Security Analysis of Web-based Identity Federation

Apurva Kumar
IBM Research - India
New Delhi, India
Email: kapurva@in.ibm.com

***Abstract*—While security of cross-domain single sign-on is a thoroughly researched subject, the closely related web identity federation has not been recognized as a distinct problem requiring analysis in its own right. In this paper, we describe a generic approach for analyzing security of web protocols through a framework for reasoning about user actions. We then use this framework to analyze security of important web identity federation protocols. We show that a secure single sign-on protocol does not necessarily ensure safety of linking identities across domains. Our analysis discovers limitations in current web identity management standards that can allow an attacker to create fraudulent identity associations across domains. We propose changes to the workflow and suggest measures for ensuring integrity of cross-domain associations in standard based implementations.**

***Keywords*-Security Protocols; Identity Management; Federated Identity; Web Security.**

I. Introduction

User management services were one of the first to be offloaded to third party cloud vendors. Today, a large number of service providers rely on trusted identity providers for managing users and their resources. At the core of these interactions involving multiple providers are a set of web-based workflows that have emerged as de-facto standards.

The most commonly deployed web identity management workflow is that of providing *single sign-on* (SSO) across domains. Web-browser cookies that are primarily used for managing authentication state in web applications, are never transmitted across DNS domains, thus requiring other mechanisms for managing sessions across providers. To avoid usage of proprietary schemes for cross-domain SSO which would adversely impact interoperability in heterogeneous collaboration environments, there was a need for standardization in this space. Identity management standards such as Security Assertion Markup Language (SAML 2.0) [13] and OpenID [23] represent industry efforts in this direction.

While identities have been consolidated to a large extent using SSO, most users hold accounts with more than one major identity providers such as Google, Yahoo and Facebook. To improve user experience while interacting with multiple domains, another identity management use case - referred to as *identity federation* - which allows users to link their accounts across domains, is now assuming importance. Among the prominent identity standards, only SAML 2.0 explicitly deals with the problem of linking user accounts across domain. More recent standards like OpenID seem to consider it as a minor extension of SSO and do not standardize the workflow.

Interestingly, while SAML SSO has been one of the more analyzed protocols in literature, the account linking facility it provides does not seem to have been the subject of a previous protocol analysis work. For this reason, the workflow described in [13] is widely presumed to be correct. Moreover, since later standards like OpenID do not explicitly address account linking, the SAML workflow for establishing federated identity is often used as a benchmark by application developers.

In this paper, we consider the problem of analyzing security of the identity federation workflow. For concreteness, our analysis is based on the account linking flow as described in the SAML 2.0 specifications. However, given that this is the only industry standard, results of our analysis potentially apply to large number of existing implementations of cross-domain identity linking.

Our approach for analyzing security of web protocols is based on a framework for reasoning about user actions. In the absence of identifying keys and global identities, it is often more important to know whether a user recently performed an action rather than knowing its identity. Our approach combines two contrasting styles of *inference construction* and *attack construction*. Inference construction approaches, first popularized by the Burrows, Abadi and Needham (BAN) [12] logic attempt to use inference in specialized logics to establish required beliefs at protocol participants. Attack construction approaches, on the other hand, reason about intruder knowledge and perform state-space exploration to construct attacks.

The intuition behind the hybrid approach described in section 4, is based on the observation that evaluating security of web protocols, involves two key elements: (*i*) establishing agreement between service providers about the context of the transaction, (*ii*) ensuring that tokens identifying users cannot be stolen or misused. The former requires establishing belief about protocol parameters at honest participants (service providers), while the latter requires reasoning about intruder knowledge. While task (*i*) is suited for the inference construction style of analysis, (*ii*) is more appropriate for model checking based approaches that perform state-space exploration.

For the first stage of analysis, we use an extension of BAN in which some common web mechanisms have been formalized as primitives. For the second stage of our hybrid approach, we use a generic model for web protocols developed using Alloy [19] - a SAT based model analysis tool - to analyze secrecy properties. An important aspect of our approach is that conclusions from belief analysis are used to further constrain the protocol model. This results in simplifications that drastically reduce the search-space needed to be explored by the model-checker.

Our analysis of identity federation (section 5) provides tremendous insight into design and analysis of web identity management protocols. Of particular interest is a vulnerability that allows an attacker to manipulate cross-domain identity associations established through the standard identity federation protocol flow. We propose a simple resolution for the issue and discuss how to incorporate it in SAML and contemporary identity management solutions.

II. Background and related work

In their seminal work [16], Dolev and Yao proposed, the widely accepted adversary model where intruder has the ability to read, alter, encrypt, decrypt, compose and deconstruct messages. The model is not ideally suited for the web. On one hand, web protocol analysis can be greatly simplified using a much more restrictive model designed for secure (SSL/TLS) communication. On the other hand, despite its flexibility, Dolev-Yao model cannot capture certain types of browser-based attacks e.g. inducing an honest principal into unintentionally sending messages (including secrets) to a server chosen by the attacker, manipulating redirection URLs etc. The adversary model we use in this work can be considered a modulation of Dolev-Yao model in which the intruder does not have access to all messages, but at the same time has the ability to exploit browser-based communication to forge requests, manipulate redirection endpoints etc.

The authors of [14] attempt to formalize a standard representation of the Dolev-Yao model where states are represented as multi-set of facts and transition rules are defined to describe protocol behavior as well as intruder capabilities. Using this formalism, the authors establish undecidability of the secrecy property for unbounded number of sessions, even with bounded message sizes, and encryption depth [17].

ProVerif [11], [7], [8], [9] is a cryptographic protocol verifier, developed by Bruno Blanchet, based on the applied-pi calculus formalism [3]. The input can be specified directly as a sequence of Horn clauses or as a process in a variant of the applied pi-calculus. It is capable of evaluating reachability properties [1], [2], correspondence properties [8] and observational equivalence [10].

An alternative to the state space analysis is based on the notion of strand-spaces [18], a graph-theoretic interpretation of the Dolev-Yao model. Several efforts have been made to automate analysis using this formalism, most notable being the Athena tool [24]. The more recent Scyther tool [15] is also based on an extension of strand space concept.

Inference construction approaches attempt to use inference in specialized logics to establish required beliefs at protocol participants. The logic of authentication described in [12], commonly known as BAN, was one of the first successful attempts at representing and reasoning about security properties of protocols. [4] provides semantics of the logic and discusses its soundness. A brief discussion on belief logics and their automation can be found in [22].

We observe that the formalisms discussed above can benefit significantly by providing native support for web protocols. E.g., authors of [5], use the SATMC tool [6], without significant changes, to analyze the SAML Single Sign-on (SSO) protocol. The multiset rewriting based formulation is quite complex for a protocol of this size. Moreover, use of the standard Dolev-Yao attacker without support for session-fixation attacks, results in a vulnerability not surfacing in their analysis. The attack we discover on the identity federation protocol (described in detail in section V) is a direct consequence of this weakness.

Our work is partially based on the belief logic for web first proposed in [20] and developed further in [21]. In [22], we first proposed use of belief logic in conjunction with a model checking stage. The interaction between the two stages is based on assumptions being made during inference construction stage and validated in the model checking stage. In this work, we significantly advance the approach in [22] by not relying on knowledge of such domain-specific assumptions. Instead, we represent output of belief logic analysis directly as constraints that relate protocol sessions at honest participants (service providers) during the model checking stage. Further, the property being checked corresponds exactly to the protocol goal, rather than a generic property such as secrecy or secure redirection. Finally, in this paper, we analyze the account linking workflow while in [22], security of a web authorization delegation protocol (OAuth 1.0) was examined.

III. Web SSO and ID Federation

In this section, we provide an overview of the premier web SSO standards and discuss how they can be used to establish federated identity.

A. The SSO Workflow

The workflow involves a web user interacting through a browser with two web based service providers, one of which acts as the identity provider. A user requesting service S at *service provider* (SP) site is redirected to *identity provider* (IdP) with an authentication request. The request also includes a callback URL to be used by IdP to redirect the user back to SP. After authenticating the user as Q, IdP redirects user back to SP site with an authentication response containing a signed token asserting the identity

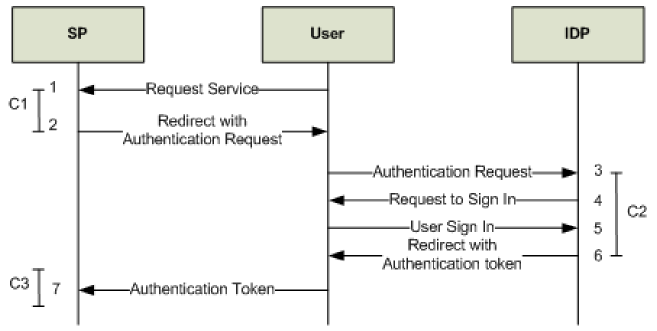

Figure 1: The browser SSO workflow.

Q. SP validates the token and identifies the user as Q. In the figure, C1-C3 identify secure SSL/TLS channels with unilateral (server) authentication. The generic workflow of Figure 1 can be used to describe both the OpenID 2.0 SSO protocol as well the "SP-initiated SSO" of SAML 2.0.

OpenID specifications refer to the identity provider as the OpenID provider (OP) and service provider as the relying party (RP). In this paper, to avoid confusion, we follow the SAML terminology for these protocol roles. Apart from terminology, important differences between the two protocols are noted below:

1) The authentication request is signed by SP in case of SAML while it is unsigned in case of OpenID. Also, while the SAML request contains a nonce which is used to identify the session, there is no such identifier in the OpenID request.
2) OpenID optionally allows the user to claim an identity in step 1. The URL of the identifier such as `http://openid.example.org/alice` is used to discover the IdP endpoint URL.
3) The authentication response for both protocols contains a token signed by the identity provider asserting the authenticated identity (Q) of the user. In case of OpenID, the signed information also includes the callback URL from the authentication request.

In addition to the SP initiated SSO workflow, SAML also provides an alternative flow that starts at the identity provider. This "IdP-initiated SSO" is similar except the first three steps in Figure 1 are omitted.

B. *The Identity Federation Workflow*

The SAML 2.0 specifications include a variation of the SSO workflow that can be used to establish and manage federated identity. The workflow can be used by a user to link her identity at the service provider with her identity at the identity provider. The workflow is simply an IDP or SP initiated SSO workflow followed by another user sign-in action at the SP. After this sign-in, the SP creates an association between user accounts at SP and IdP. In future, it can automatically map the remote IdP identity to a local identity without needing to perform authentication.

Table I: Operators in Extended Logic

$P \mid\equiv X$: P believes X	$P \overset{K}{\longleftrightarrow} Q$: Shared key K
$P \triangleleft X$: P sees X	$\underset{K}{\longmapsto} Q$: Public key K belongs to Q
$P\mid\sim X$: P said X	$P \overset{Y}{\rightleftharpoons} Q$: Shared secret Y
$P\mid\Rightarrow X$: P controls X	$\sharp X$: Fresh X
$\{X\}_K$: X encrypted by K	$\langle X\rangle_Y$: X combined with Y
$P \overset{\Delta}{\longleftrightarrow} U_c$: Secure channel C	$[\![X]\!]_C$: X over secure channel C
$X \rightsquigarrow Aname(v_{\sigma_1}, \ldots, v_{\sigma_n})$: X associated with action	$U_c \ni X$: User U_c possesses X
$U_c \quad \triangleright Aname(v_{\sigma_1}, \ldots, v_{\sigma_n})$: User U_c performs action	$Pname = val$: Parameter *Pname* has value *val*

The OpenID 2.0 specifications do not explicitly provide a workflow for linking identities. However, typical account linking implementations use a workflow analogous to SAML, i.e. the OpenID SSO protocol followed by a sign-in action at the service provider.

IV. Generic Approach for Analyzing Web Protocols

In this section, we introduce our hybrid analysis approach which is a combination of the two contrasting style of inference construction (section IV-A) and attack construction (section IV-B). In section IV-C, we show how the two styles can be combined to significantly reduce overall complexity of the approach.

A. *Belief Logic for the Web*

1) Extended Syntax: Protocol messages are 'idealized' into expressions representing formula in the specialized logic. A formula in BAN logic [12] is constructed using operators from Table I. P and Q range over principals. The three statements about keys and secrets represent atomic statements. X represents a BAN formula constructed using one or more BAN operators. The expression $\sharp X$ means that the message X is fresh and has not been used before the current run of the protocol. This is especially true for a nonce, a sequence number or timestamp generated with this specific purpose. Nonces are used in protocols to defeat replay attacks from previous executions of the protocol. The *said* and *freshness* operators can be combined into a single *has recently said* or *says* operator.

We introduce two new types of objects (sorts) to the logic: *user* and *action*. A user is defined as the client side of a secure channel which models a unilateral SSL/TLS session with server authentication. We use the channel identifier as a subscript in our notation for user. We assume *Aname* to range over function symbols representing types of user actions. A user action type has a signature of the form $\sigma_1 \times \ldots \times \sigma_n \longrightarrow action$, where $\sigma_1, \ldots \sigma_n$ are other sorts of the logic. Actions could either be generic or specific to a particular application. For example, signing in as principal *Q*, represented as $SignIn(Q)$, is a generic action.

We also introduce statements asserting values of variables in the protocol. *Pname* ranges over variable names. Variables can be used to state beliefs about protocol roles and other protocol specific parameters which are useful in establishing correspondence properties. Using variable names in idealized expressions has the effect of binding each occurrence of the name with the same value in a given protocol run. We also allow use of action and parameter names in jurisdiction (*P* controls *X*) expressions. The new operators are described in the last three rows of Table I.

2) Inference Rules: BAN defines a set of inference rules for deriving new beliefs from old ones. We describe only those rules of BAN that are relevant for the analysis of section V. The message-origin inference rule R1 states that if *P* knows that *K* is the public key belonging to *Q* and it sees a message *X* encrypted by the corresponding private key K^{-1}, then *P* is entitled to believe that *Q* said *X*.

$$\frac{P \mid\equiv \underset{K}{\mapsto} Q, P \triangleleft \{X\}_{K^{-1}}}{P \mid\equiv Q \mid\sim X} \tag{R1}$$

A nonce-verification rule R2 states that, in addition if the message is known to be fresh, then *P* believes that *Q* must still believe *X*. Further, the *jurisdiction rule* R3 states that, if in addition, *P* also believes that *Q* is an authority on the subject of *X* (i.e. *Q* controls *X*), then *P* is entitled to believe *X* itself.

$$\frac{P \mid\equiv Q \mid\sim X, P \mid\equiv \sharp X}{P \mid\equiv Q \mid\equiv X} \tag{R2}$$

$$\frac{P \mid\equiv Q \mid\equiv X, P \mid\equiv Q \mid\Rightarrow X}{P \mid\equiv X} \tag{R3}$$

New inference rules. We extend BAN through inference rules that apply to communication over one-sided (server authentication only) SSL/TLS secure channel. R4.1 says that if a principal *P* (usually server) believes that a user U_C is communicating over a secure channel *C*, then any actions it sees over the secure channel *C* can be attributed to user U_C. According to R4.2, any tokens seen over a secure channel are assumed to be possessed by the user. R4.3 states that when the client side receives a statement *X* over a secure channel, it is entitled to believe that the server principal has recently said (*says*) *X*.

$$\frac{P \mid\equiv (P \overset{\triangle}{\leftrightarrow} U_c), P \triangleleft [\![action]\!]_C}{P \mid\equiv (U_c \triangleright action)} \tag{R4.1}$$

$$\frac{P \mid\equiv (P \overset{\triangle}{\leftrightarrow} U_c), P \triangleleft [\![X]\!]_C}{P \mid\equiv (U_c \ni X)} \tag{R4.2}$$

$$\frac{U_c \mid\equiv (P \overset{\triangle}{\leftrightarrow} U_c), U_c \triangleleft [\![X]\!]_C}{U_c \mid\equiv (P\, says\, X)} \tag{R4.3}$$

B. Generic Model for Web Protocols

In the following sections, we describe our generic model for web protocols implemented using *Alloy* [19] - a declarative language for describing structures and a tool for exploring them. An alloy model specifies a set of constraints that apply to objects in the domain being modeled. Alloy Analyzer is a solver that takes constraints of a model and finds structures satisfying them using a SAT solver. Thus technically, it is a model-finder rather than a model-checker. A *signature* and a constraint on the signature are declared below:

```
sig S extends E {
       F: one T }
fact {
       all s:S | s.F in X }
```

It is often useful to think of Alloy as an object-oriented language, but underneath the covers `S` is a subset of `E` and `F` is a relation that maps each of `S` to a single `T`.

Fact statements represent constraints that must always hold. Quantified expressions of the form *quantifier* `s: S | F` mean that constraint `F` holds for `all`, `no`, `lone` (zero or one), `some` (at least one) or `one` element(s) of `S`. Fact expressions that apply to a particular signature (as is the case above) can be directly appended to the signature within curly brackets. Assertions (`assert` ...) are properties against which the specification needs to be checked. A `check` command causes the analyzer to search for a counter-example to show that the assertion does not hold. Alloy checks models of finite sizes using a specified scope which limits the maximum size of top level signatures.

1) Modeling Principals: The signature `Process` declares a set of all principals. It is extended by signatures `Server` and `User` which are (disjoint) subsets representing web service providers and end users respectively. Also declared are set of all keys (`Key`), private keys (`PvtKey`), instants (`Time`), cookies (`Cookie`) and values, (`Value`, `CkValue`, `TkValue`). A private key is associated with a public key through the relation `pubkey`. A principal knows a set of keys (`knownkeys`) and a server principal owns a private key (`ownedkey`). Most web protocols requiring security analysis involve two collaborating service providers. The `peer` relation maps a server to its collaborating partner. The relations `uniquecookie` and `uniqueval` associate a `Server` with a unique cookie and a secret/nonce value, respectively. Minor changes in the declarations are required to represent protocols needing more than cookie, secret per server role. Constraint on `uniquecookie` relation ensures that cookie points to the correct server.

Listing 1: Modeling service providers and users

```
abstract sig Process {
   knownkeys: set Key
}
```

```
sig Time { }
sig Key { }
sig Value { }
sig TkValue extends Value { }
sig CkValue extends Value { }
sig SessionID extends Value { }

sig Cookie {
    value: one CkValue,
    server: one Server }

sig PvtKey extends Key {
    pubkey: one Key
} { pubkey != this }

sig Server extends Process {
    ownedkey: one PvtKey,
    peer: one Server,
    uniqueval: one TkValue,
    uniquecookie: one Cookie
    sessions: set Session
} { peer != this, uniquecookie.server =
   this }

sig User extends Process {
    seentokens: set TkValue->Time,
    knowncookies: set Cookie->Time
} { ... }

sig HUser extends User { }

fact {
    all k1,k2: PvtKey|k1 != k2 =>
    k1.pubkey != k2.pubkey
}

fact {
    all s1,s2: Server|s1 != s2 =>
    (s1.uniqueval != s2.uniqueval)
    && (s1.ownedkey != s2.ownedkey)
    && (s1.uniquecookie)!= (s2.
       uniquecookie)
}
```

A `User` participates in two relations. `seentokens` associates the user to a set of (value, time) pairs each indicating that a *value* was known to the user at *time*. The relation `knowncookies` provides a similar association for cookies known to the user. Finally, the facts represent constraints on keys, nonces and cookies.

2) Protocol Session: We allow a server process to be involved in multiple protocol sessions, so that our approach can support attacks based on multiple sessions. Signature `session` defines a generic protocol session having an identifier and principals in the *assertion consumer* and *provider* roles.

```
sig Session{
    id: one SessionID,
    consumer: one Server,
    provider: one Server
}
```

3) Protocol Messages: The signature `Sent` is used to declare a set of possible protocol HTTP messages. Each message has a `sender` and `receiver` principal and is associated with a `time` when it is transmitted. The other relations on message are a set of values (`content`) and a set of cookies (`cookies`) contained in the message. A message may also contain a redirection URL (`redirectURL`), if it represents an HTTP redirect.

The message content is a set of tokens, each value can be a simple value or an encrypted formula containing a set of values (simple or encrypted) represented by `ComplexVal` encrypted by key `enckey`. Further, a token can also be associated with an action (`ActionTkn`). In this case, the context associated with the token such as provider, consumer, session identifier etc. is also included in the structure. Protocol specific action tokens can be defined by extending `ActionTkn`.

```
sig ComplexVal extends TkValue {
   vals: set TkValue,
   enckey: lone Key }

sig ActionTkn extends TkValue {
   consumer: set Server,
   provider: lone Server,
   session: lone SessionID }

sig URL { target: one Process }

sig Sent {
   cookies: set Cookie,
   sender: one Process,
   receiver: one Process,
   time: one Time,
   content: set TkValue,
   session: lone SessionID,
   redirectURL: lone URL,
   enckey: lone Key
}{ sender in HUser =>
   (all c: Cookie | c in cookies <=>
   c->time in sender.knowncookies
   && c.server = receiver)

   sender in User =>
   (all v: TkValue | v in content =>
   v->time in sender.seentokens)

   enckey in sender.knownkeys
          +sender.ownedkey
   sender != receiver }
```

The first constraint requires that a message sent by an honest user (`HUser`) shall only contain cookies that were known to the sender at the time of sending the message and were received earlier from the target of that message. The bi-implication requires that all such cookies must necessarily be included in the message. Next is a similar constraint requiring any values contained in a message to be known to the user. The third constraint requires the encryption key to be known to the sender. The last constraint says that sender and receiver of a message have to be distinct.

4) Learning Rules: The rules for a user learning new secret values or cookies are expressed as constraints appended to the `User` signature. The utility `ordering` is use to order elements of `Time`. The first constraint implies that a pair (`cookie`, `t`) appears in `knowncookies` if and only if the user has seen a message containing cookie at a time $\leq$ `t`. A similar constraint for other tokens is also specified.

```
open util/ordering[Time] as ord
sig User extends Process { ...} {
   all c: Cookie | all t: Time |
   (c->t in knowncookies <=>
   some s: Sent | c in s.cookies
   && s.receiver = this
   && t.ord/gte[s.time])

   all v: TkValue | all t: Time |
   (v->t in seentokens <=>
   some s: Sent | v in s.content
   && s.receiver = this
   && s.enckey in s.reciver.knownkeys
   && t.ord/gte[s.time])
}
```

5) Protocol Flow: The signature `ProtoSeq` represents all possible sequences of messages under generic and protocol specific constraints. If `p` and `q` are possible sent messages, then `p->q` appearing in the sequence implies that receiver of `p` is the sender of `q`. Also the timestamp on `q` must be the next time instant following the timestamp of `p`.

```
sig ProtoSeq {
    sequence: set Sent->Sent
}{  all p,q: Sent | (p->q in sequence)
   =>
    (q.sender = p.receiver)
    all p,q: Sent | (p->q in sequence)
        =>
    (q.time = ord/next[p.time])
    all p: Sent | (p.receiver in HUser)
    && p.redirectURL =>
    (some q: Sent | (p->q in sequence)
    && (q.receiver = p.redirectURL.
       target)
    && (q.content = p.content)) }
```

The last generic constraint describes handling of an HTTP redirect for an honest user (`HUser`). It specifies that if an honest user receives a redirect message, the next message in the sequence must be a message sent by this user to the target of the redirection URL. The message should include any values/tokens received in the redirect. The other constraints on protocol sequence are specific to the protocol being modeled.

6) Adversary Model: The intruder is simply a `User`. The redirection constraint for honest user does not apply to it. The intruder learns new values based on learning rules for tokens and can only send seen tokens (as per constraint on `Sent` discussed earlier). Communication from a dishonest to honest user (e.g. through a malicious hyperlink) is modeled as a redirect message generated by the dishonest user.

In addition, we include the possibility of dishonest servers colluding with the intruder. This is done by allowing dishonest principals to share any tokens they obtain with the attacker. This is modeled as a message from a process belonging to a signature representing a corrupted principal role to `User`.

C. Hybrid Analysis Approach

The two techniques described in sections IV-A and IV-B are employed in the first and second stages of our analysis, respectively. The stages represent different views of the protocol:

- In the first stage (belief logic analysis), the protocol is viewed from the perspective of honest participants. The aim is to establish agreement between service providers about the session parameters and the information conveyed by tokens across domains.
- In the second stage, the protocol is viewed from the perspective of the intruder and a knowledge flow analysis is performed for reasoning about actions based on secrets.

The two stages are combined in the following manner. Beliefs established in the first stage are used to constrain the protocol flow in the knowledge analysis stage. In particular, beliefs about end-points, callback URLs etc. are used to direct the protocol messages to intended recipients. Belief about session parameters are used to initialize the context of tokens representing cross-domain assertions and in evaluating token validation constraints. These constraints significantly simplify the second stage of analysis by considerably reducing the state-space to be searched.

Based on the beliefs established at participants and based on what knowledge can become available to the intruder, the security analysis problem then becomes to find whether a protocol execution is possible that violates the assertion that token must be possessed by the user who performed the corresponding action. We illustrate working of the hybrid approach in section V through the analysis of identity federation workflow.

V. Analysis of Identity Federation

The workflow we discuss here corresponds to “Federation via persistent pseudonym identifiers” described in the SAML 2.0 protocol specifications [13]. The objective of this workflow is to allow a web user to link identities across security domains. Despite being a widely deployed identity federation protocol, it does not appear to be the subject of scrutiny in a prior security analysis work.

Let us assume a customer is already registered as user R_c at a service provider (SP) site and user Q_p at the identity provider (IdP) site. While at the SP site, the user chooses to authenticate using his identity at IdP. The single sign-on workflow is invoked. When the user returns to the SP site with a signed token from IdP asserting the identity Q_p, SP requests the user to sign-in with the local identity R_c. On successful authentication, SP links local principal name R_c with remote principal name Q_p. In future, when SP sees a user carrying a SAML token from IdP asserting identity Q_p, it automatically signs in the user as R_c. All communication is assumed to take place on secure (SSL/TLS) channels identified by C1-C3. The message exchange is illustrated in Figure 2. We note that SAML allows linking the SP identity with a persistent pseudonym rather than the actual account name at IdP. We ignore this feature in our analysis, as it does not impact security of the linking operation.

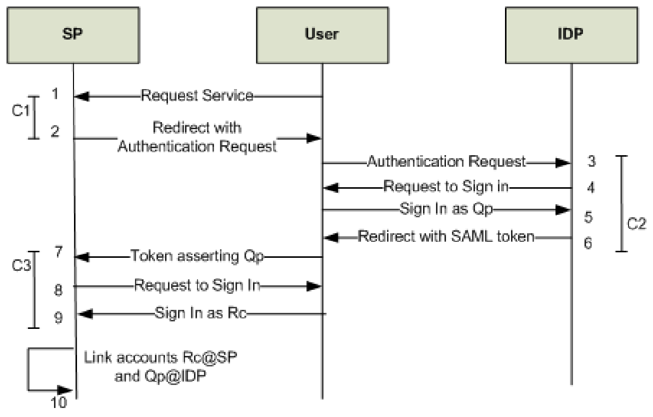

Figure 2: Identity Federation using SAML.

A. Stage 1: Belief Logic Analysis

Messages 3, 5, 7, 9 are the only messages received by either *C* (playing the SP role) and *P* (playing the IdP role) and are idealized in (1). The SAML authentication request (message 3) signed by *C* comprises of statements about protocol roles using parameters *sp*, *idp* and callback URL (*cb*) to be used for redirection in step 6. The nonce, N_{cp} represents combination of a request identifier and a timestamp. For brevity, we omit the argument identifying the session in our analysis and write parameters such $sp(N_{cp})$ and $idp(N_{cp})$ simply as *sp*, *idp*. The message is signed using the private key of service provider. In the idealization, we also include the term $auth\_req(N_{cp}, P)$. This allows us to additionally represent the fact that message 3 is a SAML request for *P* with identifier N_{cp}.

Messages 5, 9 represent sign-in actions at *P*, *C*, respectively. Message 7 represents the SAML response returned by the identity provider. The message signed by *P* contains a token associated with the sign-in action and statements about protocol roles as per *P*'s belief.

$$\begin{aligned}
&[Msg3]\, U_{C2} \longrightarrow P : [\![\{auth\_req(N_{cp}, P), N_{cp},\\
&\qquad idp = P, sp = C, cb = url_C\}_{K_c^{-1}}]\!]_{C2}\\
&[Msg5]\, U_{C2} \longrightarrow P : [\![SignIn(Q_p)]\!]_{C2}\\
&[Msg7]\, U_{C3} \longrightarrow C : [\![T, \{N_{cp}, T \rightsquigarrow SignIn(Q_p),\\
&\qquad idp = P, sp = C\}_{K_{p-1}}]\!]_{C3}\\
&[Msg9]\, U_{C3} \longrightarrow C : [\![SignIn(R_c)]\!]_{C3} \qquad (1)
\end{aligned}$$

We make the following assumptions at *C* and *P* about secure channels, nonces, public keys, protocol roles and jurisdiction over parameters and actions.

$$\begin{aligned}
&C \mid\equiv C \overset{\Delta}{\leftrightarrow} U_{C1} && P \mid\equiv P \overset{\Delta}{\leftrightarrow} U_{C2}\\
&C \mid\equiv C \overset{\Delta}{\leftrightarrow} U_{C3} &&\\
&C \mid\equiv \sharp N_{cp} && P \mid\equiv \sharp N_{cp}\\
&C \mid\equiv \underset{K_p}{\longmapsto} P && P \mid\equiv \underset{K_c}{\longmapsto} C\\
&C \mid\equiv P \mid\Rightarrow SignIn && P \mid\equiv sp(n) \mid\Rightarrow cb(n)\\
&C \mid\equiv sp = C, idp = P, cb = url_C && \text{(A1)}
\end{aligned}$$

In addition, we also use the following protocol specific rule. The rule says that when a principal playing the IdP role in the protocol receives a SAML request, it makes originator of the request as the service provider (relying party) for the session.

$$\begin{aligned}
&P \mid\equiv X \mid\sim auth\_req(n, P) \wedge P \mid\equiv \sharp n \rightarrow\\
&\qquad P \mid\equiv (sp(n) = X, idp(n) = P) \qquad \text{(S1)}
\end{aligned}$$

A BAN type forward chaining analysis of these messages results in the following final conclusions being established at service provider and identity provider:

$$\begin{aligned}
&P \mid\equiv C \mid\sim (auth\_req(N_{cp}, P), N_{cp},\\
&\quad idp = P, sp = C, cb = url_C)\\
&P \mid\equiv (sp = C, idp = P)\\
&P \mid\equiv C \mid\equiv (sp = C, idp = P, cb = url_c)\\
&P \mid\equiv cb = url_c\\
&P \mid\equiv cb = url_c SignIn(Q_p)\\
&C \mid\equiv U_{C3} \ni T\\
&P \mid\equiv (idp = P, sp = C, T \rightsquigarrow SignIn(Q_p))\\
&C \mid\equiv U_{C3} \triangleright SignIn(R_c) \qquad \text{(C1)}
\end{aligned}$$

B. Stage 2: Model Finding using Alloy

We now demonstrate the second stage of analysis which is used to associate users with actions based on possession of tokens. We show how the generic protocol model of section IV-B can be extended for SAML SSO. We also show how results from the belief logic stage in section V-A can be used to simplify analysis in the second stage.

A SAML protocol session extends the generic `Session` signature described in section IV-B2 by including a field to represent the callback URL. The fields `consumer` and `provider` in `Session` are used to identify the service provider and identity provider, respectively. A SAML token extends `ActionTkn` by including a `subject` field identifying the authenticated user.

```
sig Provider extends Server { }
sig Consumer extends Server { }
sig SAMLSession extends Session {
   callback: lone URL
}
sig SAMLRequest extends TkValue {
   session: one SessionID
}
sig SAMLToken extends ActionTkn {
    subject: one User
}
```

Simplification of SAML request. A key simplification that uses the agreement established by belief analysis, is achieved by associating SP and IdP with a single session object. We simplify a SAML request by retaining a reference to the SAML session that created the request at the service provider. When the request is received at IdP, it can infer request parameters from the reference.

We associate two constraints with sessions held at a service provider. Firstly, the `consumer` and `callback` fields must point to itself while `provider` must be set as one of the identity providers. Secondly, for each such session, a SAML request consistent with the session parameters must exist.

```
sig Consumer {...} {
  all s:Session | s in sessions =>
  s.consumer = this
  && s.provider in Provider
  && s.callback.target = this

  all s:Session | s in sessions =>
  some p: Sent | p.content in
      SAMLRequest
  && p.content.session = s.id
  &&  p.sender = s.consumer
  && p.redirectURL.target = s.provider
  && p.receiver in User
}
```

SAML token generation and validation. Instead of modeling the token as an encrypted message, we use conclusions belief analysis to associate the token with a session identifier, service provider and identity provider. The session used to initialize the token is the one identified in the received SAML request. The token generation can thus be formalized using the following constraint:

```
all p: Sent | all s: Session |
p.receiver in Provider
&& p.content in SAMLRequest
&& s.id = p.content.session =>

some q: Sent | (p->q in sequence)
&& (q.receiver = p.sender)
&& (q.content in SAMLToken)
&& (q.content.sessionid = s.id)
&& (q.content.subject = p.sender)
&& (q.content.consumer = s.consumer)
&& (q.redirectURL.target = s.consumer)
&& (q.content.provider = p.receiver)
```

A SAML token is considered valid by the recipient service provider if it has a session corresponding to the identifier in the token and the values for consumer and provider fields in the token match with those for the session.

```
all p: Sent | p.receiver in Consumer
&& p.content in SAMLToken =>

some s in p.receiver.sessions
| p.content.session = s.id
&& p.content.consumer = s.consumer &&
p.content.provider = s.provider
```

Goal Assertion of ID Federation. The goal of ID federation is to ensure that the two sign-in actions are performed by the same user. The goal assertion is thus to ensure that the `subject` field in the SAML token must point to the user who sent the message containing the token.

```
assert isSignedIn {
  all p: Sent | p.receiver in Consumer
  && p.content in SAMLToken =>
  p.content.subject = p.sender
}
```

Since the second stage performs reasoning based on secrets, only messages 2, 3, 6, 7 which contain secrets, nonces are retained in the model. These messages are simplified as described above. The goal assertion is checked for all possible structures within a scope in the presence of the adversary described in section IV-B6.

C. Result of Alloy Analysis

We executed the model with constraints and goal assertion as described above. Alloy generates the counter-example shown in Figure 3 in less than 3 seconds on an Intel Core i5 2.4 GHz, 4 GB system for a scope up to 10 messages in a protocol sequence. A correct execution of the protocol should have exactly four messages, corresponding to messages 2, 3, 6, 7 in Figure 2. However, the counter-example shows five messages where the first three messages (corresponding to messages 2, 3, 6) with timestamps `Time0`, `Time1` and `Time2` are exchanged with `User`. This is followed by a message from `User` to the victim (modeling following of a malicious link by `HUser`) and a message from `HUSer` to `Consumer` containing the SAML token actually issued to `User`. This translates to the following attack on the SAML ID federation protocol.

Attack on SAML ID linking An attacker having a valid account *A* at IdP authenticates itself and chooses to be

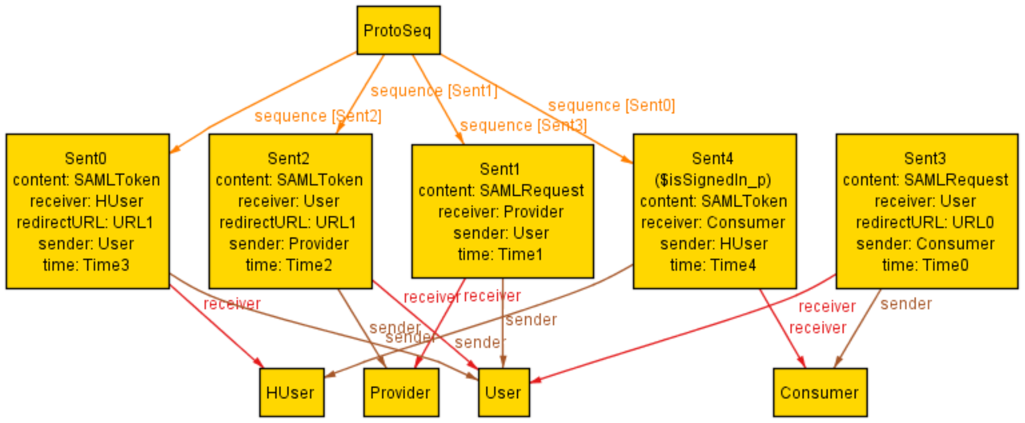

Figure 3: Counter example generated by Alloy for SAML account linking.

redirected to SP. However, instead of following the redirect request from IdP, it extracts the request parameters and induces the victim into clicking a link or submitting a form (depending on whether HTTP redirect binding or POST binding is used for the exchange). Following the link takes the victim to the SP site, unwittingly carrying the SAML token issued to the attacker. The victim has an account (say V) at the SP site and is requested to sign-in. On signing in, SP links local identity V with attacker's identity A at IdP. In future, attacker can sign-in at IdP, get redirected to SP and automatically get access to the victim's account at SP.

D. Fixing the Identity Federation Workflow

A simple fix to the SAML identity federation protocol, is to reverse the order in which the two sign-in actions are performed, thus steps 8, 9 of the protocol becoming steps 2, 3 of the protocol. The only change in our model is a cookie returned along with SAML request to the user and the same included when the user is redirected back to the SP with the token. When we execute the analyzer with this change, we do not obtain a counter-example for a scope allowing execution traces up to 20 messages.

While the fix works, the workflow might not be desirable in all situations. Firstly, the fix does not apply to the IdP initiated variation of the SAML protocol. Secondly, service providers often invoke identity linking when a user tries to sign-in using his account at the IdP for the first time. In this situation having a workflow in which the user authenticates first at the SP is not desirable.

Thus a better resolution is to succeed the identity linking flow in Figure 2 with another authentication (SSO) request from the service provider by redirecting the user to the IdP. Since the user is already signed in at the identity provider, no additional interaction is involved and user is automatically redirected back to the SP.

The modified workflow can be implemented in SAML by simply invoking the standard identity linking workflow and a single sign-on flow in succession. However, the same workflow is still not completely secure with OpenID. This is due to the absence of a nonce value in the OpenID authentication request as noted in section III. The unique request identifier in SAML acts as a safeguard against request forgery attacks on the callback URL. The only means of including a nonce in an OpenID authentication request is in the callback URL (the `return_to` parameter). The same value can then be included in the response by IdP instead of generating a new nonce value. We recommend definition of a new ID linking profile for OpenID documenting this flow.

VI. Conclusions

We studied a variation of the web browser SSO workflow which is used to establish and manage federated identity. While the SSO transaction itself has been thoroughly reviewed, security of account linking problem has not received due attention. In this work, we performed a detailed analysis of the transaction using a technique that has been tailored for reasoning about web protocols. Our approach uses an extension of BAN logic for establishing agreement between service providers. Beliefs established at participants are then used to simplify the second stage that uses a protocol model developed in Alloy to find execution traces that do not conform to a specification. We discovered a vulnerability in the account linking workflow and proposed means of fixing it for standard ID management frameworks. The analysis technique is generic and has the potential of developing into a much needed security analysis tool for web developers.

References

[1] M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. In *ACM SIGPLAN Notices*, volume 37, pages 33–44. ACM, 2002.

[2] M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM (JACM)*, 52(1):102–146, 2005.

[3] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115. ACM, 2001.

[4] M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 201–216. ACM, 1991.

[5] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: Breaking the SAML-based single sign-on for Google Apps. In *Proceedings of the 6th ACM workshop on Formal methods in security engineering*, pages 1–10. ACM, 2008.

[6] A. Armando and L. Compagna. SATMC: A SAT-based model checker for security protocols. *Logics in Artificial Intelligence*, pages 730–733, 2004.

[7] B. Blanchet. From secrecy to authenticity in security protocols. *Static Analysis, 9th International Symposium, SAS 2002, Madrid, Spain, September 17-20, 2002, Proceedings*, pages 29–78, 2002.

[8] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[9] B. Blanchet. Using Horn clauses for analyzing security protocols. *Formal Models and Techniques for Analyzing Security Protocols*, 5:86–111, 2011.

[10] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[11] B. Blanchet et al. An efficient cryptographic protocol verifier based on Prolog rules. In *Proceedings of the 14th IEEE workshop on Computer Security Foundations*, pages 82–96, 2001.

[12] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[13] S. Cantor, I. Kemp, N. Philpott, and E. Maler. Assertions and protocols for the OASIS Security Assertion Markup Language V2.0. *OASIS Standard (March 2005)*, 2005.

[14] I. Cervesato, N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Computer Security Foundations Workshop, 1999. Proceedings of the 12th IEEE*, pages 55–69. IEEE, 1999.

[15] C. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 119–128. ACM, 2008.

[16] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[17] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[18] F. Fábrega, J. Herzog, and J. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Research in Security and Privacy*, pages 160–171. IEEE, 1998.

[19] D. Jackson. Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[20] A. Kumar. Model driven security analysis of IDaaS protocols. In *Service-Oriented Computing - 9th International Conference,ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings*, pages 312–327, 2011.

[21] A. Kumar. A belief logic for analyzing security of web protocols. In *Trust and Trustworthy Computing - 5th International Conference, TRUST 2012, Vienna, Austria, Jun 13-15, 2012. Proceedings*, pages 239–254, 2012.

[22] A. Kumar. Using automated model analysis for reasoning about security of web protocols. In *28th Annual Computer Security Applications Conference, ACSAC 2012, Proceedings*, pages 289–298, 2012.

[23] D. Recordon and D. Reed. OpenID 2.0: A platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital Identity Management*, pages 11–16. ACM, 2006.

[24] D. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.