Using Automated Model Analysis for Reasoning about Security of Web Protocols

Apurva Kumar
IBM Research - India
4 Block C, Institutional Area
Vasant Kunj, New Delhi, India-110070
kapurva@in.ibm.com

ABSTRACT
Interoperable identity and trust management infrastructure plays an important role in enabling integrations in cloud computing environments. In the past decade or so, several web-based workflows have emerged as de-facto standards for user identity and resource access across enterprises. Establishing correctness of such web protocols is of immense importance to a large number of common business transactions on the web. In this paper, we propose a framework for analyzing security in web protocols. A novel aspect of our proposal is bringing together two contrasting approaches for security protocol analysis. We use the inference construction style, in which the well-known BAN logic has been extended to reason about web protocols, in conjunction with, an attack construction style that performs SAT based model-checking to rule out certain active attacks. The result is an analysis method that shares simplicity and intuitive appeal of belief logics, at the same time covers a wider range of protocols, along with an ability to automatically find attacks. To illustrate effectiveness, case study of a leading web identity and access management protocol is presented, where application of our analysis method results in a previously unreported attack being identified.

Categories and Subject Descriptors

General Terms
Security, Verification

Keywords
Security Protocols, Belief Logic, Automated security analysis

1. INTRODUCTION
Analysis of cryptographic protocols (i.e. protocols that use cryptographic techniques for distributing keys and authenticating principals over a network) has been an active area of research over the past three decades. Even seemingly simple protocols have the reputation of being notoriously error-prone [26] when exposed to an environment where the intruder is allowed to intercept, alter, delete messages and collude with dishonest principals.

In the last decade or so, a new set of protocols has emerged that manage specific transactions on the web. The protocols are characterized by a user interacting with multiple collaborating service providers, using standard web security mechanisms over a web-browser. Examples include cross-domain web single sign-on, secure electronic payments, content sharing with third parties etc. Industrial protocols implementing these transactions are responsible for security of cross-domain collaborations on the web. Some popular web protocols that have been used to implement such transactions are Security Assertion Markup Language (SAML) [2], OpenID [3] and OAuth [4].

Approaches for security protocol analysis can be broadly classified under two categories. Inference construction approaches, first popularized by the publication of Burrows, Abadi, Needham (BAN) [1] logic, attempt to use inference in specialized logics to establish required beliefs at protocol participants. These approaches have the advantage of resulting in efficiently computable formulations. Attack construction approaches, on the other hand, use model checking tools to construct attacks by modeling an intruder and using algebraic properties of the messages being transmitted. These complexities result in such approaches suffering from state-space explosion problem. We feel that inference construction based approaches (also termed as belief logics) are ideally suited for analyzing security of web protocols. The higher abstraction level and their ability to establish what a protocol achieves (in terms of beliefs established at participants) make them attractive for analyzing security in complex web transactions.

Belief logics, however, suffer from a few significant limitations. Firstly, soundness has been challenged due to their inability of handling certain types of active attacks. In particular, protocols suffering from attacks utilizing multiple parallel executions have been declared safe using these methods. Secondly, unlike model-checking, these approaches do not automatically generate an attack trace when expected beliefs cannot be established. Finally, while there have been several extensions of the original BAN, very little work has been done in extending these approaches to the web domain and supporting new types of attacks that are introduced due to browser-based communication.

In this paper, we propose a generic approach for analyzing web protocols. We perform security analysis in two stages. At a higher abstraction level, we perform inference construction using a logic that extends BAN to facilitate reasoning about web protocols. The
reasoning makes underlying assumptions about some security properties. At a lower abstraction level, we use a model finder to determine if attacks violating the property assumed by the belief logic are allowed by the protocol.

While there have been several extensions to BAN, our logic generalizes some basic concepts of the logic, towards supporting analysis of web protocols. We recognize the need to support users without identifying keys and having identities that are not global. To further simplify analysis for the web, we introduce a primitive construct in our logic to represent an SSL/TLS [29] based secure session. We also provide a framework for representing and reasoning about user actions. Users contribute in these protocols through actions like submitting a form, signing in, accepting terms, clicking a link etc. When identities are not global, establishing that a user recently performed an action is often more important than knowing its identity.

We identify an important security property which ensures that several session based attacks can be ruled out, thus allowing the belief logic analysis to be sound. To validate the security property, we develop a general model for web protocols in Alloy [15], a SAT based model analysis tool. Checking of this property is done for a much simplified version of the original protocol and a very simple intruder model, thus drastically reducing the overall complexity of our approach in comparison with typical attack construction approaches.

We illustrate effectiveness of our approach through security analysis of the OAuth protocol, an industry standard for web-based third party delegation. The analysis illustrates how our approach allows typical advantages of belief logics to be extended to web protocols. At the same time, we demonstrate that at a marginal overhead, we get benefits of model checking approaches like coverage against a wider range of attacks and automatic generation of attack traces.

We discuss related work in Section 2 and overview of BAN logic in Section 3. In Section 4, we introduce syntax and inference rules of the proposed logic. In Section 5, we introduce the hybrid approach that uses model analysis in conjunction with the logic. Section 6 describes Alloy based web protocol modeling in detail. An example analysis of OAuth is presented in Section 7, while important contributions of the work are covered in Section 8.

2. RELATED WORK

In the previous section, we mentioned two types of approaches for security protocol analysis. In this section, we review existing work in each type of approach.

Inference construction approaches attempt to use inference in specialized logics to establish required beliefs at the protocol participants. The logic of authentication described in [1], commonly known as BAN, was one of the first successful attempts at representing and reasoning about security properties of protocols. In [6], minor improvements to the logic’s syntax and inference rules suggested to remove some ambiguity. Authors of [7] introduced the concept of ‘recognizability’. Logic in [5] introduces the concept of possession along with belief and uses it to support constructs like ‘not originated here’. In [8] authors attempt to consolidate good features from earlier belief logic approaches. These logics have the advantage of being usually decidable and efficiently computable. The logics can be easily automated. In [9], a transformation of BAN logic and inference rules to first order formula is performed and theorem prover SETHEO is used for finding proofs. In [10], the authors attempt to embed BAN logic in EVES theorem prover. However, given that a real protocol has a limited number of keys, principals and messages, forward chaining approaches discussed in [28] or the model driven analysis approach in [23] are often much simpler.

Attack construction approaches on the other hand do not try to establish beliefs at the participants but use model-checking techniques to construct attacks. The states and transitions used for modeling the protocol include modeling the structure of the message passing over the channel and a model of the intruder. The intruder is usually based on a Dolev-Yao model [11], and is allowed to perform any sequence of operations such as data interception, concatenation, de-concatenation, encryption, decryption etc. These complexities result in such approaches suffering from state-space explosion problem. However, these approaches do have the advantage of generating a counter-example corresponding to the attack, when a security property is not satisfied.

Few works that are representative of this class of approaches are mentioned below. The first such approach was introduced in [11], but the class of protocols studied in this work was very limited. In [12] the author modeled an extension of Dolev-Yao model in a specialized Prolog based model-checker, the NRL protocol analyzer. Other approaches in this area include the use of FDR model checker for CSP [13], use of SAT based model-checking techniques to solve a simplified version of the protocol insecurity problem [14] and on-the-fly model-checker (OFMC) [18], a semi-decision procedure which explores the search space system in a demand-driven way. [14] and [18] have been employed as backend model-checkers in the AVISPA tool [19] for automated validation of security protocols. The Proverif tool [30] is based on replacing the more accurate multi-set rewriting representation with abstractions that allow it to perform unbounded analysis of small to medium sized protocols. It uses an extension of applied-pi calculus as its input language. An alternative to state-based analysis is the strand-space based approach [20] which uses a graph-theoretic interpretation of Dolev-Yao model. The protocol analyzer, Athena [21] and the more recent Scyther tool [31] are based on this approach.

Since our work has elements of both attack and inference construction approaches, we now compare and contrast our work with most relevant works in each category. [23] possibly represents a first attempt at a belief logic for the web. The preliminary logic was able to analyze simple web protocols correctly. The main drawback was not being able to handle browser-based attacks e.g. cross-site request forgery. In this work, we improve the logic and augment the belief logic analysis with model checking methods. We analyze a version of OAuth different from the one analyzed in [23]. This version suffers from an attack that cannot be handled by the logic of [23].

Authors of [27] model a non-compliant version of the SAML single sign-on protocol and a standard Dolev-Yao adversary using multi-set rewrite formalism and discover an attack resulting from insufficient authentication between service providers. However, use of the standard adversary instead of a specialized browser based attacker, results in another flaw, similar to the one identified in [22] for WebAuth protocol, remaining unexposed.

Authors of [22] model a few web mechanisms using Alloy [15] and analyze them for multiple security properties. They also show how to analyze simple web protocols using their framework. For the second stage of our analysis, we also develop a generic model
in Alloy, but we are able to handle much more complex protocols since we use model analysis in conjunction with belief analysis. This results in Alloy based analysis being performed for a significantly simplified version of the original protocol.

3. OVERVIEW OF BAN

BAN statements. A formula in BAN logic [1] is constructed using operators from Table 1. 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 $\exists X$ 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 says operator.

Table 1. Operators in BAN Logic. $X$ is a statement of the logic.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \models X$</td>
<td>$P$ believes $X$</td>
<td>$P \leftrightarrow Q$</td>
<td>Shared key $K$</td>
</tr>
<tr>
<td>$P \leftarrow X$</td>
<td>$P$ sees $X$</td>
<td>$\rightarrow K Q$</td>
<td>$Q$ has public key $K$</td>
</tr>
<tr>
<td>$P \models X$</td>
<td>$P$ said $X$</td>
<td>$P \models \exists Q$</td>
<td>Shared secret $Y$</td>
</tr>
<tr>
<td>$P \models X$</td>
<td>$P$ controls $X$</td>
<td>$\exists X$</td>
<td>$X$ is fresh</td>
</tr>
<tr>
<td>$(X)_{K}$</td>
<td>$X$ encrypted by $K$</td>
<td>$(X)_{Y}$</td>
<td>$X$ combined with $Y$</td>
</tr>
</tbody>
</table>

Inference Rules. There is a set of inference rules for deriving new beliefs from old ones. E.g. the message-origin inference rule below states that if $P$ knows that $K$ is a secret key between itself and $Q$ and it sees a message $X$ encrypted by $K$, then $P$ is entitled to believe that $Q$ said $X$. Similar inference rules about public keys and shared secrets are also provided, as shown below. $K^l$ represents the private key corresponding to public key $K$.

- $P \models Q \leftrightarrow P, P \models (X)_{K}$
- $P \models Q \leftarrow X$
- $P \models Q \models \exists X$
- $P \models Q \models X$

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.

- $P \models Q \models X, P \models \exists X$
- $P \models Q \models X, P \models Q \models X$

4. BELIEF LOGIC FOR WEB PROTOCOLS

4.1 Need for Extending Belief Logics

4.1.1 Typical Web-based Workflow

In a typical web-based protocol workflow, a user interacts with web-pages presented to him by one or more providers by performing actions through a user-agent (web-browser). Examples of actions are: accessing a service by clicking on a link, submitting a form, signing in, agreeing to terms etc. Since interaction take place over the stateless HTTP protocol, application state is encoded in secrets (usually an HTTP header field called cookie) and returned to the user in response. Considering that user actions are responsible for state transitions, secrets are associated with specific user actions. Secrets are usually transferred over secure SSL/TLS channels.

In workflows involving multiple providers, the user-agent processes an HTTP redirect response requesting user to be transferred from one provider domain to another. In such collaborations, a degree of trust exists between the providers and tokens, secrets issued to a user in one domain may be acceptable in another domain. Such secrets are included in the body of an HTTP request rather than a cookie header since cookies are domain specific. With this context, we now highlight some significant departures from typical cryptographic protocols that motivate the need for a belief logic designed for the web.

4.1.2 Principals without Identifying Keys

Cryptographic protocols assume that principals possess identifying keys, either a private key (public key cryptography) or a key shared by an authentication server (shared key cryptography). Identities associated with keys are global. While service providers on the web often possess identifying keys issued by trusted authorities, end-users of web protocols typically do not possess identifying keys. Moreover, when protocols require identifying a user by name, identities established are local to a provider. In web protocols, it is also not uncommon to uniquely identify a user through an action it performs rather than a name. For these reasons, we allow principals without identifying keys:
(a) a user that has recently performed a sign-in action (and not signed out yet) at a provider is considered a principal with a named local identity, (b) a principal can also be identified by virtue of any other protocol specific action it performs, e.g. a user who recently registered at a website with certain data items.

4.1.3 Need to Model Secure Channels
In the absence of identifying keys, some other mechanism is required to associate messages with principals. Clear-text communication is ignored by belief logics since it does not identify the source. However, most web transactions make use of underlying SSL/TLS based secure channels that provide unilateral (server) authentication, confidentiality and integrity. Since security properties of transport layer security mechanisms are well understood, it greatly simplifies security analysis if these properties are assumed rather than proven for each occurrence a secure channel. Secure channels not only allow associating statements with principals without identifying keys but also ensure fixed end-points and freshness of message exchanges.

4.1.4 Goals for Web Protocols
In a cryptographic protocol, important beliefs generated at principals are those concerning appropriateness of keys and secrets. In web protocols, secrets and keys used are transparent to the user and thus rarely appear as protocol goals. Instead goals are often expressed as belief about a user or principal performing an action. E.g. ensuring that user accessing a service, is the one who performed the corresponding rule for authenticated users. R5 says that if P believes secret S to be associated with action q, and it sees user UC possessing the secret, then it believes that action q was performed by UC.

\[ P \models (S \rightsquigarrow q) \Rightarrow P \models UC \ni S \]
\[ P \models (UC \ni q) \]

R4.1 requires the believing principal to be a direct observer of the action. R5, on the other hand, allows belief about an action based on a secret associated with the action. If P believes secret S to be associated with action q, and it sees user UC possessing the secret, then it believes that action q was performed by UC.

\[ P \models (S \rightsquigarrow q) \Rightarrow P \models UC \ni S \]
\[ P \models (UC \ni q) \]

While R4.1, R5 do not require a user to be authenticated, R6 is the corresponding rule for authenticated users. R5 says that if P is connected to UC over a secure channel and believes that UC is currently signed in as Q, then P can attribute any action seen on the channel to the principal Q. We use the predicate SignedIn(UC, Q) to denote that UC is signed in as Q.

\[ P \models (S \rightsquigarrow q) \Rightarrow P \models SignedIn(UC, Q) \ni S \]
\[ P \models (Q \ni q) \]

For browser based protocols, presence of a cookie can be used to establish whether a principal is signed-in or is aware of an action associated with a cookie. We use variable name ck and constants named Ck (\( x \) is a principal name or an action) for cookies.

\[ P \models (S \rightsquigarrow q) \Rightarrow P \models SignedIn(UC, Q) \ni S \]
\[ P \models (Q \ni q) \]

Due to space limitation, we are unable to include a proof of soundness but walk through important steps here. The proof builds on semantics of BAN described in [6]. In addition to the model of [6], we associate a user with two collections: a sequence of actions executed by it and a set of actions it is allowed to assert based on possession of secrets. We introduce semantics for secure channel and use it to establish soundness of R4. Soundness of R5, R6 is established by proving if premises are satisfied then (i) an action is known to appear in one of the collections (semantics of user performing action), or (ii) an action appears in the sequence after a SignedIn action (semantics of principal performing action).

We note that our semantics assumes a user to be aware of actions it is allowed to assert. This is not always true for web protocols. In presence of cross-site request forgeries where secrets are forged, the assumption is not valid. A web protocol negates such attacks if it satisfies the redirection safety property described in Section 5.4.
5. REASONING BASED ON SECRETS IN WEB PROTOCOLS

5.1 Attacks Based on Parallel Runs

A significant limitation of belief logics has been their inability to handle certain attacks based on multiple protocol runs. In such attacks, intruder participates in multiple simultaneous executions of the protocol and uses secret values learnt from one session, in another session. A case in point is the multiple session based attack on Needham-Schroeder Public Key (NSPK) protocol first reported in [13]. The protocol was analyzed as safe in the original BAN work [1]. While this does not impact BAN’s reasoning based on signed messages, it does demonstrate an inadequacy related to reasoning based on secrets.

5.2 Request Forgery in Web Protocols

In web protocols, an additional complexity introduced while dealing with secrets is that of request forgery. In cryptographic protocols, a principal is always aware of the content of messages it sends (unless it is relaying an encrypted message for which it is not held accountable). In browser-based protocols, a user may be induced into clicking a link or submitting a form at a malicious website. Both the content of the message and the receiving endpoint can thus be controlled by an attacker. Moreover, an HTTP cookie identifying any context information (e.g. login context) is automatically inserted by the user-agent if the request is directed to a URL within its scope (defined as a combination of domain and path). The class of attacks is termed as cross-site request forgery (CSRF). Since secrets are message content and can be forged, reasoning in a logic designed for the web should include measures to ensure soundness in presence of such attacks.

5.3 Using Model Checking to Supplement Belief Logic Analysis

One way to address issues described in Sections 5.1 and 5.2 is to extend the logic to address the above scenarios. However, not only will this make the logic more complex, it goes against the spirit of inference construction approaches to build support for specific attacks. Thus, instead of extending the logic, we take the approach of augmenting belief logic analysis with attack construction approaches. The general approach works as follows. We use the belief logic to analyze the protocol first and obtain a set of beliefs established by the protocol. However, these beliefs are subject to a security property being satisfied by the protocol. The specific security property can then be checked using a model checking approach. Moreover, depending on the security property, it may be possible to use a simplified version of the protocol, to check for it.

In Section 5.4, we discuss such a security property for web protocols. The belief logic of Section 4 is designed to be used with the assumption of this property. In Section 5.5, we show that validation of the property can be performed using a much simplified version of the original protocol being analyzed.

5.4 Secure Redirection Property

Secrets are used in web protocols primarily to identify a user who has performed an action, possibly at a different place or time. Secrets can either be included in the HTTP header as a cookie, or carried in the body of the message. In the presence of request forgery, secrets carried in the body of the request cannot by themselves be used to conclusively establish beliefs at service providers.

Assumption of secure channel (HTTPS) makes protocols easier to analyze. Since most protocols carrying secret information recommend using secure HTTPS communication, the assumption is not unrealistic. An SSL/TLS session eliminates a major class of man-in-the-middle attacks. With secured sessions, in order to forge requests containing secrets, the adversary is required to be a participant in the protocol. Since authentication of service providers is analyzed using belief logic in our approach, the model checking stage needs to only rule out possibility of a malicious third party participating as a user in the protocol. Moreover, malicious user behavior e.g. tricking an honest user into joining a manipulated protocol session can only occur between secured sessions. The intruder model is thus greatly simplified. In particular, presence of an adversary, is indicated by a ‘message’ exchanged between two users (one honest and one dishonest). The communication could happen through an asynchronous user action e.g. clicking a malicious hyperlink sent by the attacker, or an automated browser action e.g. redirection to a manipulated callback URI. If we can establish that the protocol does not allow user-to-user communication, we say that it satisfies the secure redirection property.

5.5 Protocol Simplification

Establishing security property of Section 5.4 essentially involves associating two or more secure sessions in the protocol with the same web user. In the absence of identifying keys, possession of secrets and cookies is the only means to establish this. Since this analysis only requires reasoning based on possession of secrets and cookies, it can be done on a much simplified version of the protocol in which only messages containing secret values exchanged with a user are retained. The rules for simplification described below are tabulated in Table 3.

Rule 1: A protocol message that is not sent or received by a user can be removed, i.e. server to server communication is ignored.

Rule 2: Any term which does not contain a secret or nonce value is dropped.

Rule 3: In the remaining messages, an encrypted formula \(\{X\}_K\), received at a user is represented by an opaque token \(N_{ik}\) if the decryption key can be assumed to be unavailable to the intruder. Otherwise, in addition to the token, the formula is also included (with or without encryption as shown in Table 3).

Rule 4: Similarly, an encrypted formula \(\{X\}_k\) received from a user is interpreted as the user possessing the corresponding token \(N_{ik}\).

<table>
<thead>
<tr>
<th>Formula</th>
<th>Decryption Key in Intruder Knowledge</th>
<th>Transformed Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>({X}_k)</td>
<td>Always (X, N_{ik}, K)</td>
<td>(N_{ik}^{-1})</td>
</tr>
<tr>
<td>({X}_k)</td>
<td>Never (N_{ik})</td>
<td>(N_{ik})</td>
</tr>
<tr>
<td>({X}_k)</td>
<td>Always (X, N_{ik}, K)</td>
<td>(N_{ik})</td>
</tr>
<tr>
<td>({X}_k)</td>
<td>Never (N_{ik})</td>
<td>(N_{ik})</td>
</tr>
<tr>
<td>({X}_k)</td>
<td>No assumption (X, N_{ik}, K)</td>
<td>(N_{ik})</td>
</tr>
</tbody>
</table>

\(K\): public key, \(K^{-1}\): private key, \(K_{ik}\): shared key.
6. MODELING WEB PROTOCOLS USING ALLOY

6.1 Overview of Alloy

Alloy [15], [17], [29] is 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:

\[
\begin{align*}
\text{sig } S & \text{ extends E } \{ \\
F & \text{: one } T \\
\} \\
\text{fact } \{ \\
\text{all } s:S & \text{ | s.F in X } \\
\} \\
\end{align*}
\]

It is often useful to think of Alloy as an object-oriented language, e.g. S is a class (s being an instance), that extends superclass E. F is a member of S pointing to T. However, under 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 \( \text{quantifier } s:S \ | \ F \text{ mean that constraint } F \text{ holds for all, } \text{no, } \text{one } (\text{zero or one}), \text{some } (\text{at least one}) \text{ or } \text{one } \text{ of } S. \text{ Fact expressions that apply to a particular signature (as is the case above) can be directly appended to the signature within curly brackets. Predicate (pred \{ ... \}) and functions are optional facts that can be conditionally invoked. Assertions (assert \{ ... \}) are properties against which the specification needs to be checked. A run command causes the analyzer to search for consistency of a function or predicate, while a check command causes it 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. Exceptions can be specified for specific signatures. E.g. the below command tries to find a counter-example for assertion acyclic with default scope up to 5, but up to 2 fileSystems and 7 FSObjects.

\[
\begin{align*}
\text{check acyclic for } 5 & \text{ but } 2 \text{ fileSystem, } \\
\text{exactly } 7 & \text{ FSObject} \\
\end{align*}
\]

We also use a utility ordering to define an order on elements of a signature. The function greater than or equal (gte) defined in ordering can be used in an expression such as the one shown in example below to specify vals with times \( \geq \text{time} \).

\[
\begin{align*}
\text{open util/ordering[Time] as ord} \\
\text{..} \\
\text{all t: Time | t in vals } \leftrightarrow t.\text{ord/gte[time]} \\
\end{align*}
\]

6.2 General Model for Web Protocols

We now describe our general Alloy model that allows reasoning about secrets for a wide range of web protocols.

**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.

\[
\begin{align*}
\text{abstract sig Process} & \{ \\
\text{knownkeys: set Key} \} \\
\text{sig Time} & \{ \} \text{ sig Key} \{ \} \text{ sig Value} \{ \} \\
\text{sig TkValue extends Value} \{ \} \\
\text{sig CkValue extends Value} \{ \}
\end{align*}
\]

\[
\begin{align*}
\text{sig Cookie} & \{ \\
\text{value: one CkValue, server: one Server} \} \\
\text{sig PvtKey extends Key} & \{ \\
\text{pubkey: one Key} \} \{ \text{pubkey != this} \}
\end{align*}
\]

\[
\begin{align*}
\text{sig Server extends Process} & \{ \\
\text{ownedkey: one PvtKey, peer: one Server,} \\
\text{uniqueval: one TkValue,} \\
\text{uniquecookie: one Cookie} \\
\} \{ \text{peer != this, uniquecookie.server = this} \}
\end{align*}
\]

\[
\begin{align*}
\text{sig User extends Process} & \{ \\
\text{seentokens: set TkValue->Time,} \\
\text{knowncookies: set Cookie->Time}\} \{ \ldots \}
\end{align*}
\]

\[
\begin{align*}
\text{sig HUser extends User} & \{ \}
\end{align*}
\]

\[
\begin{align*}
\text{fact } \{ \text{all k1,k2: PvtKey|k1 != k2 =>} \\
k1.\text{pubkey != k2.pubkey} \}
\end{align*}
\]

\[
\begin{align*}
\text{fact } \{ \text{all s1,s2: Server|s1 != s2 =>} \\
(s1.\text{uniqueval != s2.uniqueval}) \& \& \\
(s1.\text{ownedkey != s2.ownedkey}) \& \& \\
(s1.\text{uniquecookie})!= (s2.\text{uniquecookie}) \}
\end{align*}
\]

User participates in two relations. seentokens associates the user to a set of (value, time) pairs each indicating that a value was known to user at time. The relation knowncookies provides a similar association for cookies known to the user.

Finally, the two facts represent constraints on private keys and servers. Two private keys may not be mapped to the same public key. Similarly each server must own a unique private key and should generate distinct nonces and cookies.

**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. If a key is used to encrypt the message it is identified using the enckey relation. The first constraint says that a message sent by a user can only contain cookies that are known to 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.
sig URL { target: one Process }

sig Sent {
  cookies: set Cookie, sender: one Process, receiver: one Process, time: one Time, content: set TkValue, redirectURL: lone URL, enckey: lone Key }

{} 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. 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.

**Intruder 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 is modeled as a redirect message generated by the dishonest user.

### 7. ANALYSIS OF OAUTH PROTOCOL

#### 7.1 Protocol Description

![Figure 1. The OAuth Protocol (RFC version)](image)

The OAuth protocol [4] provides a web based workflow that allows a user to temporarily delegate privileges of his account at a provider to a third party without sharing his login credentials. Privileges could for example mean access to pictures, friend list, blogs etc. OAuth is the primary protocol used by Google, Facebook, Twitter to allow third party access to user content.

The original version, OAuth Core 1.0 [24] is known to suffer from a session-fixation attack and was analyzed in [23]. In this paper, we analyze the revised version (Core 1.0a) also an approved IETF RFC [4], *OAuth 1.0 Protocol*. We also refer to this as the *RFC version* of OAuth. Workflow for the RFC version of OAuth protocol shown in Figure 1 is described below.

**Steps 1-4**, user requests service $S$ from consumer (C). The service requires a set of privileges (permissions) Priv to the user account at provider (P). Consumer registers delegation request with P and gets returned a request token $N_b$. C redirects user to P with this token. **Steps 5-10**, user is requested to sign in and delegate set of privileges Priv to C. User signs in as principal Q and performs

---

1 We note that some providers like Google, Facebook have moved to OAuth 2.0 [25] which bears little resemblance with the original protocol and is not analyzed here. Other providers e.g. Twitter have chosen to stay with the IETF approved version [4].
requested delegation. User is redirected back to C with the request token and another verifier token, Np. Steps 11–14, C uses tokens Ns, Np to request a protected resource directly from P. User receives requested service S in step 14. All communication happens over secure SSL/TLS channels and requests from consumer are signed and verifiable at the provider. The secure channels used are identified as C1–C5 in Figure 1.

7.2 Belief Logic Analysis of OAuth
The protocol is idealized as shown below. Only messages received by either C or P are idealized because we are interested in beliefs at these principals. There are two named variables in the idealized protocol: scope representing the set of privileges to be delegated and callback identifying the URL used by P for redirection in step 10.

Message 1 \( U_{c1} \rightarrow C : \llbracket \text{Request}(S) \rrbracket_{c1} \)
Message 2 \( C_{c2} \rightarrow P : \llbracket \{ N_s, \text{scope} = \text{Priv}, \text{callback} = \text{url}_r \} \rrbracket_{c2} \)
Message 3 \( P \rightarrow C_{c2} : \llbracket \{ N_s, \text{callback} = \text{url}_r, \text{scope} = \text{Priv} \} \rrbracket_{c2} \)
Message 5 \( U_{c3} \rightarrow P : \llbracket \{ N_p \} \rrbracket_{c3} \)
Message 7 \( U_{c3} \rightarrow P : \llbracket \text{SignIn}(Q) \rrbracket_{c3} \)
Message 9 \( U_{c3} \rightarrow P : \llbracket \text{Delegate}(\text{scope}, C), C_{k_q} \rrbracket_{c3} \)
Message 11 \( U_{c4} \rightarrow C : \llbracket \{ N_s, N_p \} \rrbracket_{c4} \)
Message 12 \( C_{c5} \rightarrow P : \llbracket \{ C, N_2, N_s, N_p, p1 \} \rrbracket_{c5} \)
Message 13 \( P \rightarrow C_{c5} : \llbracket \{ N_p \rightarrow \text{Delegate}(\text{scope}, C) \} \rrbracket_{c5} \)

Idealization of messages 1, 7 and 9 represents user actions performed in the protocol. Messages 2 and 12 are direct requests from C to P for request token and protected resource respectively. In message 2, the set of privileges Priv, for which delegation is required, is included. In message 12, a protected resource requiring a privilege p1 \( \in \text{Priv} \) is requested. Ns and Np are nonces (combination of a timestamp and nonce in actual protocol). Cookie Ckq represents login context for user signed in as principal Q at P. When P returns the requested resource in message 13, it conveys to C that the verifier token corresponds to a valid delegation action.

Apart from the more obvious assumptions about secure channels (C1–C5), knowledge of public keys, and freshness of nonces (N1 and N2), we also make the following assumptions.

\[ C \models N_p \rightarrow \text{Request}(S) \]
\[ P \models N_p \rightarrow \text{Delegate}(\text{Priv}, C) \]
\[ C \models (\forall x, r, P \models x \rightarrow \text{Delegate}(r, C), C) \]
\[ P \models C_k_q \rightarrow \text{SignIn}(Q) \]
\[ C \models (\forall y, r, P \models y \rightarrow \text{Delegate}(r, C)) \]

These include association of secrets Ns, Np and cookie Ckq with the user actions and C’s complete trust in P for the delegation action. The goal of the protocol is as specified below. (G1) ensures that the delegation was performed by the user who has previously performed the sign-in action at provider. (G2) is required to ensure that recipient of the service at Consumer in step 14 has also performed the delegation action.

\[ P \models \quad Q : \text{Delegate}(\text{priv}, C) \]  \quad (G1)
\[ C \models U_{c4} \rightarrow \text{Delegate} \quad (\text{priv}, C) \]  \quad (G2)

We now present our belief logic analysis. The forward chaining based analysis can be easily automated using existing approaches like [28], [23] as mentioned in Section 2.

Message 2: OAuth token request received at P.
Combining received message with assumption about C’s public key using R1, followed by application of rule R2 using assumption about freshness of Ns.

\[ P \models \{ N_s, \text{scope} = \text{Priv}, \text{callback} = \text{url}_r \} \]
\[ P \models \{ N_s \} \]
\[ P \models \{ \text{Request}(S) \} \]
\[ P \models \text{Delegate}(\text{priv}, C) \]

Message 9: Delegation Action seen by P.
First apply R7 to establish that UC3 is signed in as Q using presence of cookie and assumption about Ckq. Next, associate observed action with Q using R6.

\[ P \models \text{SignedIn}(U_{c3}, Q) \]
\[ P \models \text{Delegate}(\text{Priv}, C) \]

Message 11: Establish that UC4 \( \supseteq \text{Np} \) and UC4 \( \supseteq \text{Np} \)

Message 13: Use R4.3 to establish that P believes about validity of Np and then use C’s trust in P for delegation action (R3). Finally, combine with conclusion of message 11 to infer that UC4 has performed the delegation action.

\[ C \models \{ N_p \rightarrow \text{Delegate}(\text{scope}, C) \} \]
\[ C \models P \models \text{Delegate}(\text{priv}, C) \]
\[ C \models \{ \text{scope} = \text{Priv} \} \]

From analysis of messages 9 and 13, we find that goals G1 and G2 are satisfied. However, we still need to establish that the secure redirection property assumed by the belief logic holds for the protocol.

7.3 Checking Redirection Security using Alloy
In the protocol of Figure 1, messages 5 and 11 are the only messages demonstrating possession of secret with a user. The two secrets considered in the analysis are Ns and Np. Applying protocol simplification rules of Section 5.5, we obtain the following simplified protocol. For each message, the message sequence number in the original protocol of Figure 1 is indicated in parentheses.

Message 1(4) \( C \rightarrow U : \llbracket N_s, \text{url}_r \rrbracket \)
Message 2(5) \( U \rightarrow P : \llbracket N_s \rrbracket \)
Message 3(10) \( P \rightarrow U : \llbracket N_s, N_p, \text{url}_r \rrbracket \)
Message 4(11) \( U \rightarrow C : \llbracket N_s, N_p \rrbracket \)

Note that we assume that url points to C in our model due to the belief \( P \models C \models \text{callback} = \text{url}_r \) established from analysis of message 2 in Section 7.2.

We use the general Alloy model described in Section 6.2. To add constraints corresponding to this protocol, we declare two new signatures Producer and Consumer by extending Server.
We also add following protocol specific constraints to the signature ProtoSeq:

**Constraint 1:** \( P \) must send message 3 containing verifier token and redirection URL in response to receiving message 2.

\[
\text{all } p: \text{Sent} \mid (p.\text{receiver in Provider}) \Rightarrow \\
\text{some } q: \text{Sent} \mid (p \rightarrow q \text{ in sequence}) \& \& \\
(q.\text{content} = p.\text{content} + p.\text{receiver}.\text{uniqueval}) \& \& \\
(q.\text{redirectURL}.\text{target} = q.\text{sender}.\text{peer})
\]

**Constraint 2:** First message in protocol sequence must be sent by Consumer and must contain a secret (request token) and redirection URL pointing to the Producer.

\[
\text{one } p: \text{Sent} \mid (\text{some } t1: \text{Sent} \mid p \rightarrow t1 \text{ in sequence}) \& \& \\
(t1.\text{sender in Consumer}) \& \& (t1.\text{content} = t1.\text{sender}.\text{uniqueval}) \& \& \\
(t1.\text{cookies} = t1.\text{sender}.\text{uniquecookie}) \& \& (t1.\text{time} = \text{ord}/\text{first}[]) \& \& (t1.\text{redirectURL}.\text{target} = t1.\text{sender}.\text{peer})
\]

**Constraint 3:** Last message in the protocol must be received by the consumer and must contain request token and the verifier.

\[
\text{one } p: \text{Sent} \mid (\text{some } t1: \text{Sent} \mid t1 \rightarrow p \text{ in sequence}) \& \& \\
(t1.\text{receiver in Consumer}) \& \& (t1.\text{content} = t1.\text{receiver}.\text{uniqueval} + \\
t1.\text{receiver}.\text{peer}.\text{uniqueval})
\]

We execute Alloy analyzer with these additional constraints and an assertion that there is only one User and it is honest (\( H_{\text{User}} \)). Alloy generates the counter example shown in Figure 2 in less than 4 seconds on an Intel Core i5 2.4 GHz, 4 GB system. The counter-example clearly shows possibility of two users, one of which is honest. Also there are five messages instead of expected four. The sequence is determined by the value of the time attribute. The value \( \text{TkValue}4 \) represents the request token (\( N_b \)) while \( \text{TkValue}0 \) is the verifier (\( N_p \)). Note that the redirection constraint of Section 6.2 only applies for an honest user (\( H_{\text{User}} \)). We see that a (dishonest) user can perform steps 1, 2, 3 of the protocol (messages \( \text{sent}0, \text{sent}4, \text{sent}2 \)) and then create a message for the honest user (\( \text{sent}3 \)) who then sends message 4 to \( C \) (\( \text{sent}1 \)). This translates to the following attack on the original protocol.

**Previously unreported attack on OAuth (RFC version).** An attacker performs steps 1-10 of the protocol and delegates access to its account \( X \) at \( P \) for a limited period of time to \( C \). However, instead of getting redirected to \( C \), it induces a victim - having a valid account \( V \) at \( C \) – to click a link that contains both request token and verifier. On clicking the link, \( V \) is transferred to \( C \), where it either starts a new login session or continues with an existing session. \( C \) believes that the valid request token and verifier are for \( V \)’s account at \( P \), while they are actually associated with the attacker’s account \( X \). If \( V \) accesses a service at \( C \) that requires information to be shared with a remote account at \( P \) (e.g. backing up an address book), \( C \) inadvertently releases the sensitive information to the attacker.

**Fixing the protocol.** The protocol can be easily fixed by requiring consumer to include a cookie along with the request token while redirecting user to provider in step 4. It can then check for presence of this cookie when the user returns in step 11. This ensures that steps 4 and 11 are performed by the same user.

When we execute the analyzer again after changing constraints 2 and 3 in the Alloy model to include the cookie, we are not presented with a counter-example even for a scope allowing a protocol sequence of up to 20 messages.

**8. CONCLUSIONS**

We consider the problem of providing a generic framework for security analysis of web protocols. Belief logics are known to be fast and cost effective tools for this purpose and we feel these techniques could be even better suited for the web domain. However, there are known issues with multiple session attacks and new challenges are introduced due to browser-based communication.

We propose a novel two stage approach where belief logic analysis is followed by automated model finding. The two stages
are linked through a security property which is assumed in the first stage and validated by the second stage. The belief logic we use is our extension of BAN logic for the web domain. It supports principals without identifying keys, secure SSL/TLS channels and simple reasoning about user actions, thus enabling and simplifying analysis of web protocols.

We develop a generic model for web protocol analysis in Alloy. Since scope of this analysis is to check for a particular security property, we make considerable simplifications in both the protocol and intruder models. This significantly reduces complexity of model checking in our approach. We demonstrate our hybrid method through analysis of OAuth, a leading web identity and access management protocol. We identify a previously unreported vulnerability in an approved RFC version which is still being used by several service providers. We also propose a simple fix that service providers can use to overcome the insecurity. Use of lightweight analysis methods makes it practical for our method to be incorporated in design, development of web protocols and standards.

9. REFERENCES