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) [17]. 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. This means that do not require the installation of additional client software and are, therefore, cost-efficient to deploy. However, designing secure protocols with a standard web browser as the client is not trivial. One of the major challenges of this protocol class is that the browser is not aware of the protocol it participates in. The browser has a predefined behavior, reacts to predefined messages and generates information flow both to the underlying operating system and to communication partners. Each of these properties may put a protocol's security at stake. In particular, protocols that transfer confidential information through a browser's URL are endangered by this protocol-unaware behavior of a standard web browser. In this paper, we analyze the browser/artifact profile of SAML, a prominent example of this protocol class. It leverages the browser redirect URL to transfer a random reference to a user's credential, the so called artifact, to a service provider.

The SAML V1.1 Web SSO Browser/Artifact Profile [17] was already analyzed by [7] and suffered from some problems introduced by a standard web browser as client. In the meantime, SAML has advanced to Version 2.0 [18] and revised also this profile, repairing most of the problems discussed in [7] and discussing the improvements in an SSTC response [15]. 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 [16] was the first protocol to be analyzed. Its detailed analysis by [13] also discussed inherent problems of browser-based protocols. Inherent problems of browser-based client authentication were also discussed in [5] independent of the federation aspect. Liberty [14] is an identity federation protocol that is based upon SAML V1.1, however, has also influenced the development of SAML V2.0. Weaknesses in the original version of the Liberty enabled-client profile were found by [19] and repaired in subsequent Liberty versions. Shibboleth [4] 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 [12] was analyzed first by [8] 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 [9] that is to be used for more in-depth proofs such as [10]. 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 starting with [6, 3, 2] and the original SAML V1.1 [11]. The inherent problems of modelling Web Services and identity federation protocols with Dolev Yao-alike abstractions were pointed out in [1]. However, protocols involving a standard web browser were not considered by the tool-support approach.

Our Contribution. We analyze the security measures newly introduced by version 2.0 into the browser/artifact profile of SAML [18, 15]. We discuss the generic security goals for the profile and their impact on 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, [7] 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 attacks by introducing a one-time request constraint at the service provider, it can still be compromised by information flow of the SAML artifact. 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.

Outline. We begin our discussion of the SAML V2.0 Web Browser SSO/Response/Artifact Feature with a protocol overview in Section 2. This section introduces our notation for the elements of the SAML message standard as well as for the corresponding protocol meta-data. We complement this section with an tabular overview over SAML as multipart standard in Appendix Section A.1. In Section 3, we define authenticity as our main security goal for identity federation protocols and SAML in particular. Given this basis, we analyze concrete security properties of SAML V2.0 Web Browser SSO/Response/Artifact Feature in Section 4, where we focus of the artifact information flow as main source of vunerabilities. This analysis motivates the introduction of a new SAML profile called Janus in Section 5, which leverages the abstract idea of using two artifacts that produce a different information flow in a standard web browser and is complemented by a security and an efficiency analysis.

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 for which we denote the most important parts for our analysis in Table 2 of Appendix Section A.1.

According to the SAML conformance specification, the combination of profile, message exchange and a selected binding is termed a SAML V2.0 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 [18, Metadata, Section 2.2.1] by idi for identity providers and idp for service providers. Likewise, we denote unique user identities as registered with identity providers by idu. 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 $\mathsf{I}.ACS$. 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. We use the notation $U(Params)$ to denote a URI U augmented by a list of parameters $Params$ encoded in its querystring. The predicate $\mathsf{controls}(id, URI)$ states 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 and that one can evaluate whether an address belongs to a participant.

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, the service provider P requests a single sign-on authentication from the identity provider I. This is done by means of a SAML AuthnReq message that P includes in a redirect response. The service provider sends $AuthnReq$ via the browser to a single sign-on service address SSO of a desired identity provider with identity idi. Its most important parameters are $Issuer$, the request issuer, and a request identifier ID. The only constraint is that the service provider has to set the issuer parameter to its own identity idp.

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

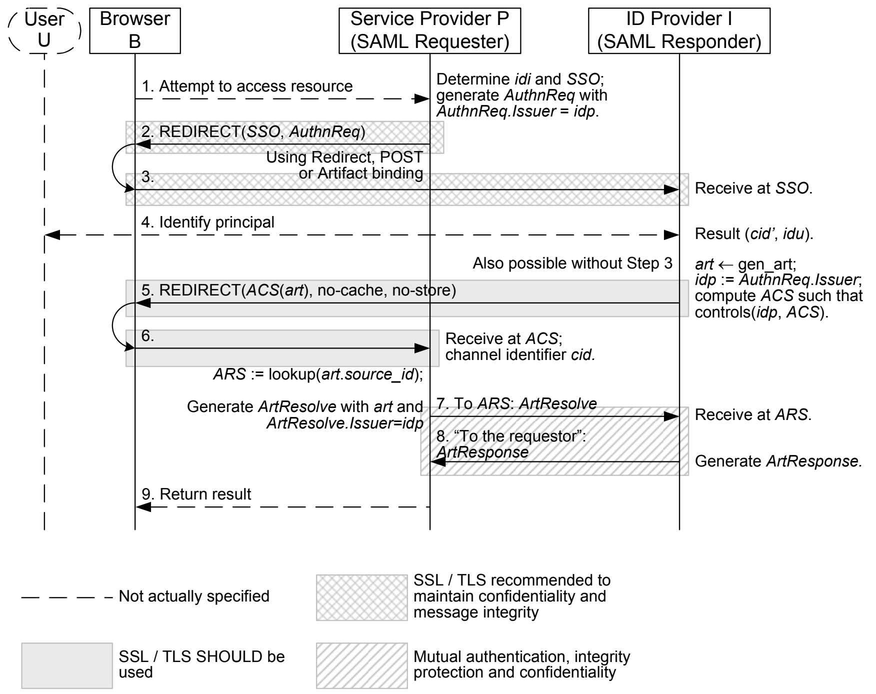
User U
Browser B
Service Provider P (SAML Requester)
ID Provider I (SAML Responder)
1. Attempt to access resource
Determine *idi* and *SSO*; generate *AuthnReq* with *AuthnReq.Issuer* = *idp*.
2. REDIRECT(*SSO*, *AuthnReq*)
Using Redirect, POST or Artifact binding
3.
Receive at *SSO*.
4. Identify principal
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

Then, for Step 5, the identity provider generates an artifact art and redirects the browser back to the service provider. In our case, the artifact is placed in a query string parameter named $SAMLart$, other profile variants place it 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. [18, Profiles, 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 Step 7, the service provider P opens a secure back-channel to the identity provider and asks I 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 does not directly specify 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.

SAML poses several requirements on the Response Res element within $ArtResponse$ in Step 8. It is required that the issuer of the response and all assertions included match the identity provider, that is $Res.Issuer \in \{idi, \epsilon\}$ 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, and that has subject confirmation method bearer and the assertion consumer URL as recipient, $assert^*.recipient = ACS$. Furthermore, if there was Step 3, then the Step 8 message must match the identifier of the Step 3 message: $Res.InResponseTo = AuthnReq.ID$.[1]

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

3 Security Goal: Authenticity

In this section, we define authenticity as the main security goal of the SAML V2.0 Web Browser SSO/Response/Artifact Feature.

> 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.

In order to model this security goal, we clarify the meaning of three statements: (i) Who is the service provider's communication partner considered that SSL/TLS does not provide client authentication? (ii) How is the user identity idu derived? (iii) What does a successful protocol run imply?

As the service provider P has no means to determine "its communication partner" before the SAML protocol run, we refer to the secure channel that P has with that partner. A unilaterally authenticated channel like SSL or TLS only guarantees that there is one fixed communication partner, but not the client's identity. We refer to the channel by an identifier cid.

The user identity idu is derived from the assertion $assert^*$ retrieved in Step 8 in Section 2.2. For simplicity, we assume that P derives idu from the subject identifier in this assertion: $assert^*.Subject$.

Thus we model the successful termination of a SAML protocol run by an output $(\mathsf{accepted}, cid, idu, att)$, which binds the user identity idu to the secure channel identified by cid.

We make the authenticity definition for one given service provider and one given identity provider. Though not explicitly specified in SAML, we can assume

[1]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 $Res.InResponseTo = AuthnReq.ID$.

that the different principals have means to enforce the SAML constraints about identifier and URI uniqueness and addressing, which we call setup. Thus we refine the informal authenticity statement of the beginning of this section as follows:

Honest principals. P is an honest service provider and I its honest identity provider. U is an honest user with correct browser B who has the identity idu at the identity provider I.

Setup execution. U, P, and I have executed setup according to SAML 2.0.

Condition for authenticity. *IF* the service provider P obtains an output $(\mathsf{accepted}, cid, idu, att)$ from the SAML V2.0 Web Browser SSO/Response/Artifact Feature, ...

Authenticity claim. ... *THEN* the secure channel with channel identifier cid is indeed a channel with the honest user U with identity idu (unless the user authentication and tracking of U at I is compromised by other means).

4 Analysis of Selected Security Measures

In this section, we analyze selected security measures added to SAML V2.0 [18, Bindings, Security Consideration] and described in the SSTC response [15]. 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 order 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, [7] 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 [7] 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, Bindings, Section 3.6.5]. This measure defeats the so-called Referer attack of [7]. 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 [15, Section 1.3.4], [18, Security Consideration, 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 [7] 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 [18, Core, Section 3.5.1], [18, Bindings, 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 [7]; 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. For completeness, the scenario up to the leakage of art is shown in Figure 3 (Appendix A.2).

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.3).

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.

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

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. 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:

> A browser B shows consistent Referer tag behavior if 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, the predicate $\mathsf{SetsReferer}$ is TRUE if the browser does set Referer tags. A *Janus artifact* is a SAML artifact which is chosen from two artifacts transported in different ways in the protocol, depending on $\mathsf{SetsReferer}$:

$$\textbf{if } \mathsf{SetsReferer}(\mathsf{B}) \textbf{ then } art = art_1 \textbf{ else } art_2$$

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.) The two artifacts are transmitted in different ways to ensure that an adversary can obtain at most one.

Notation 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 $\mathsf{SAMLart}$ we denote the domain of SAML artifacts; we only need to know here that a correctly chosen SAML artifact has a sufficiently large (pseudo-)random part to be guessable by an adversary only with negligible probability.

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 $\mathsf{gen\_art}$. 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.[2] Given a Step 4.z request, I derives a user identity $idu \neq \epsilon$

[2]In implementations of I where the principal identification in Step 4 is based upon a GET request

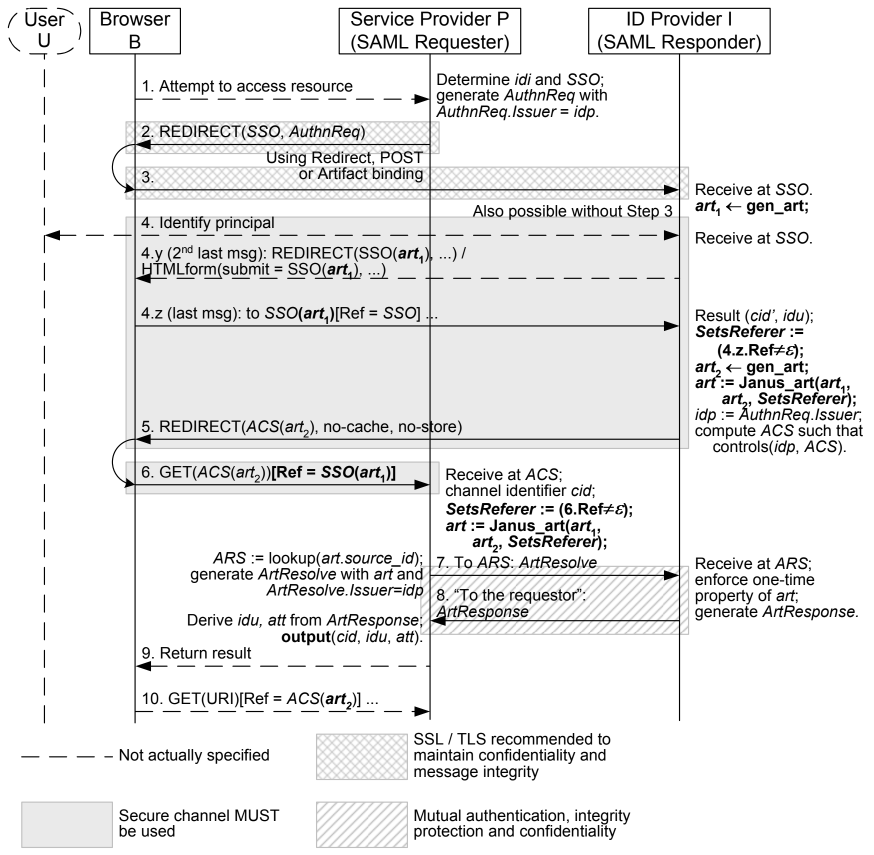
User U
Browser B
Service Provider P (SAML Requester)
ID Provider I (SAML Responder)
1. Attempt to access resource
Determine idi and SSO; generate AuthnReq with AuthnReq.Issuer = idp.
2. REDIRECT(SSO, AuthnReq)
Using Redirect, POST or Artifact binding
3.
Receive at SSO. art1 ← gen_art;
Also possible without 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.

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.

In Step 5, I generates a second SAML artifact art_2 independently from art_1 using gen_art, and computes a Janus artifact art from 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. Then P can also apply the function Janus_art. Note that if SetsReferer(B) is false, then P does not

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.

have art_1, but also does not need it in this function.

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

The core security property achieved by Janus artifacts is the *general prevention of artifact information flow*. This property is crucial for the security of all SAML artifact profiles and renders the SAML 2.0 WebSSO Browser/Janus Artifact Feature superior to other proposals. We claim the following information flow property:

> If the trust and setup preconditions of authenticity of Section 3 are true, and the browser has consistent Referer tag behavior (Section 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.

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 uses art_1 as the Janus artifact art. 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 art_1 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. P sees art_1 in the Referer tag and therefore computes the same Janus 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 or art_1 in the profile. Hence an adversary has no advantage in guessing 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 Efficiency

How does the Janus profile behave compared to other measures that address the artifact information flow problem? We consider three dimensions of efficiency measures: message complexity, storage complexity at the service provider P, and assumptions about the environment. We compare our proposal to the original SAML 2.0 WebSSO Browser/Artifact Feature and discuss two prominent alternatives to enhance its security apart from Janus. We give an overview of these methods and their efficiency in Table 1. In this table, we use the variable n for the total number of artifacts I has sent/P has seen; f denotes the number of artifacts that failed to be send to the identity provider I.

We start with the alternative 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 (see Table 1, column "Redirect"). 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 (see Table 1, column "Full Blacklist"). 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.

Table 1: Efficiency measures for SAML 2.0 WebSSO Browser/Artifact proposals.

	SAML 2.0	**Redirect**	**Full Blacklist**	**Janus**
Msg POST Auth	0	+2	0	0
Msg GET Auth	0	+2	0	(+2)
Storage P	$O(f)$	$O(f)$	$O(n)$	$O(0)$
Storage I	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Assumptions	timing P,I ∞ cache	timing P,I ∞ cache	timing P,I ∞ cache	consistent B

With the SAML Web Browser SSO/Response/Janus Artifact Feature, we pursue a solution that does not have these drawbacks (see Table 1, column "Janus"). 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 reach-able 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] M. Backes and T. Groß. Tailoring the Dolev-Yao abstraction to web services realities. In *Proceedings of the 2005 ACM Workshop on Secure Web Services (SWS)*, pages 65–74. ACM Press, Nov. 2005.

[2] M. Backes, S. Mödersheim, B. Pfitzmann, and L. Viganò. Symbolic and cryptographic analysis of the Secure WS-ReliableMessaging scenario. In *In Foundations of Software Science and Computation Structures (FOSSACS)*, volume 3921 of *Lecture Notes in Computer Science*, pages 428–445. Springer-Verlag, Berlin Germany, 2006.

[3] K. Bhargavan, C. Fournet, and A. 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.

[4] S. Cantor and M. Erdos. Shibboleth-architecture draft v05, May 2002. `http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v05.pdf`.

[5] K. Fu, E. Sit, K. Smith, and N. Feamster. Dos and don'ts of client authentication on the web. In *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., Aug. 2001. USENIX. An extended version is available as MIT-LCS-TR-818.

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

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

[8] T. Groß and B. Pfitzmann. Proving a WS-Federation Passive Requestor profile. In *2004 ACM Workshop on Secure Web Services (SWS)*, Washington, DC, USA, Oct. 2004. ACM Press.

[9] T. Groß, B. Pfitzmann, and A.-R. 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.

[10] T. Groß, B. Pfitzmann, and A.-R. Sadeghi. Proving a WS-Federation Passive Requestor profile with a browser model. In *Proceedings of the 2005 ACM Workshop on Secure Web Services (SWS)*, pages 54–64. ACM Press, Nov. 2005.

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

[12] C. Kaler and A. N. (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/`.

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

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

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

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

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

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

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

A Appendix

A.1 SAML 2.0 as a Multipart Standard

Table 2: SAML 2.0 as multi-part standard

Document	**Element**	**Description**
Core		Basic message formats.
	Assertion	Corresponds 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.
	Request/Response	Message pairs that can transport assertions.
	AuthnRequest/ Response	Transport a request for authentication and the resulting one or more assertions.
	ArtifactResolve	Asks for the real response referred to by an artifact.
	ArtifactResponse	Delivers this response.
Bindings		Bind messages to concrete transport protocols (e.g., HTTP)
	Artifact Binding	Describes how an artifact is transported through a browser. The corresponding actual SAML message is retrieved directly, using the artifact as a reference.
Profiles		Define entire sequences of message exchanges, for instance for single sign-on. Due to reference to the bindings, these profiles are rather modular.

A.2 Details of the Artifact Accumulation

In Figure 3 we depict the artifact accumulation attack.

A.3 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 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.

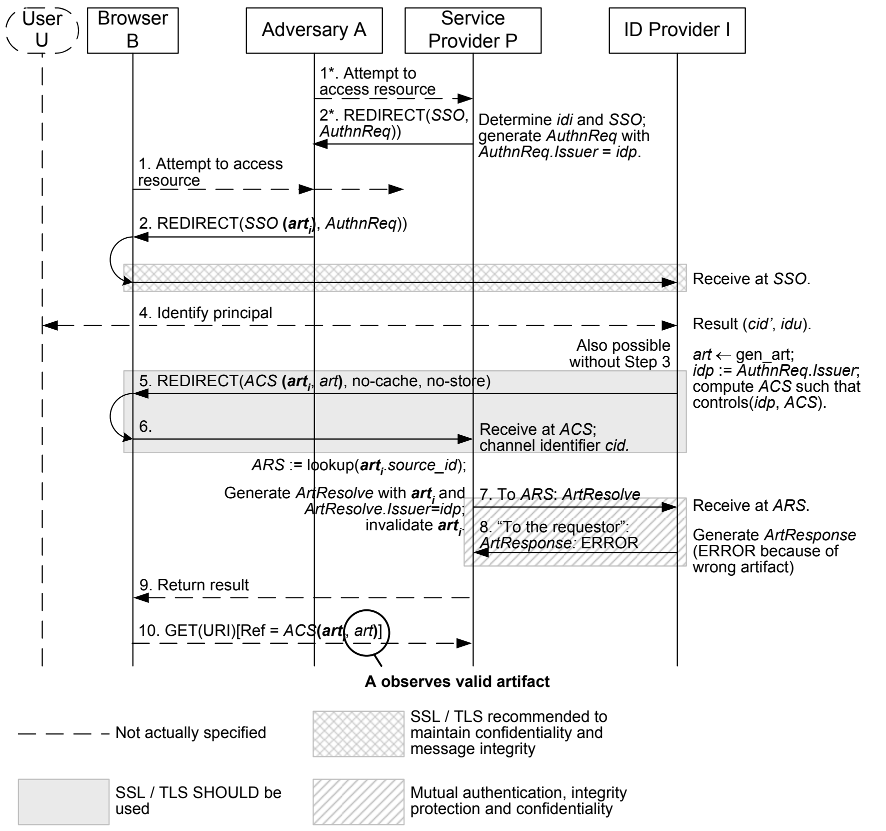
User U
Browser B
Adversary A
Service Provider P
ID Provider I
1*. Attempt to access resource
2*. REDIRECT(SSO, AuthnReq))
Determine idi and SSO; generate AuthnReq with AuthnReq.Issuer = idp.
1. Attempt to access resource
2. REDIRECT(SSO (art_i), AuthnReq))
Receive at SSO.
4. Identify principal
Result (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

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.