complement the general advantages of federated identity management solutions with the property of being zero-footprint, i.e., not requiring installation of additional client software. Therefore, browser-based profiles are cost-efficient to deploy. However, designing secure protocols with a standard web browser as the client is not trivial. The browser, not being aware of the protocol it participates in, has a predefined behavior, reacts to predefined messages and generates information flow both to the underlying operating system and to communication partners. Especially the security of protocols that transfer confidential information through a browser's URL is put at stake by this protocol-unaware behavior of a standard web browser. The browser/artifact profiles of SAML belong to this class of protocols, because they issue a random artifact as reference to a security token and transports it via the browser redirect URL.

The SAML V1.1 Web SSO Browser/Artifact Profile [16] was already analyzed by [6] and suffered from some problems introduced by a standard web browser as client. In the meantime, SAML has advanced to Version 2.0 [22] and revised also this profile, repairing most of the problems discussed in [6] and discussing the improvements in an SSTC response [14]. The structure and naming in the standards has also slightly changed, hence the corresponding protocol (in the terminology of security protocol research) is now the SAML V2.0 Web Browser SSO/Response/Artifact Feature. In the meantime, research on federated identity management protocols and in particular browser-based protocols has also advanced. Microsoft Passport [15] was the first protocol to be analyzed. Its detailed analysis by [12] also discussed inherent problems of browser-based protocols. Inherent problems of browser-based client authentication were also discussed in [4] independent of the federation aspect. Liberty [13] is an identity federation protocol that is based upon SAML V1.1, yet, has also influenced the development of SAML V2.0. Weaknesses in the original version of the Liberty enabled-client profile were found by [23] and repaired in subsequent Liberty versions. Shibboleth [3] is a federated identity management solution for universities also based upon SAML. Research also started to provide positive security statements for federated identity management protocols; a profile of WS-Federation [10, 11] was analyzed first by [7] based upon top-down assumptions and without a detailed browser model. Very recently a generic model for the analysis and security proofs of browser-based protocols was proposed [8] that is to be used for more in-depth proofs such as [9]. However, there is no positive statement for a browser/artifact profile such as the SAML V2.0 Web Browser SSO/Response/Artifact Feature yet. Research in this area is complemented by the tool-supported analysis of standards such as in the Web Services area, which starts with [5, 2, 1]. However, protocols involving a standard web browser were not considered by this approach.

Our Contribution. We analyze the security measures newly introduced by version 2.0 into the browser/artifact profile of SAML [22, 14]. We discuss the generic security goals for the profile and their impact on the scope of the SAML specification. We point out the still existing problem of artifact information flow and construct a concrete exploit for a specific scenario to demonstrate its impact. Furthermore, we turn to the general problem of information flow of SAML artifacts through a standard web browser. This problem is inherent to the whole protocol class with artifacts and may compromise the protocol security: if a valid artifact flows to an adversary, the adversary may impersonate the corresponding honest user. For instance, [6] proposed an attack based upon an adversary provoking information flow of an artifact through the browser's Referer tag. Currently, there is no solution that fully solves this problem. Though the SAML V2.0 Web Browser SSO/Response/Artifact Feature strengthened the protection against such information flow by introducing a one-time request constraint at the service provider, it can still be compromised by information flow of the SAML artifact, which can still happen in specific scenarios. The potential resulting damage is only reduced by recommendations about the artifact's lifetime, assumptions about the clock skew between protocol principals, and the reasoning that an adversary cannot cause too much harm within that timeframe. We discuss solutions to this problem that do not rely on timing assumptions. Furthermore, we devise a new variant of the SAML V2.0 Web Browser SSO/Response/Artifact Feature that renders a potential artifact information flow via the Referer tag unusable by an adversary. We do this by splitting a SAML artifact into two independent shares that produce different information flow in a standard web browser. Thus, we introduce a completely new approach into the options for solving the artifact information flow problem that does not rely on timing. Its advantages are that it uses neither additional messages in most cases, and thus does not introduce new latency into the profile, nor storage-expensive measures like artifact blacklists.

2 Protocol Overview

In this section, we give an overview of the SAML V2.0 Web Browser SSO/Response/Artifact Feature. SAML is a multi-part standard. The core [17] defines SAML assertions, which correspond to security tokens or credentials in other terminologies. An assertion has an issuer, typically a subject about whom the issuer asserts something, and optional elements like signatures and conditions. The core also contains request-response message pairs that can transport assertions. The pair of AuthnRequest and Response transport a request for authentication and the resulting one or more assertions. ArtifactResolve asks for the real response referred to by an artifact and ArtifactResponse delivers this response. The SAML bindings [18] bind such messages to concrete transport protocols such as HTTP. The Artifact Binding describes how first an artifact is transported through a browser and then an actual SAML message is retrieved directly, using the artifact as a reference. For the artifact, transport both in the redirect URL and via a form POST must be implemented. The SAML profiles [20] define entire sequences of message exchanges, for instance for single sign-on. Due to reference to the bindings, these profiles are rather modular. According to the SAML conformance specification, the combination of profile, message exchange and a selected binding is termed a SAML V2.0 feature. As the artifact information flow problem concerns an artifact binding of the Response message, we therefore call the analyzed profile(s) the SAML V2.0 Web Browser SSO/Response/Artifact Feature. This corresponds to a security protocol in the classical sense of security research. However, for a security analysis one has to remember that an adversary might try to replay message parts from one protocol in another. Furthermore, one has to remember that several steps in the protocol are not concrete algorithms, but only described by certain values to be generated and constraints on them. Similarly, the messages do not have one specific format, but a range of possible formats with constraints.

2.1 Notation

We assign identifiers to variables like identities and different URIs. We denote SAML's unique identifiers for entities as defined in [19, Section 2.2.1] by idi for identity providers and idp for service providers.[1] Likewise, we denote unique user identities as registered with identity providers by idu.[2] If a variable occurs at several participants, we prefix it with the participant whose view we discuss. E.g., , the assertion consumer service URI ACS of a service provider P in the view of this service provider is $\mathsf{P}.ACS$ while the view of the identity provider I of this URI in a SAML protocol run is $\mathsf{I}.ACS$. (Usually in such cases the security analysis shows that the variable values are indeed the same.) We omit these prefixes again if it is clear from the context which participant's view we are considering. We also use the dot notation for elements of structured messages, e.g., $art.handle$ for the handle field in an artifact art. By ϵ we denote that such a parameter is not present. Finally, we use the notation $U(Params)$ to denote a URI U augmented by a list of parameters $Params$ encoded in its querystring.

2.2 Protocol Steps

Figure 1 summarizes the SAML V2.0 Web Browser SSO/Response/Artifact Feature. As indicated in the figure, we show the simpler redirect binding for the request, and the entire request phase is optional. We only show those parameters and constraints that will be most important later.

In Step 2, we show a message AuthnRequest as a variable $AuthnReq$. Its most important parameters are $Issuer$, the request issuer, and a request identifier ID. How the service provider P generates $AuthnReq$ is relatively freely decidable; the only constraint is that the service provider has to set the issuer parameter to its own identity. The service provider P sends $AuthnReq$ via the browser to a single sign-on service address SSO of a desired identity provider I with identity idi.

In Step 4, the identity provider identifies the user by some means. We call the resulting user identity idu.

Then, for Step 5, the identity provider generates an artifact art. It can be abstracted as a tuple $(tc, ep, source\_id, handle)$, where tc is a (here) constant type code, ep an endpoint index, $source\_id$ is a SHA-1 hash of idi, and $handle$ is a randomly or pseudorandomly chosen 20-byte sequence. In the transport version with a URL-encoded artifact (which is mandatory to implement), the artifact is placed in a query string parameter named $SAMLart$, otherwise in a hidden form control. The identity provider addresses this redirect to the assertion consumer service URI ACS of the service provider indicated in the request; recall that $ACS(art)$ denotes ACS plus the artifact art encoded in the querystring. The values no-cache and no-store in this message refer to Cache-Control and Pragma header fields. [20, Section 4.1.3.5] stresses that the identity provider MUST have some means to establish that [the assertion consumer service] is in fact controlled by the service provider, which we model by the predicate $\mathsf{controls}(idp, ACS)$. In general, the predicate $\mathsf{controls}(id, URI)$ denotes that the participant with identity id controls the URI URI. This is a meaningful (although not fully specified) statement for SAML by the assumption that all participants in SAML have unique identifiers (see Section 2.1 and

[1] The uniqueness of these parameters is not defined in [17, Section 8.3.6], however, the Web SSO Browser profile stresses that the parameters in the profile's messages must be unique [20, Sections 4.1.4.1 and 4.1.4.2]. The metadata specification [19, Section 2.2.1] specifies that entity identifiers "MUST be unique across all entities that interact within a given deployment". Though it is optional to use the concrete methods for metadata publication of [19], we assume that its uniqueness constraint generally applies.

[2] Our assumption of uniqueness of the user identities is based upon two considerations. Firstly, [17, Section 2.4.1] stresses that "A $<$ Subject $>$ element SHOULD NOT identify more than one principal", which we interpret system-wide constraint. Secondly, [17, Section 2.2] discusses that SAML V2.0 provides name qualifiers to disambiguate name identifiers for different identities.

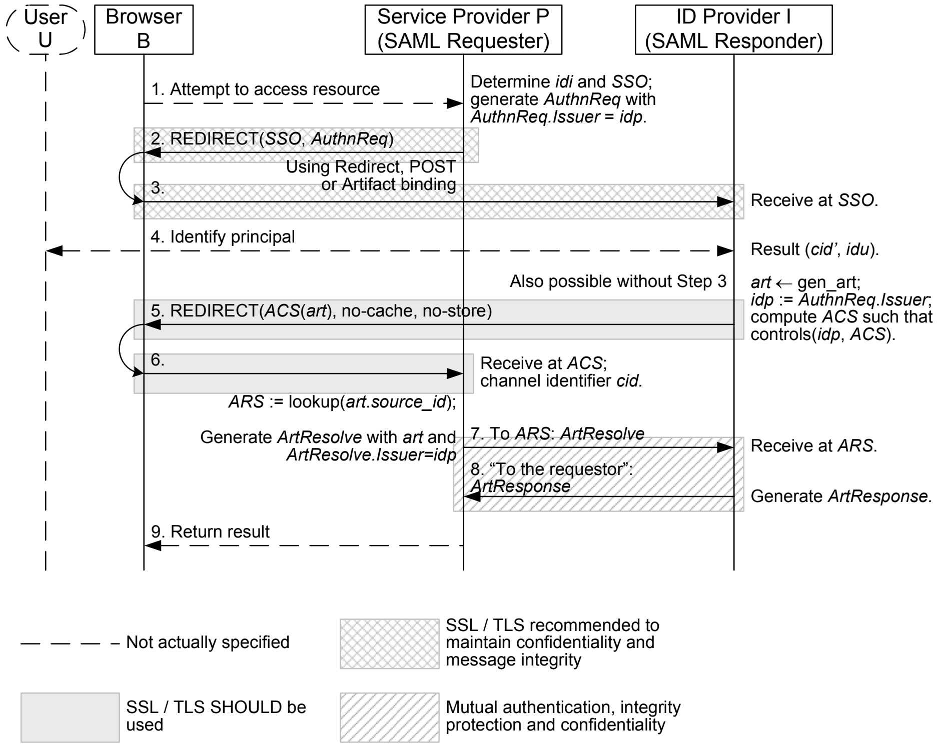
User U
Browser B
Service Provider P (SAML Requester)
ID Provider I (SAML Responder)
1. Attempt to access resource
Determine idi and SSO; generate AuthnReq with AuthnReq.Issuer = idp.
2. REDIRECT(SSO, AuthnReq)
Using Redirect, POST or Artifact binding
3.
Receive at SSO.
4. Identify principal
Result (cid', idu).
Also possible without Step 3
art ← gen_art; idp := AuthnReq.Issuer; compute ACS such that controls(idp, ACS).
5. REDIRECT(ACS(art), no-cache, no-store)
6.
Receive at ACS; channel identifier cid.
ARS := lookup(art.source_id);
Generate ArtResolve with art and ArtResolve.Issuer=idp
7. To ARS: ArtResolve
Receive at ARS.
8. "To the requestor": ArtResponse
Generate ArtResponse.
9. Return result
Not actually specified
SSL / TLS recommended to maintain confidentiality and message integrity
SSL / TLS SHOULD be used
Mutual authentication, integrity protection and confidentiality

Figure 1. SAML V2.0 Web Browser SSO/Response/Artifact Feature

especially Footnotes 1 and 2) and that one can evaluate whether an address belongs to a participant.[3] More precisely, we mean by "controls" that messages arriving at URI will be handled according to the constraints for SAML under identity id if the participant with this identity is honest.

In Step 7, the service provider asks the identity provider for the assertion corresponding to the artifact; this is done in an ArtifactResolve message $ArtResolve$. The function lookup models the lookup of an artifact resolution service address ARS under the $source\_id$ of an artifact. It is supposed to guarantee $\mathsf{controls}(idi, ARS)$, but not directly specified how to fulfill this constraint. The profile and the artifact binding prescribe mutual authentication, integrity protection, and confidentiality for Steps 7 and 8, and both state that this can be done by signing or by binding-specific measures (here the lower-level binding for these steps is meant, e.g., using SOAP). We will assume that this authentication refers to the identities idi and idp; certainly I has a local view $\mathsf{I}.idp$ of the service provider's identity associated with the artifact art from before Step 5. P can use $art.source\_id$ to look up idi.

As to message fields in Step 8, let Res be the Response element within $ArtResponse$. It is required that $Res.Issuer \in \{idi, \epsilon\}$ and $assert.Issuer = idi$ for every assertion $assert$ in Res. Furthermore, there must be at least one assertion, say $assert^*$, that contains an authentication statement, say sta^*, and such that $assert^*.confirm = \mathsf{bearer}$ and $assert^*.recipient = ACS$, where the parameter $confirm$ denotes the subject confirmation method. Furthermore, if there was Step 3, then $Res.InResponseTo = AuthnReq.ID$ is required. This constraint, however, cannot be verified in practice; if P receives a response Res with $Res.InResponseTo \neq \epsilon$, then P should check that there exists a valid $AuthnReq$ of P issued to idi with $Res.InResponseTo = AuthnReq.ID$.

There are lifetime and one-time use properties associated with artifacts and assertions which we will describe and analyze in Section 4.

[3] The requirement that entities can establish that a URL is controlled by another entity is made in the protocol description of the Web SSO browser profile [20, Section 4.1.3.5 and Section 3.1.4.1]. SAML provides the metadata specification [19] as one means to do so, however, its use is not prescribed, as stated in the Web SSO Browser profile [20, Section 4.1.3.5].

3 Security Goals

In this section, we define authenticity as the main security goal of the SAML V2.0 Web Browser SSO/Response/Artifact Feature. We precede this by a discussion of a suitable scope for our SAML analysis that reflects the SAML intention of modularity but nevertheless captures all features that might introduce vulnerabilities. We also describe several sub-goals of an adversary that imply breaking the authenticity requirement and are therefore sufficient for an adversary to reach.

3.1 Scope of SAML

We consider the correct authentication of a user to a service provider the main goal of federated identity management protocols that provide single sign-on. We follow the principle that such a protocol, and thus the SAML V2.0 Web Browser SSO/Response/Artifact Feature, should enforce the authenticity requirement in a self-contained way. This does not mean that it needs to spell out the implementation details of its submodules. However, in order to achieve authenticity it should only rely on explicit interfaces and assumptions about mechanisms used or the environment. We believe that this is essentially also what the SSTC response [14] is proposing, but with this principle in mind, our concrete analysis needs to cover a few aspects that are less explicitly covered in the SAML security considerations.

The SAML binding [18] as well as the SSTC response [14, Sections 1.2.1, 1.3.1, 1.4.2.1] rule the initial user authentication (including authentication of the identity provider to the user in the password-based case) and subsequent user tracking as out-of-scope. We reflect this in the upcoming Definition 3.1 by stating that SAML can only be as secure as the underlying user authentication at the identity provider. As to session tracking, we will assume that subsequent message exchanges with the same partner in a profile are securely tracked, but we would prefer this to be a more explicit requirement on implementers in the standard, in particular for the link between Steps 4 and 5.

The SSTC response [14, Section 1.3.6] defines Step 9 of the SAML V2.0 Web Browser SSO/Response/Artifact Feature as out-of-scope. However, we recommend not to do so entirely. Why? As shown in [6], in Step 9 and subsequent steps information flow of the SAML artifact may occur because of behavior inherent to the web browser. An adversary may be able to exploit such information flows to compromise authenticity. Hence, we recommend that a browser/artifact protocol must prevent all further information flow before the browser leaves the protocol's sphere of influence, or it must be analyzed under the assumption that the maximum possible information flow will occur in the unspecified next steps.

3.2 Authenticity of Single Sign-on

Authenticity means that if a service provider P has finished a SAML protocol run successfully, then it can be sure that its communication partner is a user with a certain received user identity idu and certain attributes att at identity provider I. To make the notion of "its communication partner" meaningful, we have to refer to a channel identifier cid, and a channel only guarantees that there is one fixed communication partner if it is secure. For SAML, we have to refer to the channel used in Step 6. In reality, what one wants is that the application-level use of this channel from Step 9 onwards is with this user, but user tracking at P from Step 6 to 9 is not prescribed in SAML. Furthermore, we have to specify how the service provider actually derives idu from $ArtResolve$, which may contain multiple assertions with multiple authentication statements. We assume that this is done by retrieving any assertion $assert^*$ fulfilling the conditions from Step 8 in Section 2.2, and deriving idu from $assert^*.Subject$. Thus we model the successful termination of a SAML protocol run by an output $(\texttt{accepted}, cid, idu, att)$, which binds the user identity idu to the secure channel identified by cid. As discussed in Section 3.1, the security of SAML is limited by the security of the initial user authentication and subsequent user tracking at identity provider I.

We make the authenticity definition for one given service provider and identity provider. As we mainly consider the security of the protocol as such, and not of naming and metadata issues, an extension to multiple trusted identity providers would add unnecessary complexity. Though not explicitly specified in SAML, we can assume that the different principals have means to enforce the SAML constraints about identifier and URI uniqueness and addressing, which we call setup.

Definition 3.1 (Authenticity) *Let* P *be an honest service provider and* I *its honest identity provider, and let* U *be an honest user with correct browser* B *who has the identity* idu *at the identity provider* I*. Let* U*,* P*, and* I *have executed setup according to the SAML specification. Then if the service provider* P *obtains an output* $(\mathsf{accepted}, cid, idu, att)$ *from the SAML V2.0 Web Browser SSO/Response/Artifact Feature, the secure channel with channel identifier* cid *is a channel with* U *(unless the user authentication and tracking of* U *at* I *is compromised by other means).* ◇

3.3 Adversary Goals

An adversary A may break the authenticity as defined in Definition 3.1 by reaching several sub-goals, which we define as adversary goals. We will only use one of these goals for an actual attack on SAML in a specific scenario. However, we find this discussion important as SAML prescribes several security measures only as SHOULDs. Hence we believe that implementers, or inventors of additional profiles, should be aware of specific dangers when deviating from those security measures with the feeling of having a scenario that makes such a deviation safe.

Lemma 3.1 (Adversary Goals) *An adversary may compromise the authenticity of an honest user* U *who has the identity* idu *at an honest identity provider* I *towards a service provider* P *as defined in Definition 3.1 by any of the following means:*

(i) A *learns a SAML artifact* art *issued by* I *for* U *at* P *in Step 5 that neither* I *nor* P *have invalidated.*

(ii) A *learns a signed SAML assertion issued by* I *for* U *at* P*, where the assertion has not been used with* P *yet and its validity time has not expired in the view of* P*.*

(iii) A *acts as man-in-the-middle between browser* B *and service provider* P *in Step 6, i.e.,* A *is able to transparently forward the messages between* B *and* P*.*

□

We prove this lemma in Section A.1. We see that the adversary goals (i) and (ii) are information flow goals, whereas goal (iii) concerns the establishment of a secure channel between B and P. Note that the impossibility of an adversary to reach these sub-goals does not yet imply the authenticity of SAML according to Definition 3.1.

4 Analysis of Selected Security Measures

In this section, we analyze selected security measures added to SAML V2.0 [18, 21] and described in the SSTC response [14]. The SSTC response stresses the use of server-side authenticated secure channels by means of SSL3.0 and TLS1.0. Therefore, we take such channels for granted in most of this paper and focus on other security measures in this analysis section. However, note that the profiles still do not really prescribe secure channels in all places. In the appendix, Section A.4 we give an example for the POST profile where a pretty reasonable implementation that replaces secure channels by message-level security is vulnerable. The main recommendation following from this observation is that the SAML standards and security recommendations should clearly distinguish the requirement of secure channels from the requirement of mutual authentication, integrity, and confidentiality. The former is stronger. SSL and TLS are considered to provide secure channels, and thus following the SAML recommendations is usually safe, but individual implementations that follow the surrounding considerations to provide an alternative solution may be problematic.

We focus on the information flow of SAML artifacts. SAML has designed this part of the artifact binding carefully and taken several precautions against this problem. These precautions provide good protection against information flow attacks on the SAML artifact known in prior literature. However, this protection is (as in all browser/artifact profiles known) not perfect. Spotting a weakness in the well-elaborated harness of security measures is not trivial. Therefore, we first shed light on different aspects of the problem and corresponding security measures, in oder to allow developers of future SAML or other profiles to understand the advantages and disadvantages of the different security measures. Then we construct one attack exploiting different aspects of SAML to circumvent the security measures in a specific scenario. We do this for the purpose of demonstration that the protection against artifact information flow is still not perfect and as motivation that one needs to look at further measures to solve this problem once and for all.

4.1 One Time Request Property

An important security measure for the SAML artifacts is the so-called one-time request property of the SAML artifact: the identity provider I enforces that an artifact may only be used once to obtain an assertion. If the identity provider sees the artifact a second time it will behave as if it does not know the artifact. This property provides protection from replay attacks. However, [6] proposed to interrupt the channel between service provider P and identity provider I in order to prevent an artifact from being invalidated. A counter-measure proposed by [6] and adopted by SAML V2.0 is the provision of checks of the one-time request property by the service provider P. SAML V2.0 uses a variant of this proposal, in which the service provider P puts an artifact on a blacklist only if "an attempt to resolve an artifact does not complete successfully" [18, Section 3.6.5]. This measure defeats the so-called Referer attack of [6]. However, it does not cope with arbitrary information flow of valid artifacts to an adversary A. We consider a specific scenario that defeats this security measure in Section 4.3.

Recommendation. Step 8 of the SAML V2.0 Web Browser SSO/Response/Artifact Feature must have the postcondition that *all* artifacts that the identity provider sent out in Step 5 are invalidated either in the view of I or the view of P. One way to reach this goal is that the service provider P puts all artifacts seen on a blacklist; however, this is costly in terms of state space to hold at P. Alternatively, I can explicitly confirm the invalidation of SAML artifacts in its Step 8 response to P. Then P puts all artifacts seen and not confirmed to be invalidated on its own blacklist; consequently all artifacts are put on the blacklist if communication fails or the artifact resolution was unsuccessful.

4.2 Artifact Lifetimes

The SAML V2.0 Web Browser SSO/Response/Artifact Feature uses the short lifetime of the SAML artifacts as an additional security measure. The lifetime is specified as a few minutes, where identity providers and service providers should have clock skews of at most a few minutes [14, Section 1.3.4], [21, Section 6.5.1]. We believe (and so, we think, do the SAML designers) that timestamps as freshness measure are a suitable heuristic to prevent accidental disclosure of a still valid artifact, however, if a determined adversary tries to break a SAML protocol run of a specific user, several minutes are enough time to do so. Therefore, we see timing as a complement for other security measures, yet, timing does not guarantee security. Instead, we prefer to base security on active prevention of information flow of the artifact beyond the sphere of influence of the protocol.

Recommendation. Do not rely on artifact lifetime as a primary security argument of a browser/artifact profile. Render information flow of the artifact beyond the protocol run itself impossible.

4.3 Accumulating Artifacts

The SAML security analysis [6] noted the possibility of an adversary accumulating multiple artifacts in a Step 6 redirect to the service provider. How may this possibility affect the protocol's security? The SAML specification only prescribes service provider P to send the (one) artifact to the identity provider I [17, Section 3.5.1], [18, Section 3.6.5]; SAML does not contain provisions for handling URLs with multiple artifacts. Thus, accumulating multiple artifacts in a request may leave valid artifacts around. This leaves the adversary the option to get hold of those artifacts that were not invalidated. However, no concrete exploit based on this idea was contained in [6]; in particular it mentioned only the possibility that a malicious service provider can accumulate valid artifacts in the URL by executing Steps 3-6 repeatedly. However, these artifacts were issued for the malicious service provider P^* and will only lead to assertions accepted by P^*, and thus do not threaten authenticity by themselves.

Instead, we will now use these artifacts as a disguise for a valid artifact for an honest service provider. We devise a concrete exploit as follows. Let us assume a scenario where an adversary A wants to get access to an honest service provider P impersonating an honest user U. First A makes up some artifacts art_i with the parameters used by the identity provider I (or collects them by repeatedly executing Steps 3 to 5 with I). In addition, A contacts P in the role of a user in order to make A issue an AuthnRequest $AuthnReq$. Now A redirects the browser B of user U to the identity provider I; for this A must either intercept an unprotected Step 1 message or be contacted by U in the role of a service provider. In this redirect, it includes $AuthnReq$ from P, and all the accumulated artifacts art_i in the querystring.

I will issue an artifact art valid for P and redirect the browser B to P. The postcondition of this flow that defines the scenario to be one where the attack works is that the redirect target URL ACS is augmented by the accumulated artifacts art_i as well as art.

In Step 6, P now chooses one artifact from the URI, where SAML does not define which one. Moreover, the artifact format does not allow P to distinguish which artifact was indeed issued for it. Therefore, we can assume that there exists a combination of identity provider I and service provider P where the probability that I puts the artifact art issued for P at a different position than that one from which P takes the artifact is not negligible. Whenever this happens, i.e., P picks an artifact art_i, then P and I both invalidate art_i and not art. The valid artifact art can flow to A by means of, for instance, a browser Referer tag. Then A can impersonate U at P as shown in Lemma 3.1(i). For completeness, the scenario up to the leakage of art is shown in the appendix, Figure 3.

Discussion. Is such an exploit realistic? To answer this question we need to check on the one hand how browser and servers react upon having multiple parameters with the same name in a URL's query string. We checked by experiment that (i) both entities accept such input and that (ii) different servers have different heuristics which element to choose from the querystring (see Appendix, Section A.2).

On the other hand, the postcondition defined above is only fulfilled if the implementation of the identity provider's single sign-on service copies the parameters in the querystring of SSO in Step 3 into the redirect target ACS. In SAML, the exact format of those URLs is not specified, thus, an implementation doing so is behaving according to the specification. Still, the question arises whether any reasonable implementation might behave this way. Firstly, we note that identity federation systems are mostly not stand-alone solutions, but embedded in a larger access control and identity management environment. Therefore identity federation systems have a natural selection of solutions that are compliant with the access control environment. Secondly, we observe that there are access control systems that use querystring parameters internally. One example is the dynamic URL addressing of resources, which dispatches requests depending on values of querystring parameters. Another example is the enrichment of the URL querystring with a session id and user attribute parameters. An identity federation system not forwarding querystring parameters when redirecting a browser may hamper other functionality of its environment, which may lead architects of those systems to come to a design decision to copy querystring parameters where allowed.

Recommendation. A SAML deployment must be capable of handling all sorts of message formats, especially messages that contain multiple artifacts. We propose that either identity providers control that no artifact is already included in requests issued to them in Step 3, or service providers are extended by rules how to handle and invalidate multiple artifacts in Step 6.

4.4 Misdelivered Valid Artifact

Sending around an artifact by means of a protocol-unaware web browser introduces a major risk of uncontrolled information flows to the SAML V2.0 Web Browser SSO/Response/Artifact Feature. This risk is inherent to the class of browser-based protocols. Specifically for profiles where an artifact is transfered in the redirect URL, the Referer tag set by the browser potentially generates undesired information flow to communication partners. If such an artifact is misdelivered, the protocol security is compromised. We agree to the SSTC response [14, Section 1.3.2] that a misdelivered SAML artifact is still insufficient to obtain the assertion to which it corresponds. However, the adversary could compromise the protocol authenticity (Definition 3.1) without obtaining a SAML assertion. In particular, we have seen in Lemma 3.1 that the adversary can impersonate a user U by constructing a Step 6 message and sending the still valid artifact to the service provider P for which the artifact was issued.

It is crucial to prevent information flows of a valid SAML artifact through a standard web browser. Even more so, we would like to devise an option that renders an artifact misdelivered useless for an adversary by construction. We will discuss such a proposal in the following section.

5 The SAML Web Browser SSO/Response/Janus Artifact Feature

In this section, we construct a profile and binding of SAML, i.e., a feature in SAML terms, that can tolerate certain misdelivery of SAML artifacts. We call it the Janus profile according to the homonymous god of the Roman mythology, the gate-keeper, or, as full feature name, the SAML Web Browser SSO/Response/Janus Artifact Feature.

> *Janus* is the Roman god of gates and doors (ianua), beginnings and endings, and hence represented with a double-faced head, each looking in opposite directions.

Our profile is based on the idea to issue two artifacts instead of one, where each artifact produces a different information flow within a standard web browser. Following the Janus metaphor, we want the adversary to be able to observe one face of Janus, yet not both. Thus we include one artifact in the URI at P to which the browser is redirected by I as the standard SAML V2.0 Web Browser SSO/Response/Artifact Feature does. However, we include a second artifact in the last user authentication URI of I. The browser will potentially include this second artifact in the Referer tag of the Step 6 request to the service provider P. It is crucial to note that now two artifacts arrive at service provider P in Step 6, but that they have different information flow in subsequent steps.

5.1 Profile Description

As SAML does with its profiles and bindings, we specify the Janus profile by means of constraints. Actually, regarding real constraints, Janus is a sub-feature of the SAML V2.0 Web Browser SSO/Response/Artifact Feature. Therefore Janus inherits the constraints of its parent feature and extends them by using a so-called Janus artifact with additional constraints.

The profile relies on one assumption about the consistency of browser behavior:

Definition 5.1 (Consistent Referer Tag Behavior) *A browser* B *shows consistent Referer tag behavior if (at least throughout one SAML protocol run and immediately following steps) it either sets Referer tags in the communication with all servers or with none if the preconditions for Referer tags from HTTP are fulfilled. For such a browser, let the predicate* SetsReferer *be* TRUE *if the browser does set Referer tags.* ◇

We define a Janus artifact as follows:

Definition 5.2 (Janus Artifact) *A* Janus artifact *art is a SAML artifact which is hidden by secret sharing in two artifact shares art_1 and art_2. With respect to a browser* B*, the Janus artifact and its two shares have the following property:*

```
**if** SetsReferer(B) **then** {
    $art.handle = art_1.handle \oplus art_2.handle$,
    where $art_1.handle, art_2.handle \in_{\mathcal{R}} \{0,1\}^l$;
} **else** $art.handle \in_{\mathcal{R}} \{0,1\}^l$;
```

where $\in_{\mathcal{R}}$ denotes uniformly random or pseudorandom and independent choice of a value from a domain, and $l = 160$. (In general l could be a security parameter.) ◇

In the case of real random choice of the artifact handles, the shares art_1 and art_2 of the Janus artifact art are information-theoretically independent of art. Therefore, if an adversary A only obtains one share, then A cannot reconstruct the Janus artifact art, and does not even obtain any information about it that would increase A's

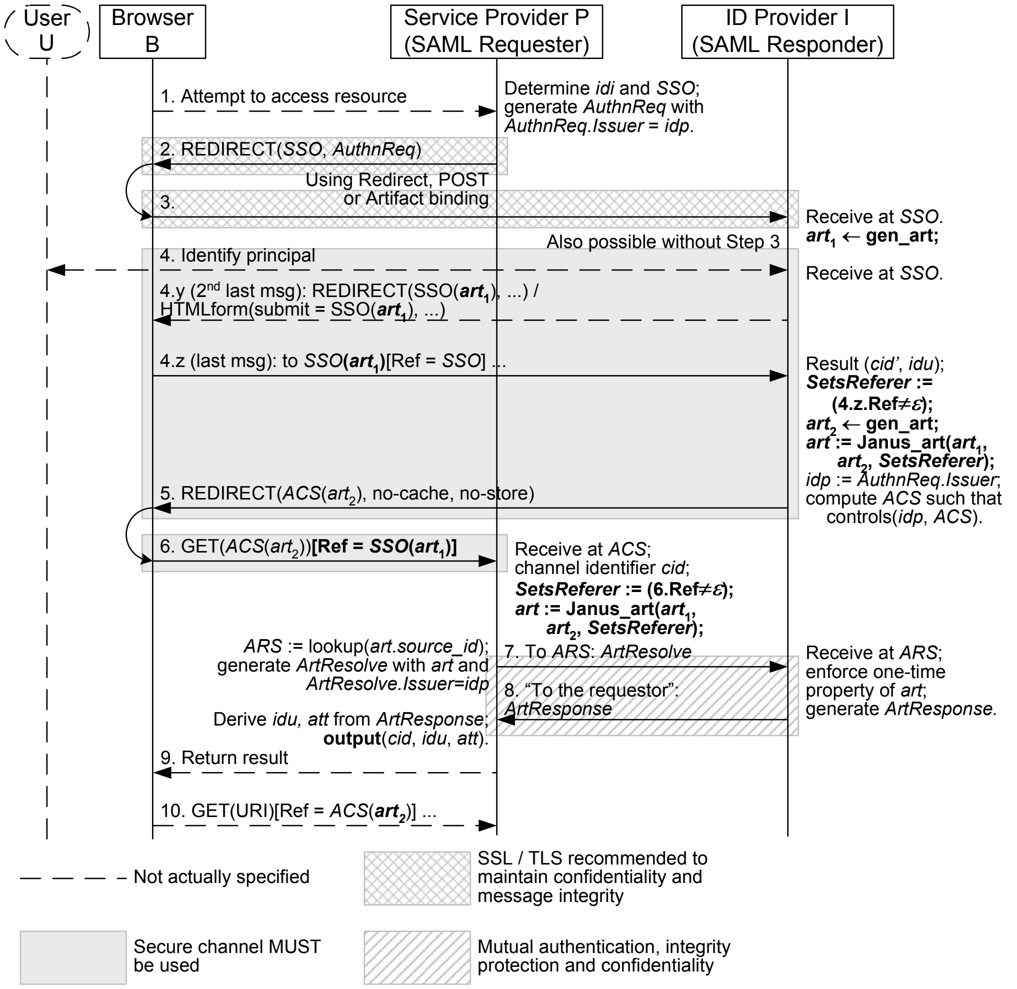
User U
Browser B
Service Provider P (SAML Requester)
ID Provider I (SAML Responder)
1. Attempt to access resource
Determine idi and SSO; generate AuthnReq with AuthnReq.Issuer = idp.
2. REDIRECT(SSO, AuthnReq)
Using Redirect, POST or Artifact binding
3.
Receive at SSO. art₁ ← gen_art;
Also possible without Step 3
4. Identify principal
Receive at SSO.
4.y (2nd last msg): REDIRECT(SSO(art₁), ...) / HTMLform(submit = SSO(art₁), ...)
4.z (last msg): to SSO(art₁)[Ref = SSO] ...
Result (cid', idu); SetsReferer := (4.z.Ref≠ε); art₂ ← gen_art; art' := Janus_art(art₁, art₂, SetsReferer); idp := AuthnReq.Issuer; compute ACS such that controls(idp, ACS).
5. REDIRECT(ACS(art₂), no-cache, no-store)
6. GET(ACS(art₂))[Ref = SSO(art₁)]
Receive at ACS; channel identifier cid; SetsReferer := (6.Ref≠ε); art := Janus_art(art₁, art₂, SetsReferer);
ARS := lookup(art.source_id); generate ArtResolve with art and ArtResolve.Issuer=idp
7. To ARS: ArtResolve
Receive at ARS; enforce one-time property of art; generate ArtResponse.
8. "To the requestor": ArtResponse
Derive idu, att from ArtResponse; output(cid, idu, att).
9. Return result
10. GET(URI)[Ref = ACS(art₂)] ...
Not actually specified
SSL / TLS recommended to maintain confidentiality and message integrity
Secure channel MUST be used
Mutual authentication, integrity protection and confidentiality

Figure 2. Janus profile, or SAML Web Browser SSO/Response/Janus Artifact Feature. Special aspects are shown in bold face.

advantage in guessing art. For pseudorandom choice, by the definition of cryptographic pseudo-randomness, the additional advantage of a computationally bounded adversary is negligible. The Janus artifact shares are transmitted in different ways to ensure that an adversary can obtain at most one share.

Notation and Functions. As in Section 2, I and P have unique entity identifiers idi and idp, respectively. The identity provider I controls two URLs: it performs user authentication and issues SAML artifacts at SSO and has its artifact resolution service at ARS. The service provider P controls the assertion consumer service URI ACS. By SAMLart we denote the domain of SAML artifacts as described in Section 2. We define the functions handling artifacts in more detail in the Appendix in Section A.5. Definition A.1 defines the generation of SAML artifacts according to the SAML specification [22]. Two Janus artifacts are combined by combining both pseudo-random handles by means of the XOR function. We describe the exact process of Janus artifact combination in Definition A.2 of Section A.5.

Step by Step. An overview of the Janus profile is shown in Figure 2. The general flow is as in the SAML V2.0 Web Browser SSO/Response/Artifact Feature surveyed in Section 2. We start the step-by-step description with the principal identification at I. SAML defines Step 4 as out-of-scope; in Janus we require that it ends in Step 4.z at the identity provider's address SSO and that Step 4 is done through a server-authenticated secure channel. The querystring in Step 4.z is augmented by a SAML artifact art_1 generated by the function gen_art from Definition A.1; this is the first share of a Janus artifact we construct later. How this is done depends on the principal identification solution used by I. If the principal identification is POST-based, the HTML form querying the user for authentication may hold $SSO(art_1)$ as the submission address. Solutions based upon a GET request may issue an explicit redirect to $SSO(art_1)$ in the preceding Step 4.y. However, if the overall solution takes more than one request-response pair, the identity provider may find a way to execute Step 4.y at URI $SSO(art_1)$ without any additional round-trip.[4] Given a Step 4.z request, I derives a user identity $idu \neq \epsilon$ corresponding to the principal identification; this is bound to the channel identifier cid'. Furthermore, I tests whether the browser B is setting Referer tags by checking whether the Step 4.z request contains such a tag. Potential Referer tags are indicated in the figure by elements Ref in brackets. Here we assume that the authentication method used gets the address SSO of Step 4.z from a source with its own URI, so that the precondition for setting the Referer Tag is fulfilled. Now I generates a second SAML artifact art_2 with $handle$ independent from art_1 using gen_art, and employs function Janus_art from Definition A.2 to compute a Janus artifact art from the shares art_1 and art_2. Next, I derives the service provider's entity identifier idp from $AuthnRequest.Issuer$ and computes ACS such that controls (idp, ACS) holds. Finally, it redirects the browser to $ACS(art_2)$ by sending a REDIRECT response to the channel with identifier cid'.

Upon the browser's GET request at the end of the redirect (Step 6), service provider P checks whether the browser has set a Referer tag and whether this Referer tag contains a SAML artifact art_1. If so, P assumes that I issues a Janus artifact consisting of two shares art_1 and art_2. P computes the Janus artifact art from these shares using the function Janus_art. Otherwise, this function returns art_2 as the SAML artifact art. Now P looks up the identity provider's artifact resolution service URI ARS by means of the artifact's $source\_id$. In Step 7, P sends the artifact art in an $ArtResolve$ message to ARS to resolve the SAML artifact. The identity provider looks up the SAML artifact and enforces the one-time request property. If the artifact can be resolved to an assertion, then I sends this assertion enclosed in an $ArtResponse$ message to service provider P in Step 8. Recall that Janus inherits the security checks and constraints prescribed by the SAML V2.0 Web Browser SSO/Response/Artifact Feature for these steps, in particular mutual authentication and confidentiality for Steps 7 and 8.

5.2 Security Consideration

The core security property achieved by Janus artifacts is captured by the following lemma.

Lemma 5.1 (Information flow of Janus artifact) *Let the trust and setup preconditions of authenticity (Definition 3.1) be true, and let the browser have consistent Referer tag behavior (Definition 5.1). Then the Janus profile defined in Section 5.1 produces no information flow from the Janus artifact art to other parties than* P, I *and* B. □

Proof (sketch). We have two cases depending on the Referer tag behavior of the browser.

Case 1: B *sets Referer tags.* If the identity provider I observes in Step 4.z that browser B sets Referer tags, it generates art as a real Janus artifact from shares art_1 and art_2. Here art_1 is issued in the redirect location and Step 4.y and used in the URI in Step 4.z. Thus the browser B puts in the Referer tag of the subsequent HTTP request, which is Step 6. The usages in Step 4 are over a secure channel and thus unobservable except for I and B, and Step 6 is over a secure channel to an address controlled by P so that only P learns art_1 here. The second artifact share art_2 is issued in the redirect location of Step 5 and used in the URI in Step 6, and potentially as Referer tag in Step 10. While the first two uses are again protected, the last use may be unprotected. P sees both artifact shares in the GET request of Step 6, one in the Referer tag and one in ACS itself, and can therefore compute the same resulting artifact art as I. P uses the artifact art in Step 7, but this step provides confidentiality, so that there is no information flow of art here except back to I. There is no further use of art

[4] In implementations of I where the principal identification in Step 4 is based upon a GET request and only takes one request-response pair, or where identification is retained from a previous protocol run, I needs to issue one additional redirect message to direct the browser to $SSO(art_1)$ and therefore loses the round-trip-time advantage of Janus. However, a typical case is an initial internal redirect to an authentication service.

or its shares in the profile. Hence an adversary learns at most art_2, and the Janus artifact art is information-theoretically or computationally independent of one share alone, i.e., an adversary has no advantage in guessing a valid artifact art.

Case 2: B *does not set Referer tags.* In this case, the protocol flow is identical to the normal SAML V2.0 Web Browser SSO/Response/Artifact Feature; however, information flow by means of the Referer tag is now prevented by the precondition of the case. ■

5.3 Discussion

How does the Janus profile behave compared to other measures that address the artifact information flow problem? We first discuss two prominent alternatives. One is that the service provider employs a cleaning self-redirect after Step 8 to make the browser strip off a potentially valid SAML artifact in the Referer tag. Such solutions are widely used by e-mail providers preventing a user's session identifier to be disclosed when redirecting the user's browser to other servers. However, these solutions have the disadvantage of additional round-trip times. Another proposal is to enforce a full one-time request property at the service provider P. Instead of storing only a blacklist of artifacts where the resolution failed, the service provider stores *all* artifacts seen. In Section 4.1 we proposed a light-weight alternative to store all artifacts that were not confirmed by the identity provider I to be invalidated. Still, both solutions burden service providers with storing a potentially high number of artifacts. Additionally, an adversary may attack this solution by sending large numbers of artifacts in Step 6 messages to the service provider and have the service provider exhaust its storage bounds for the artifacts. Moreover, any cleanup measures on the artifacts that are based on expiration times would again be based on timing assumptions.

With the SAML Web Browser SSO/Response/Janus Artifact Feature, we pursue a solution that does not have these drawbacks. This new solution does not need an additional self-redirect in most cases, nor does it rely on P storing lots of artifacts or assumptions about the artifact's lifetime. Actually, the artifact in the Step 6 ACS URL may indeed flow to the adversary. The point of the Janus profile is that this artifact is completely worthless for an adversary. We recall that the profile relies on the assumption that a standard web browser behaves consistently in communication with other servers: either the browser sends Referer tags to all servers, or does not send them to anyone. A browser that does not send Referer tags to P, yet, does send Referer tags to another server reachable from Step 9 of the profile, may break the profile. If one does not trust this assumption, we recommend to complement the Janus profile with the light-weight blacklist measure with additional cleanup after times significantly beyond the artifacts' lifetimes, and thus using only a very weak timing assumption.

6 Conclusion

We have analyzed the SAML V2.0 Web Browser SSO/Response/Artifact Feature and focused on the problem of information flow of the SAML artifact, which is inherent to all browser/artifact profiles. For a specific scenario, we have devised a concrete exploit that circumvents the current security measures in SAML and demonstrates such an information flow. With the Janus profile, or SAML Web Browser SSO/Response/Janus Artifact Feature, we have devised a novel efficient solution to this problem. This solution neither relies on timing assumptions about the artifact's lifetime, nor does it need additional messages in most cases, nor does the service provider have to hold state in the form of blacklists or similar space-consuming measures. Only leveraging the information-theoretical or computational independence of two artifact shares and an assumption about the consistency of a standard browser's behavior, it presents a new approach to the problem of artifact information flow.

References

[1] Michael Backes, Sebastian Mödersheim, Birgit Pfitzmann, and Luca Viganò. Symbolic and cryptographic analysis of the Secure WS-ReliableMessaging scenario. Technical Report IBM Research Report RZ 3619, IBM Research Division, August 2005.

[2] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. A semantics for web services authentication. In *31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 198–209. ACM Press, 2004.

[3] Scott Cantor and Marlena Erdos. Shibboleth-architecture draft v05, May 2002. `http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v0%5.pdf`.

[4] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. Dos and don'ts of client authentication on the web. In *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., August 2001. USENIX. An extended version is available as MIT-LCS-TR-818.

[5] Andrew D. Gordon and Riccardo Pucella. Validating a web service security abstraction by typing. In *Proc. 2002 ACM Workshop on XML Security*, pages 18–29, Fairfax VA, USA, November 2002.

[6] Thomas Groß. Security analysis of the SAML Single Sign-on Browser/Artifact profile. In *Proc. 19th Annual Computer Security Applications Conference*. IEEE, December 2003.

[7] Thomas Groß and Birgit Pfitzmann. Proving a WS-Federation Passive Requestor profile. In *2004 ACM Workshop on Secure Web Services (SWS)*, Washington, DC, USA, October 2004. ACM Press.

[8] Thomas Groß, Birgit Pfitzmann, and Ahmad-Reza Sadeghi. Browser model for security analysis of browser-based protocols. In *ESORICS: 10th European Symposium on Research in Computer Security*, volume 3679 of *Lecture Notes in Computer Science*, pages 489–508. Springer-Verlag, Berlin Germany, 2005. To appear; preliminary version IBM Research Report RZ 3600, April 2005.

[9] Thomas Groß, Birgit Pfitzmann, and Ahmad-Reza Sadeghi. Proving a WS-Federation Passive Requestor profile with a browser model. Technical Report IBM Research Report RZ 3623, IBM Research Division, July 2005.

[10] Chris Kaler and Anthony Nadalin (ed.). Web Services Federation Language (WS-Federation), Version 1.0, July 2003. BEA and IBM and Microsoft and RSA Security and VeriSign, `http://www-106.ibm.com/developerworks/webservices/library/ws-fed/`.

[11] Chris Kaler and Anthony Nadalin (ed.). WS-Federation: Passive Requestor Profile, Version 1.0, July 2003. BEA and IBM and Microsoft and RSA Security and VeriSign, `http://www-106.ibm.com/developerworks/library/ws-fedpass/`.

[12] David P. Kormann and Aviel D. Rubin. Risks of the Passport single signon protocol. *Computer Networks*, 33(1–6):51–58, June 2000.

[13] Liberty Alliance Project. Liberty Phase 2 final specifications, November 2003. `http://www.projectliberty.org/`.

[14] John Linn and Prateek Mishra. SSTC response to "security analysis of the SAML Single Sign-on Browser/Artifact", working draft 01, January 2005. `http://www.oasis-open.org/committees/documents.php?wg_abbrev=security`.

[15] Microsoft Corporation. .NET Passport documentation, in particular Technical Overview, and SDK 2.1 Documentation (started 1999), September 2001.

[16] OASIS Standard. Security assertion markup language (SAML) V1.1, November 2002.

[17] OASIS Standard. Assertions and protocols for the oasis security assertion markup language (SAML) V2.0, March 2005.

[18] OASIS Standard. Bindings for the oasis security assertion markup language (SAML) V2.0, March 2005.

[19] OASIS Standard. Metadata for the oasis security assertion markup language (SAML) V2.0, March 2005.

[20] OASIS Standard. Profiles for the oasis security assertion markup language (SAML) V2.0, March 2005.

[21] OASIS Standard. Security and privacy considerations for the oasis security assertion markup language (SAML) V2.0, March 2005.

[22] OASIS Standard. Security assertion markup language (SAML) V2.0, March 2005.

[23] Birgit Pfitzmann and Michael Waidner. Analysis of Liberty single-signon with enabled clients. *IEEE Internet Computing*, 7(6):38–44, 2003.

A Appendix

A.1 Proof of Relation of Adversary Goals

We prove Lemma 3.1 as follows:

Proof. We show that it is sufficient for an adversary A to reach one of the sub-goals of Lemma 3.1 to compromise authenticity.

(i) If an adversary A learns a still valid artifact art, then A can construct a Step 6 message to P including art. (A can run Step 1 with P in advance if P does not accept unsolicited authentications, and there is no message-level protection in Step 6.) Service provider P will query I for the assertion corresponding to art and then consider A authenticated under the identity idu contained in that assertion. Thus, A may impersonate U.[5]

(ii) If A learns a valid signed SAML assertion from I containing the identity idu and meant for P, then it can use this assertion in a protocol run with POST binding. Essentially, it simply uses the assertion in an AuthnResponse message of the Web SSO profile with POST binding, where this message is sent via the browser, so that the adversary can impersonate the browser of user U. For a more detailed reference, we show the SAML V2.0 Web Browser SSO/Response/Artifact Feature in the appendix, Figure 4. Note that assertions only refer to P via idp which is independent of the binding,[6] that there is no required (not even recommended) protection on the Response Res including the assertion, and that the recommended use of SSL for Step 6 does not hamper the attack because there is no client authentication. Also note that it does not help here if the assertion must match a valid AuthnRequest, as P issued such an AuthnRequest for the protocol run with artifact binding and the AuthnRequest is not bound to a specific SAML protocol run or profile. Thus, the service provider P will accept this assertion, and A may therefore impersonate U.

(iii) If an adversary A manages to become man-in-the-middle in Step 6 of the protocol run, then the channel with identifier cid that the service provider P associates with idu as the result of the SAML protocol run actually belongs to A. ■

A.2 On Multiple Querystring Parts

As an example that there are sites that accept multiple querystring elements with the same name, and that different sites interpret these querystrings differently, one can check out the ACM and IEEE digital libraries:

For ACM, a given URL is `http://portal.acm.org/browse_dl.cfm?linked=1&part=transaction&coll=portal`. We now add another element named "part". This URL gives the same page: `http://portal.acm.org/browse_dl.cfm?linked=1&part=transaction2&part=transaction&coll=portal&dl=ACM`. In contrast, the following one looks up non-existing

[5] SAML V2.0 contains the possibility to include the client IP address in the assertion. This security measure restricts the adversary's elbowroom, however, it is optional and may be circumvented by IP spoofing.

[6] Especially [19, Section 2.3.2] structures the $<$ EntityDescriptor $>$ elements such that a single entity can act in multiple roles under the same entity identifier idp. Thus, it is reasonable to assume that there are service providers using the same idp for POST and Artifact bindings.

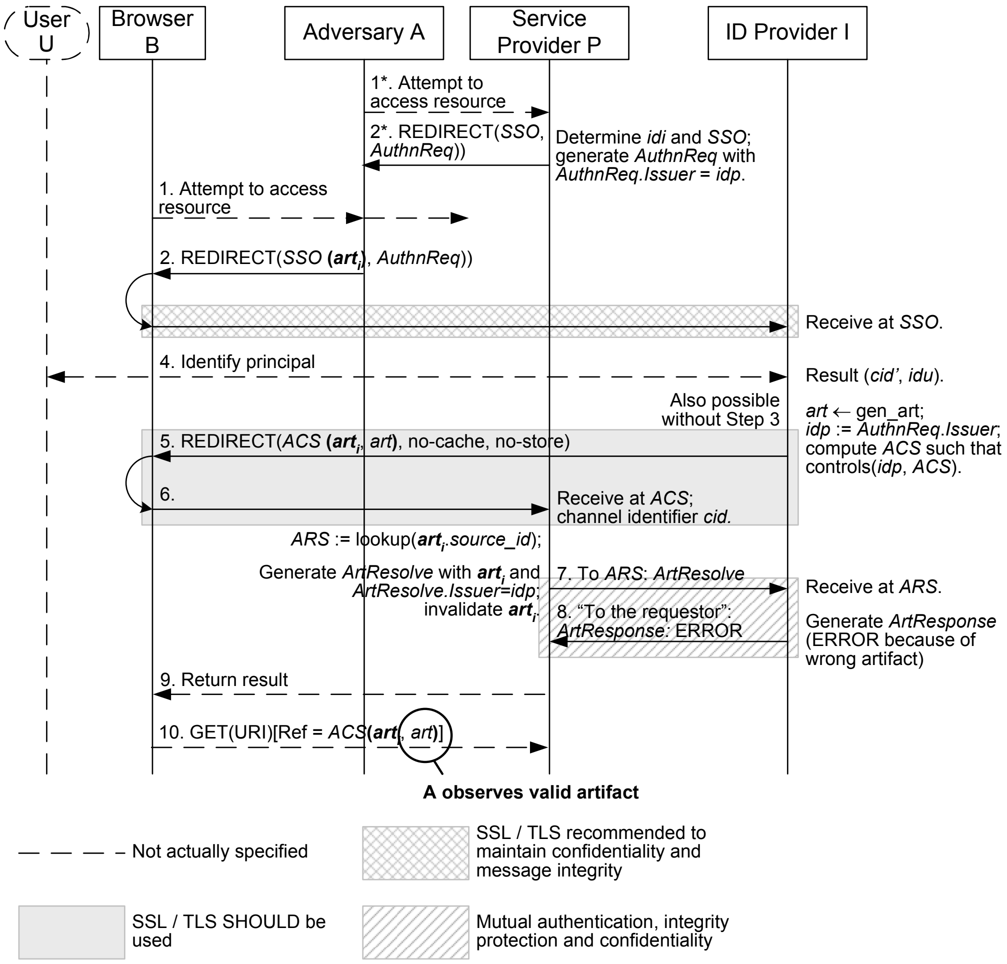
User U
Browser B
Adversary A
Service Provider P
ID Provider I
1*. Attempt to access resource
2*. REDIRECT(*SSO*, *AuthnReq*))
Determine *idi* and *SSO*; generate *AuthnReq* with *AuthnReq.Issuer* = *idp*.
1. Attempt to access resource
2. REDIRECT(*SSO* (**art$_i$**), *AuthnReq*))
Receive at *SSO*.
4. Identify principal
Result (*cid'*, *idu*).
Also possible without Step 3
art ← gen_art; *idp* := *AuthnReq.Issuer*; compute *ACS* such that controls(*idp*, *ACS*).
5. REDIRECT(*ACS* (**art$_i$**, *art*), no-cache, no-store)
6.
Receive at *ACS*; channel identifier *cid*.
ARS := lookup(**art$_i$**.*source_id*);
Generate *ArtResolve* with **art$_i$** and *ArtResolve.Issuer=idp*; invalidate **art$_i$**.
7. To *ARS*: *ArtResolve*
Receive at *ARS*.
8. "To the requestor": *ArtResponse:* ERROR
Generate *ArtResponse* (ERROR because of wrong artifact)
9. Return result
10. GET(URI)[Ref = *ACS*(**art$_i$**, *art*)]
A observes valid artifact
Not actually specified
SSL / TLS recommended to maintain confidentiality and message integrity
SSL / TLS SHOULD be used
Mutual authentication, integrity protection and confidentiality

Figure 3. Artifact accumulation attack up to artifact leakage

transactions "transaction2": http://portal.acm.org/browse_dl.cfm?linked=1&part=transaction&part=transaction2&coll=portal&dl=ACM. The conclusion seems to be that the last version counts.

In contrast, for the correct URL http://www.computer.org/portal/site/transactions/index.jsp?&pName=transactions_level1&path=transactions/tc/mc&file=author.xml&xsl=article.xsl& adding an element file=author2.xml after the original one keeps the page, while adding it before the original gives an error. So here the first version seems to count.

A.3 Details of Accumulating Artifacts

Figure 3 shows the accumulation of artifacts art_i and the subsequent leakage of a target artifact art that neither the identity provider I nor the honest service provider P has invalidated. An overview of this attack was given in Section 4.3. Text in bold face denotes the specific elements changed in the attack. The main assumption defining this scenario is that art_i will indeed survive in the querystring from Step 3 to Step 5, as discussed in Section 4.3. Let us briefly consider the other tests made by P and I: As we let P construct $AuthnReq$, the attack works even if I requires signed AuthnRequests. Furthermore, the final $Response$ message when the adversary exploits the valid artifact at P will correspond to this $AuthnReq$. The possibility to include the client IP address in the assertion somewhat restricts this attack like all impersonation attacks based on stealing artifacts or assertions, but on the one hand it is optional and on the other hand the adversary may use IP spoofing.

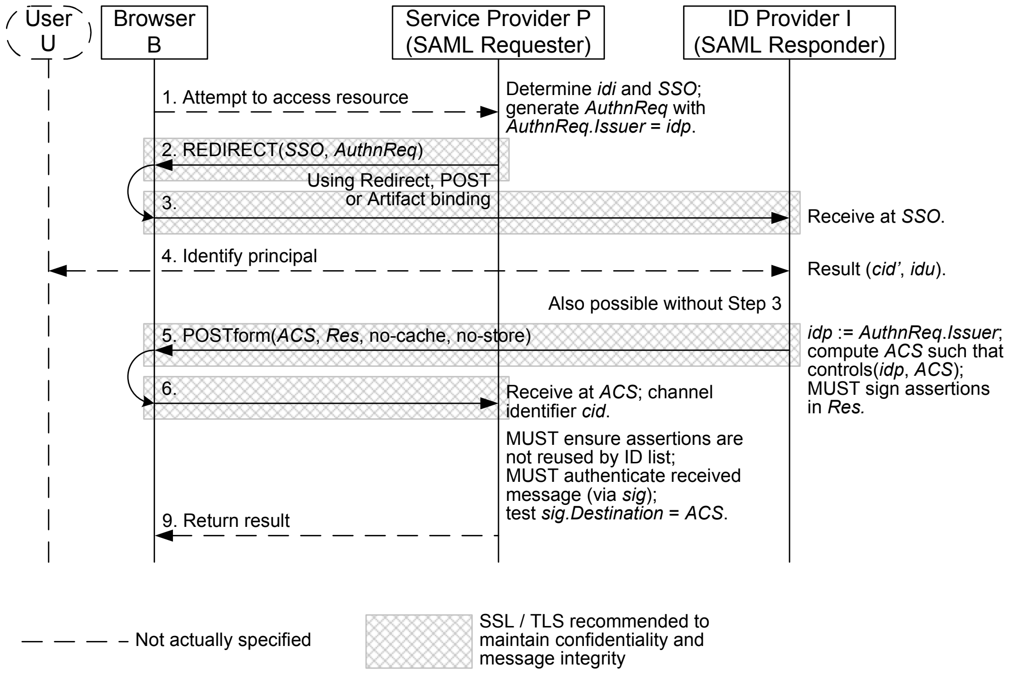
User U
Browser B
Service Provider P (SAML Requester)
ID Provider I (SAML Responder)
1. Attempt to access resource
Determine idi and SSO; generate AuthnReq with AuthnReq.Issuer = idp.
2. REDIRECT(SSO, AuthnReq)
Using Redirect, POST or Artifact binding
3.
Receive at SSO.
4. Identify principal
Result (cid', idu).
Also possible without Step 3
5. POSTform(ACS, Res, no-cache, no-store)
idp := AuthnReq.Issuer; compute ACS such that controls(idp, ACS); MUST sign assertions in Res.
6.
Receive at ACS; channel identifier cid.
MUST ensure assertions are not reused by ID list; MUST authenticate received message (via sig); test sig.Destination = ACS.
9. Return result
Not actually specified
SSL / TLS recommended to maintain confidentiality and message integrity

Figure 4. SAML V2.0 Web Browser SSO/Response/POST Feature

A.4 POST Binding and an Attack Possibility

Figure 4 summarizes the SAML V2.0 Web Browser SSO/Response/POST Feature. As before, we show the simple redirect binding for the AuthnRequest, and we only show the most important parameters and constraints. We argue that an implementer may feel to have a compliant implementation and nevertheless allow a successful attack. The problem lies in the recommendations about the authentication of Steps 5 and 6. In the overall profile [20, Section 4.1.3.5], this is formulated as follows: "It is RECOMMENDED that the HTTP requests in this step be made over either SSL 3.0 or TLS 1.0 to maintain confidentiality and message integrity." The formulation in the POST binding [18, Section 3.5.5.2] is: "The presence of the user agent intermediary means that the requester and responder cannot rely on the transport layer for end-end authentication, integrity or confidentiality protection and must authenticate the messages received instead. SAML provides for a signature on protocol messages for authentication and integrity for such cases." It proceeds to discuss confidentiality with statements like "If confidentiality is necessary, SSL 3.0 or TLS 1.0 SHOULD be used to protect the message in transit between the user agent and the SAML requester and responder."

We believe that an implementer with reasonable knowledge of security may take these statements together and believe that he provides sufficient integrity if he signs the Response message Res, and potentially encrypts it additionally: He might believe that this is sufficient for the intention expressed by the part "to maintain ..." in the profile recommendation, and this belief is supported by the more specific binding considerations. We do not argue whether this really *is* SAML compliant, only that it is a real danger for implementations. Such an implementer might be motivated a significant efficiency gain over the use of SSL/TLS if I and P share symmetric keys because they interact often, while they otherwise need SSL setup phases with each user that I identifies for P. SAML allows XML signatures and XML encryption, and does not exclude symmetric implementations.

The danger in this case is that a man-in-the-middle attack as in Lemma 3.1 (ii) becomes possible. To summarize, an adversary would pick up the signed and encrypted response Res in Step 5 or Step 6 before it reaches the service provider P. (Step 6 is more likely because Step 5 may still be protected as an HTTP response to part of the user identification of Step 4.) The adversary then simply uses this assertion from its own browser with the same service provider P. All the cryptographic protection applies to P and I and is therefore still valid. Furthermore, this response and thus the contained assertions are used for the first time and thus not invalidated at P. The only measure that somewhat restricts this possibility is the inclusion of the client IP address in the assertion, but on the one hand it is optional and on the other hand the adversary may use IP spoofing.

This attack points out one more time the importance of the distinction between secure channels, as SSL and TLS strive to provide, and the combination of authentication, integrity, and confidentiality: The latter are not a replacement for the former.

A.5 Details of Janus Artifact Generation

We describe the generation of artifacts and combination of Janus artifacts in full details:

Definition A.1 (Artifact Generation) *The artifact generation function* $\mathsf{gen\_art} : \mathbb{N} \times \mathsf{URI} \times \mathbb{N} \longrightarrow \mathsf{SAMLart}$ *is defined as follows:*

```
$\mathsf{gen\_art}(l, uri, ep) := \{$
    $handle \in_{\mathcal{R}} \{0,1\}^l;$
    $tc := \mathsf{0x0004};$
    $source\_id \leftarrow \mathsf{h_{SHA1}}(uri);$
    $\textbf{return}\ (tc, ep, source\_id, handle); \},$
```

where $\mathsf{h_{SHA1}}$ *denotes SHA-1 hashing.* ◇

Definition A.2 (Janus Artifact Combination) *The artifact combination function* $\mathsf{Janus\_art} : \mathsf{SAMLart} \times \mathsf{SAMLart} \times \mathsf{Bool} \longrightarrow \mathsf{SAMLart} \cup \{\epsilon\}$ *is defined as follows:*

```
$\mathsf{Janus\_art}(art_1, art_2, SetsReferer) := \{$
    $\textbf{if}\ (\neg SetsReferer)\ \textbf{then return}\ art_2;$
    $(tc_1, ep_1, source\_id_1, handle_1) \leftarrow art_1;$
    $(tc_2, ep_2, source\_id_2, handle_2) \leftarrow art_2;$
    $\textbf{if}\ ((tc_1 \neq \mathsf{0x0004}) \vee (tc_2 \neq \mathsf{0x0004}) \vee (ep_1 \neq ep_2) \vee (source\_id_1 \neq source\_id_2))\ \textbf{then}\ \{$
        $\textbf{return}\ \epsilon;$
    $\}\ \textbf{else}\ \{handle := handle_1 \oplus handle_2;$
        $art := (tc_1, ep_1, source\_id_1, handle);$
        $\textbf{return}\ art; \},$
```

where an assignment "←" from a value to a tuple denotes (unique) tuple decomposition; the results are ϵ *if the desired decomposition fails.* ◇